Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

IMPLEMENTACIÓN DE UN SERVIDOR DE FICHEROS VÍA BLUETOOTH

Alumno: **Antonio Lozano Vida**

Tutor: Prof. D. José Ángel Fernández Prieto
Depto.: Ingeniería de Telecomunicación

Septiembre, 2014

INDICE

1. Memoria

1.1. Objeto

El objeto principal de este proyecto es el de desarrollar una aplicación para la implementación de un servidor de ficheros basado en la tecnología Bluetooth. Dicha aplicación se ejecutará sobre un PC con sistema operativo Windows y que dispondrá de un adaptador Bluetooth. Para esto se utilizará el lenguaje de programación Java en su plataforma J2SE (*Java 2 Standard Edition*) y la implementación de la API JSR-82 *BlueCove*.

El enfoque práctico de la aplicación desarrollada será el de dar soporte al envío bajo demanda o automático de ofertas comerciales por parte de un establecimiento a sus clientes. El envío de una oferta consistirá en la transferencia de un fichero al Smartphone de los clientes que se encuentren en el radio de cobertura del adaptador Bluetooth, utilizando para ello el protocolo de intercambio de objetos OBEX. Tanto las ofertas, como los clientes registrados serán gestionados por el administrador de la aplicación, quien podrá crearlas, modificarlas, eliminarlas, dar de baja clientes, etc.

En la *figura 1.1* se aprecia un esquema del objetivo del proyecto:

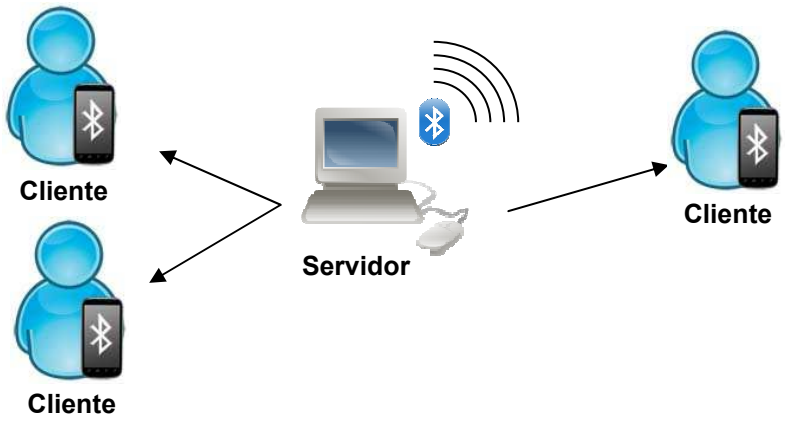


Figura 1.1: Esquema de la aplicación desarrollada

1.2. Alcance

El alcance del presente proyecto consistirá en el desarrollo y entrega de la aplicación software ejecutable, cumpliendo los requisitos enunciados en el objetivo, así como la redacción de un manual de usuario, instalación y mantenimiento lo suficientemente claros para que el administrador de la aplicación pueda ponerlo en marcha y utilizarlo sin mayor problema.

Queda fuera del alcance del proyecto la instalación física de la aplicación así como la adquisición de materiales necesarios para ello.

1.3. Antecedentes

La tecnología Bluetooth se ha convertido en un estándar en las comunicaciones inalámbricas, siendo empleada en enlaces de radio de corto alcance. Esta tecnología está destinada a reemplazar el cableado existente entre dispositivos electrónicos tanto móviles como estáticos, tales como teléfonos móviles, ordenadores, tablets, dispositivos de audio o vídeo, impresoras, etc. permitiendo conexiones instantáneas de voz y datos en tiempo real entre los dispositivos conectados. Además, esta tecnología utiliza un modo de transmisión que asegura protección contra interferencias y seguridad durante la transmisión de los datos.

En este apartado se realizará una introducción a esta tecnología, describiendo las características más relevantes de la misma. Se estudiarán las nociones principales de ella, algunas especificaciones técnicas y estándares, sus diferentes y múltiples usos y aplicaciones, etc.

La necesidad de dotar al usuario de una mayor libertad en el uso de las comunicaciones, partía de la eliminación de barreras como los cables para la comunicación entre dispositivos.

El nombre de la tecnología Bluetooth procede del rey danés *Harald Blätand* (Harald II), del cual se dice que tenía grandes facultades comunicativas que lo hicieron famoso. Este rey del siglo X reunificó bajo su mandato numerosos pequeños reinos que existían en Noruega y Dinamarca, iniciando el proceso de cristianización de la sociedad.

La historia del Bluetooth es relativamente corta. Fue la compañía de telecomunicaciones Ericsson la que comenzó un estudio en el año 1994 para investigar la posibilidad de implementar una interfaz vía radio de baja potencia y coste reducido entre teléfonos móviles y sus accesorios. El objetivo de dicho estudio era eliminar los cables entre los teléfonos móviles y los demás accesorios que éstos utilizaban.

En Febrero de 1998 se formó el Grupo de Interés Especial o SIG (*Special Interest Group*), que inicialmente estuvo formado por las compañías, Ericsson, IBM, Nokia, Toshiba e Intel, aunque más tarde se invitó a otras compañías para participar en el grupo. Dicho grupo trabaja para desarrollar, promover, definir y publicar las especificaciones de esta tecnología inalámbrica a corto alcance dando como resultado la publicación de la versión 1.0 de las especificaciones Bluetooth en julio de 1999.

Las compañías inscritas en el SIG podían dotar de Bluetooth a sus productos con la garantía que ofrece el pertenecer al grupo y conocer las especificaciones técnicas de la

tecnología mientras que las demás compañías externas no podían aplicar esta tecnología al no tener su patente. Por este motivo el grupo de compañías pertenecientes al SIG creció hasta llegar a más de 20.000 miembros en el año 2014.

La última versión de la especificación Bluetooth es la 4.0, adoptada en junio de 2010.

Paralelamente, las empresas creadoras de software, se interesaron en el desarrollo de lenguajes y medios para programar los dispositivos que utilizaban esta tecnología, ya que hasta el momento no había manera de desarrollar aplicaciones personalizadas para los usuarios.

De esta forma Java creó la API JSR-82, que estandarizó la forma de desarrollar aplicaciones Bluetooth usando Java. Gracias a esta API, es posible centrarse en el desarrollo de una aplicación en lugar de en los detalles de bajo nivel que esconde la complejidad del protocolo Bluetooth.

1.3.1. Estado del arte

Para dar solución al problema planteado por el proyecto se ha tenido que realizar un estudio previo sobre el estado actual de la tecnología Bluetooth, así como de las diferentes herramientas existentes hoy en día para abordar de manera óptima la solución del mismo.

De esta forma, después de un exhaustivo proceso de documentación, se describirán en este apartado los aspectos fundamentales de la tecnología Bluetooth como son la seguridad, la protección contra interferencias, ventajas y desventajas frente a otras tecnologías inalámbricas, la arquitectura de protocolos que utiliza, redes de área personal (PAN), etc.

Se describirán además las herramientas de desarrollo seleccionadas para la programación de la aplicación, así como la API de Java utilizada, las diferentes implementaciones que existen de la misma, etc. Se introducirán las nociones básicas de manejo de la API, describiendo los paquetes principales que la componen, así como las funciones y métodos utilizados para gestionar los dispositivos Bluetooth.

1.3.2. Introducción a la tecnología Bluetooth y Piconets

¿QUÉ ES BLUETOOTH?

Bluetooth es una tecnología que se ha convertido en un estándar global en las comunicaciones inalámbricas, la cual posibilita la transmisión de voz y datos entre diferentes equipos tanto móviles como estáticos mediante un enlace por radiofrecuencia, de manera que no necesita visión directa entre los aparatos que se conectan. Esta

tecnología se diseñó pensando básicamente en tres objetivos: tamaño reducido, mínimo consumo y bajo coste.

De esta forma, gracias al Bluetooth, se hace factible la posibilidad de formar redes de comunicación con casi cualquier dispositivo electrónico que implemente la especificación Bluetooth, ya sean ordenadores de sobremesa, portátiles, teléfonos móviles, tablets e incluso electrodomésticos.

CARACTERÍSTICAS PRINCIPALES

- Se trata de una **tecnología inalámbrica** que reemplaza la conexión cableada en distancias de hasta 100 metros (según la clase de dispositivo utilizado), ando al usuario una libertad total de comunicación.
- **Bajo consumo de potencia**, ya que las reducidas dimensiones de los dispositivos y su portabilidad requieren de un uso adecuado de la energía.
- **Bajo costo**. El coste de fabricación de los chips es bastante económico, siendo el precio de venta aproximado unos 10 € con tendencia a bajar.
- **Integración de servicios**. Puede soportar transmisiones de voz y datos de manera simultánea entre los dispositivos que implementen la tecnología.
- **Transmisión omnidireccional**. Debido a que utiliza radiofrecuencia como base de la comunicación, no requiere de vista directa entre los dispositivos implicados.
- **Seguridad en la transmisión**. Con el fin de evitar interferencias con otras tecnologías que operen en la misma banda de frecuencias, Bluetooth emplea la técnica de salto de frecuencias (FHSS, Frequency Hopping Spread Spectrum), lo que disminuye el riesgo de que las comunicaciones sean interceptadas o presenten interferencia con otras aplicaciones. Provee también especificaciones para autenticar dispositivos que intenten conectarse a una red Bluetooth, así como cifrado para proteger la información.
- **Configuración de redes**. Tiene la posibilidad de formar redes en una topología donde un dispositivo hace las veces de maestro pudiendo haber hasta otros siete más operando como esclavos. Esta configuración se conoce como piconet.

BLUETOOTH VS INFRARROJO

Aunque ambas tecnologías especifican una comunicación inalámbrica a corta distancia, hoy en día Bluetooth está muy por encima de las aplicaciones de infrarrojo por las claras ventajas que provee, las cuales se deducen de sus propias características.

- El infrarrojo requiere de una comunicación lineal entre transmisor y receptor, lo que hace imprescindible la línea de vista para su efectiva transmisión.

- Las frecuencias de la banda del infrarrojo no permiten la penetración a través de paredes, dándole una importante ventaja a la radiofrecuencia con la que opera Bluetooth.
- La comunicación con infrarrojo siempre será uno a uno, dejando de lado las configuraciones punto-multipunto.
- Bluetooth permite la generación de redes.

BLUETOOTH VS Wi-Fi

Después de años de lucha por el liderato en el ámbito de las comunicaciones inalámbricas, Bluetooth y Wi-Fi han sabido convivir en el mundo de la tecnología y cada una de ellas se propone con una solución inalámbrica específica.

Muchas compañías creen en la naturaleza complementaria de Bluetooth con Wi-Fi, y en su papel conjunto, a la hora de desarrollar una gama complementaria de opciones de conectividad universal para los usuarios.

Mientras que Bluetooth fue diseñado para intercambios rápidos de información con bajo ancho de banda, Wi-Fi es la tecnología adecuada a redes LAN, para la transferencia de archivos que requieran gran ancho de banda o para conexiones a la red o a Internet permanentes.

ASPECTOS TÉNICOS DE LA TECNOLOGÍA BLUETOOTH

Bluetooth trabaja a una frecuencia de radio situada en el rango de 2.4 a 2.48 GHz en la banda ISM (*Industrial, Scientific and Medical*), la cual está disponible a nivel mundial y no requiere licencia de operador, lo que posibilita una compatibilidad universal entre dispositivos Bluetooth.

La **velocidad de transmisión** varía según la versión del núcleo utilizado:

- Versión 1.1: 723.1 Kbps
- Versión 1.2: 1 Mbps
- Versión 2.0 + EDR: 2.1 ~ 3 Mbps
- Versión 3.0 + HS 24 Mbps
- Versión 4.0 24 Mbps

La **potencia de transmisión** se varía según la clase del dispositivo. Actualmente se definen tres **clases de dispositivos** Bluetooth:

- Clase 1: 100 mW / 20 dBm, con un radio de alcance de ~100 m.
- Clase 2: 2.5 mW / 4 dBm, con un radio de alcance de ~10 m.
- Clase 3: 1 mW / 0 dBm, con un radio de alcance de ~1 m.

VERSIONES DE LA ESPECIFICACIÓN

A partir de la versión 1.0, que apareció en julio de 1999, se han publicado sucesivas versiones, las cuales han ido mejorando multitud de factores relevantes como la velocidad de transmisión, la calidad, la potencia media consumida, etc. A continuación se presenta un resumen de dichas mejoras según la versión de la especificación:

Versión 1.1

- Soluciona erratas de la especificación 1.0.
- Añade el Indicador de Calidad de Señal Recibida (RSSI).

Versión 1.2

- Implementa la técnica de salto en frecuencia, Adaptive Frequency Hopping, para mejorar la resistencia a interferencias.
- Introduce el tipo de enlace para aplicaciones de audio extended Synchronous Connections (eSCO) que mejora la calidad de voz.
- Mejoras en el Host Controller Interface (HCI) para una sincronización más rápida de las comunicaciones.
- A diferencia de la 1.1, provee una solución inalámbrica complementaria para coexistir Bluetooth y WiFi en el espectro de los 2.4 GHz, sin interferencia entre ellos.

Versión 2.0

- Nueva versión compatible con la anterior 1.x.
- Incorpora la tecnología Enhanced Data Rate (EDR), que incrementa las velocidades de transmisión hasta 3 Mbps.
- Reducción del consumo de energía a pesar del incremento de velocidad.

Versión 2.1

- Compatibilidad con la 1.2.
- Mejora la experiencia de emparejamiento de dispositivos.
- Reducción considerable del consumo de energía.

Versión 3.0

- Compatibilidad con la 1.2.
- Aumento de la velocidad de transferencia hasta los 24 Mbps.
- Uso de estándar 802.11 para transferencias de alta velocidad.

Versión 4.0

- Compatibilidad con la 3.0.
- Reducción drástica del consumo de energía
- Aumento de la seguridad en la transferencia de información

TIPOS DE DISPOSITIVOS (APLICACIONES Y USOS)

Desde la aparición de la tecnología, el intento de la mayoría de las empresas productoras de dispositivos electrónicos de adaptar sus productos a la misma ha sido constante. Lo que inicialmente se creó pensado para la comunicación entre teléfonos móviles y sus periféricos, a fecha de hoy se ha extendido a casi todos los ámbitos de nuestra vida cotidiana. La tecnología Bluetooth permite el intercambio de información entre dispositivos de distinta naturaleza que cumplan las especificaciones del estándar por lo que hoy en día pueden encontrarse multitud de productos que integran conectividad mediante Bluetooth, desde electrodomésticos hasta vehículos que ya lo traen preinstalado de serie.

En la *figura 1.2* se ilustran algunos ejemplos de los diferentes tipos de dispositivos existentes en el mercado que incorporan tecnología Bluetooth.

De izquierda a derecha se pueden observar auriculares stereo, manos libres personales y sistemas integrados en vehículos, módulos GPS, ordenadores portátiles con Bluetooth integrado, adaptadores USB Bluetooth, puertas de acceso a otras redes (gateways), teclados y ratones inalámbricos, impresoras, smartphones, tablets, cámaras de fotos, cámaras de video y proyectores.

Figura 1.2: Tipos de dispositivos que pueden incorporar tecnología Bluetooth

SEGURIDAD

Una de las características a destacar de la tecnología Bluetooth es que incorpora varios mecanismos de seguridad que lo convierten en uno de los protocolos de comunicaciones más seguros y robustos frente a ataques, interferencias y capturas de datos. Así pues tenemos:

Seguridad a nivel de banda base: Salto de frecuencias (FHSS, *Frequency Hopping Spread Spectrum*)

Con el objetivo de evitar interferencias con otras tecnologías que operen cerca de la misma banda de frecuencias, Bluetooth utiliza una técnica de salto de frecuencias consistente en dividir la banda utilizada en 79 canales (23 en España, Francia y Japón) de 1 MHz de longitud y realizar 1600 saltos por segundo. En la *figura 1.3* se aprecia un esquema de la división de la banda en los canales.

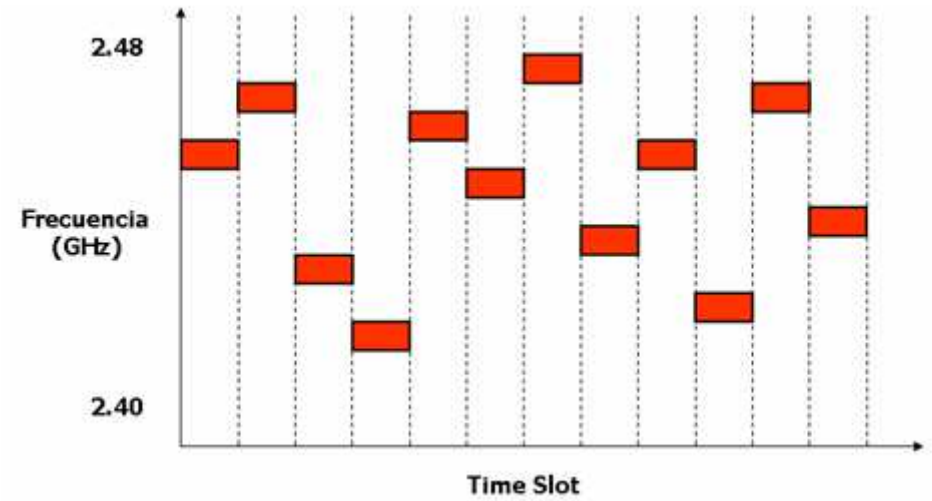


Figura 1.3: División de la banda en canales para el salto de frecuencia

El proceso mediante el cual un módulo Bluetooth se sincroniza con el resto de dispositivos implicados en una comunicación utilizando la técnica de salto de frecuencias se resume de la siguiente manera:

- En la fase inicial de establecimiento de la conexión en una piconet, el dispositivo que toma el papel de maestro genera una tabla pseudoaleatoria con una secuencia o patrón de saltos de frecuencia, la cual deberán utilizar los dispositivos pertenecientes a la piconet durante las comunicaciones que se establezcan. El intercambio de la tabla de saltos desde el maestro hacia el resto de dispositivos implicados se realiza en un canal determinado del espectro de frecuencias, de forma que todos los dispositivos pueden acceder a ésta.

- Una vez que un dispositivo esclavo detecta la piconet, recibe un paquete FHS (Frecuency Hop Synchronization) que le permite sincronizar su reloj interno con el reloj del maestro agregando un desplazamiento a su reloj interno.
- A continuación, una vez iniciada la comunicación, el intercambio de paquetes de datos se realiza de acuerdo con el patrón de saltos de frecuencia establecido y a una velocidad marcada por el reloj interno. Esto quiere decir que en cada instante de tiempo cada dispositivo escribirá o escuchará durante su timeslot en un determinado canal del espectro.
- De esta forma, cualquier dispositivo ajeno que no pertenezca a la piconet no podrá participar en la comunicación enviando paquetes o escuchando tráfico, ya que no dispone de la tabla con la secuencia de saltos utilizada en la piconet.
- En resumen, la técnica de saltos de frecuencia empleada por Bluetooth garantiza, en principio, la participación exclusiva de dispositivos autorizados en una piconet y una comunicación libre de escuchas por parte de usuarios ajenos a la misma.

Hay que decir también, que al igual que el resto de tecnologías para la transmisión de información, Bluetooth también es susceptible de sufrir ataques provenientes de dispositivos o entidades maliciosas capaces por ejemplo de estar escuchando constantemente y capturar las tablas de saltos de frecuencia, ya que como se ha mencionado anteriormente el intercambio de estas tablas se lleva a cabo en una frecuencia conocida. De esta forma el dispositivo podría sincronizarse con la red establecida. No obstante, se trataría de módulos Bluetooth especiales que no resultan accesibles para la mayoría de los usuarios y cuyo coste es elevado.

Seguridad a nivel de enlace:

- **Autenticación:** La autenticación es el proceso por el que un dispositivo Bluetooth verifica su identidad en otro dispositivo remoto para poder optar a los servicios que este ofrece. Este mecanismo implica un esquema de pregunta / respuesta entre cada par de dispositivos, empleando una clave de enlace secreta común de 128 bits. Consecuentemente, este esquema se utiliza para autentificar dispositivos y no los usuarios.

  Esto evita problemas en sistemas de baja seguridad, en los que los dispositivos Bluetooth utilizados no tengan implementada una interfaz de usuario para permitir al mismo introducir un código de seguridad, como ocurre por ejemplo dispositivos GPS o manos libres, los cuales suelen incorporar un código prefijado de fábrica, como 0000, 1234, etc.

- **Autorización:** La autorización es el procedimiento por el que determina los privilegios y derechos que tiene un dispositivo Bluetooth para acceder a los servicios que ofrece un sistema.

  El mecanismo de autorización en dispositivos Bluetooth se lleva a cabo mediante *niveles de confianza*. Los dispositivos tienen tres niveles de confianza, los cuales determinan la capacidad de acceso a los servicios: total, parcial o restringida y nula. Casi todos los dispositivos Bluetooth suelen disponer de una base de datos interna con su lista de dispositivos de confianza.
- **Cifrado de datos:** Este mecanismo protege la información que se transmite en una conexión de dispositivos Bluetooth, garantizando la confidencialidad del mensaje transmitido, de forma que si el paquete es capturado por un usuario que no posea la clave de descifrado, el mensaje le resultará ininteligible.

  Su utilización es opcional, pero necesita que se haya producido anteriormente una autenticación. Los dispositivos implicados en la comunicación deben ponerse de acuerdo en utilizar cifrado o no.

EL MODELO DE DATAGRAMA DE BLUETOOTH

En Bluetooth todos los paquetes de datos (datagramas) están compuestos por tres secciones: 72 bits para los códigos de acceso, 54 bits para el encabezado del datagrama o cabecera y 2745 bits para datos. Con lo cual se obtienen datagramas de un tamaño máximo de 2871 bits, de los cuales el 96% están destinados para transporte de datos.

El campo de cabecera contiene alguna información de control, como tres bits de acceso de dirección, tipo de paquete, bits de control de flujo, bits de retransmisión automática de la pregunta, y chequeo de errores de campos de cabeza.

En la *figura 1.4* se aprecia un esquema del modelo de datagrama utilizado por Bluetooth.

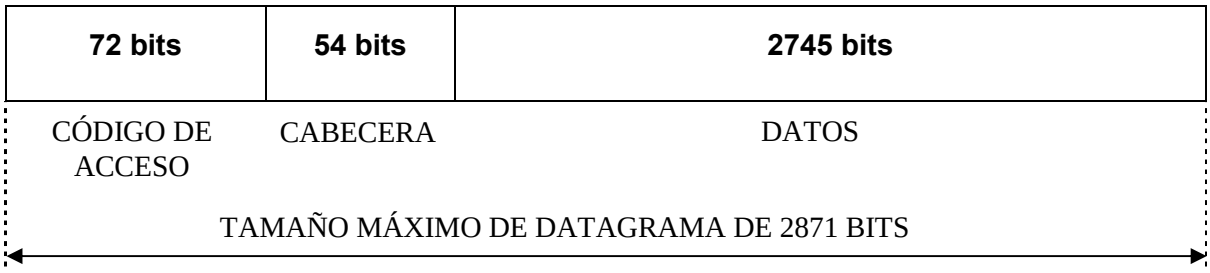


Figura 1.4: Modelo de datagrama utilizado por Bluetooth

ARQUITECTURA DE PROTOCOLOS DE BLUETOOTH

La pila o *stack* de protocolos Bluetooth está basada en el modelo de referencia OSI (*Open Systems Interconnection*) de ISO (*Internacional Standard Organization*) para interconexión de sistemas abiertos. La arquitectura de protocolos que se define en la especificación Bluetooth divide las diversas funciones de red en un sistema de niveles. Estas divisiones permiten en conjunto el intercambio transparente de información entre aplicaciones diseñadas de acuerdo con dicha especificación, fomentando de esta forma la interoperabilidad entre los productos de diferentes fabricantes, siempre que estos cumplan el estándar definido en la arquitectura descrita.

A continuación se puede observar la arquitectura de protocolos utilizada por Bluetooth con los niveles y divisiones mencionados anteriormente:

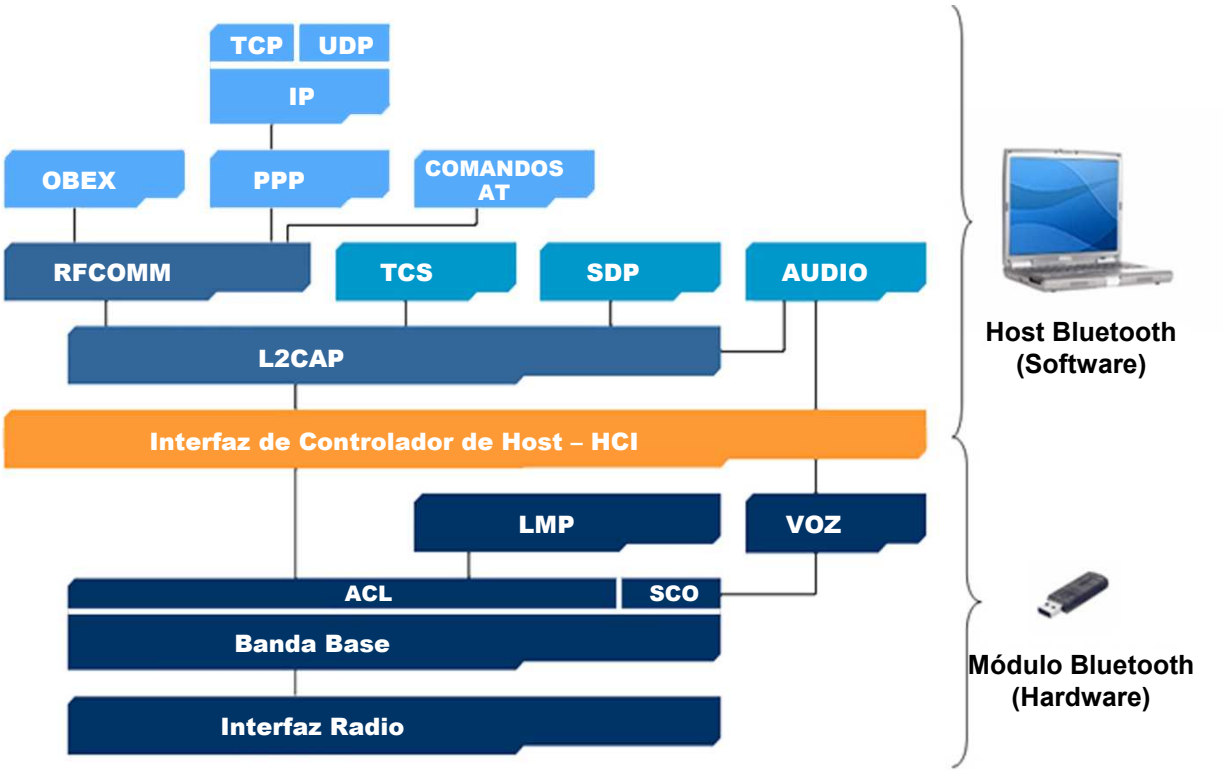


Figura 1.5: Arquitectura de protocolos de Bluetooth

Como anteriormente hemos mencionado, la pila de protocolos Bluetooth se divide en dos zonas:

- El **módulo Bluetooth** (hardware), encargado de las tareas relacionadas con el envío de información a través del interfaz de radiofrecuencia.
- El **host Bluetooth** (software), encargado de la parte relacionada con las capas superiores de enlace y aplicación.

Capa de banda base e interfaz de radio:

En la base de la pila de protocolos Bluetooth se encuentran la capa de banda base y el interfaz de radio. Su función principal es permitir el enlace físico por radiofrecuencia (RF) entre unidades Bluetooth dentro de una piconet realizando tareas de modulación y demodulación de los datos en señales RF que se transmiten por el aire.

El nivel de banda base proporciona dos tipos de enlace físico:

- Enlace asíncrono sin conexión (ACL, *Asynchronous Connectionless*).
- Enlace síncrono orientado a conexión (SCO, *Synchronous Connection-Oriented*).

Capas de Audio y Voz:

En realidad no se trata de unas capas definidas en la arquitectura de protocolos de Bluetooth, sin embargo se suelen incluir ya que son comúnmente tratadas en las comunicaciones por Bluetooth. Los datos de audio y voz son típicamente enrutados hacia y desde la capa de banda base y sobre el enlace SCO. En el caso de que un canal de datos esté en uso, los datos de audio serán transmitidos sobre el enlace ACL.

Capa de Protocolo de Gestión de Enlace (LMP):

LMP (*Link Manager Protocol*) es el protocolo responsable de la configuración y control de enlace entre dispositivos Bluetooth, incluyendo el control y negociación del tamaño de los paquetes de banda base.

Capa de Interfaz de Controlador de Host (HCI):

La capa HCI (*Host Controller Interface*) actúa como frontera entre las capas de protocolo relativas al hardware (módulo Bluetooth) y las relativas al software (host Bluetooth). Proporciona una interfaz de comandos para la comunicación entre el dispositivo y el firmware del módulo Bluetooth, permitiendo de esta forma disponer de una capa de acceso homogénea para todos los módulos Bluetooth de banda base, aunque sean de distintos fabricantes.

Capa de Protocolo de Adaptación y Control del Enlace Lógico (L2CAP):

La especificación Bluetooth incluye el protocolo L2CAP (*Logical Link Control and Adaptation Protocol*), que se encarga de la multiplexación de protocolos, ya que el protocolo de banda base no soporta un campo *tipo* para identificar el protocolo de nivel superior al que quiere transmitir la información, por ejemplo SDP, RFCOMM y TCS. Otra función que se realiza en el nivel L2CAP es la segmentación y recomposición de

paquetes, necesaria para permitir la utilización de protocolos que utilicen paquetes de mayor tamaño que los soportados por la capa de banda base.

Capa de Protocolo de Descubrimiento de Servicios (SDP):

El protocolo SDP (*Service Discovery Protocol*) permite a una aplicación cliente obtener información sobre servidores SDP disponibles en otros dispositivos Bluetooth cercanos, enumerar los servicios que ofrecen y las características de dichos servicios. Después de haber localizado los servicios disponibles en un dispositivo, el usuario puede elegir aquel de ellos que resulte más apropiado para el tipo de comunicación que desea establecer.

Capa TCS - Binary (*Telephony Control Specification*):

Esta capa especifica cómo se deben manejar las llamadas telefónicas de datos y conversaciones en conjunto con los teléfonos móviles. Esto es especialmente importante para aplicaciones que manejan dispositivos como manos libres o audífonos.

Capa RFCOMM:

El protocolo RFCOMM (*Radio Frequency Communication*) es un protocolo de emulación de línea serie que proporciona una emulación de los puertos serie RS-232 sobre el protocolo L2CAP. Este protocolo emula las señales de control y datos RS-232 sobre la banda base, proporcionando capacidades de transporte a los servicios de niveles superiores que utilizan el cable serie como mecanismo de transporte. RFCOMM pretende soportar aquellas aplicaciones que utilizan los puertos serie de los dispositivos donde se ejecutan.

Comandos AT:

Los *comandos AT*, también conocidos como comandos *Hayes*, son instrucciones que conforman un lenguaje de comunicación entre un usuario y un terminal módem.

La telefonía móvil GSM también ha adoptado como estándar este lenguaje para poder comunicarse con sus terminales. De esta forma, todos los teléfonos móviles GSM poseen un juego de comandos AT específico que sirve de interfaz para configurar y proporcionar instrucciones a los terminales.

De esta forma, podemos:

- Configurar el teléfono para una conexión inalámbrica, a través de Bluetooth o por el sistema de bus o cable.
- Configurar el módem interno del teléfono para una conexión inalámbrica, a través de Bluetooth o por el sistema de bus o cable.

- Solicitar información sobre la configuración actual o estado operativo del teléfono o módem.
- Probar la disponibilidad del teléfono o módem.
- Solicitar el rango valido de parámetros aceptados y cuando éstos son aplicables.

Existe un juego extenso de comandos AT que permiten multitud de funciones. Estos pueden encontrarse en la documentación técnica de los terminales GSM.

Capa TCP/UDP, IP y PPP:

Este conjunto de capas, también conocido como WAP (*Wireless Application Protocol*), define un protocolo de aplicación para normalizar el modo en que los dispositivos inalámbricos se pueden utilizar para acceder a servicios como Internet, correo electrónico, grupos de noticias, etc.

Capa de intercambio de objetos OBEX (Object Exchange):

Se trata de un protocolo de transferencia que define *objetos de datos* (ficheros por ejemplo) y un protocolo de comunicación mediante el que dos entidades pueden intercambiarlos de forma muy sencilla.

Es un protocolo adaptado del protocolo de Infrarrojos IrOBEX, ya que las dos capas inferiores del mismo son muy similares a las dos capas inferiores de la torre de protocolos de Bluetooth. Aunque OBEX puede ser soportado sobre TCP/IP, esto no está descrito en la especificación de Bluetooth.

Utilizando este protocolo, se definen los papeles de cliente y servidor en las entidades implicadas. Así pues, todas las transacciones realizadas mediante OBEX, deberán ser inicializadas por el cliente.

Sobre la arquitectura de protocolos específicos de Bluetooth, cada fabricante puede implementar su capa de protocolos de aplicación propios. De esta forma, la especificación abierta de Bluetooth expande enormemente el número de aplicaciones que pueden beneficiarse de las capacidades que ofrece esta tecnología inalámbrica. Sin embargo, la especificación Bluetooth exige que, a pesar de la existencia de diferentes pilas de protocolos de aplicación propias, se mantenga la interoperabilidad entre dispositivos que implementen diferentes pilas.

Las pilas de protocolos Bluetooth más conocidas son *Widcomm, Toshiba Bluetooth Stack, Microsoft Bluetooth Stack* y *IVT BlueSoleil Stack.* Linux dispone de las pilas de protocolos Bluetooth *BlueZ, OpenBT y Affix, de Nokia.*

PERFILES O SERVICIOS BLUETOOTH

Un *servicio*, también llamado un *perfil*, es cualquier entidad que puede ofrecer información, ejecutar una acción o controlar un recurso. Un servicio puede estar implementado como hardware, software o una combinación de hardware y software.

Un perfil o servicio Bluetooth consiste en una lista de las opciones, características o funcionalidad de un dispositivo Bluetooth determinado.

Existen varios modelos de uso del estándar de comunicaciones Bluetooth definidos por el SIG. Cada uno de estos modelos está acompañado por un perfil. Los perfiles definen los protocolos y características que soportan un modelo de uso particular. Esto garantiza la interoperabilidad, ya que si dos dispositivos de distintos fabricantes cumplen con la misma especificación del perfil Bluetooth, podemos esperar que interactúen correctamente cuando se utilicen para un uso particular.

Los perfiles se clasifican dentro de modelos de uso. Dentro de cada modelo existen uno o más perfiles que definen las diferentes capas del protocolo Bluetooth.

Un perfil Bluetooth define los mensajes específicos y procedimientos usados para implementar un modelo de uso. Algunas características son obligatorias y algunas pueden ser opcionales.

La misma naturaleza del equipo determina que perfiles ha de cumplir. Por ejemplo, un auricular cumplirá el perfil de auricular (*headset*) y el de acceso genérico (GAP), pero no el de puerto serie (SPP) o el de propiedades de marcado en red (DUN), que corresponderían a una tarjeta adaptadora para PC.

Así pues, se definen:

- **Perfiles genéricos**: Se definen cuatro perfiles genéricos que contienen la especificación de los perfiles específicos siguientes: el Perfil de Acceso Genérico (GAP, Generis Access Profile), el Perfil de Puerto Serie (SPP, Serial Port Profile), el Perfil de Aplicación de Descubrimiento de Servicios (SDAP, Service Discovery Application Profile) y el Perfil Genérico de Intercambio de Objetos (GOEP, Generic Object Exchange Profile).
- **Perfiles específicos**: Sobre los anteriores se definen los diferentes perfiles específicos para cada modelo particular de uso. Estos perfiles Bluetooth para modelos de uso son múltiples y variados, y se implementan de manera opcional e independiente por cada fabricante y tipo de dispositivo.

  Algunos de los perfiles más comunes más comunes son:

  - Perfil de Telefonía Inalámbrica (CTP, Cordless Telephony Profile).
  - Perfil de Intercomunicación (IP, Intercom Profile).
  - Perfil de Puerto Serie (SP, Serial Port Profile).

- Perfil de Acceso Telefónico a Redes (DUN, Dial-Up Networking).
- Perfil de Auriculares (HS, HeadSet Profile).
- Perfil de Fax (FP, Fax Profile).
- Perfil de Acceso a Red (LAP, LAN Access Profile).
- Perfil de Transferencia de Archivos (FTP, File Transfer Profile).
- Perfil de Carga de Objetos (OPUSH u OPP, Object Push Profile).
- Perfil de Sincronización (Sync, Synchronization Profile).

Aunque hoy en día, muchos otros perfiles han sido aprobados por el SIG o están en fase de desarrollo, como por ejemplo:

- ESDP, Extended Service Discovery Profile.
- A2DP, Advanced Audio Distribution Profile.
- AVRCP, Audio Video Remote Control Profile.
- BIP, Basic Imaging Profile.
- BPP, Basic Printing Profile.
- CIP, Common ISDN Access Profile.
- GAVDP, Generic Audio Video Distribution Profile.
- HFR, Hands-Free Profile.
- HCRP, Hardcopy Cable Replacement Profile.
- HID, Human Interface Device Profile.
- PAN, Personal Area Networking Profile.
- SAP, SIM Access Profile.

Un servicio concreto soportado por cierto dispositivo es una instancia de un *Service Class* o clase de servicio. El *Service Class* de un dispositivo describe los servicios genéricos soportados por el mismo.

Para dar a conocer los servicios genéricos que soporta un dispositivo Bluetooth, este incorpora en la cabecera de nivel de banda base de sus paquetes un campo *Class of Device/Service* que contiene información acerca de su *Service Class*. Conociendo el Class of Device/Service de un dispositivo Bluetooth, se puede averiguar fácilmente el conjunto de servicios genéricos soportados por el mismo.

TOPOLOGÍA DE REDES BLUETOOTH

Cuando el usuario posee una red personal para la comunicación entre distintos dispositivos como ordenadores, puntos de acceso a Internet, teléfonos móviles, PDA, dispositivos de audio, impresoras, etc., se dice que ha creado una **Red de Área Personal** o PAN (*Personal Area Network*). Este tipo de redes son de unos pocos metros y para uso personal.

Cuando los dispositivos que conforman una Red de Área Personal se comunican mediante Bluetooth, a esta red se le llama ***Piconet***. Este tipo de red estará compuesta por un mínimo de 2 dispositivos activos en relación jerárquica (*maestro - esclavo*) y un máximo de 8. El primer dispositivo en formar una piconet asumirá el papel de maestro, mientras que el resto funcionarán como esclavos del anterior, dependiendo de él en todo momento. Normalmente este tipo de red suele tener un rango de alcance de 10 o 15 metros, aunque en condiciones ideales podría llegar hasta los 100 metros.

A un grupo de piconets se le llama ***Scatternet***. Dado que la especificación Bluetooth soporta tanto conexiones punto a punto como punto a multipunto, se pueden establecer y enlazar varias piconets dando lugar a una scatternet. Debido a que individualmente cada piconet tiene un salto de frecuencia diferente, diferentes piconets pueden usar simultáneamente diferentes canales de salto. Todos los dispositivos que participan en la misma piconet deben sincronizarse con su correspondiente tiempo de reloj y patrón de saltos determinado. El resto de piconets utilizarán diferentes patrones de saltos y frecuencias de relojes distintas, lo que supone distintas velocidades de salto entre canales. Sin embargo, un mismo dispositivo no podrá sincronizarse al mismo tiempo con diferentes piconets, para ello, deberá participar de forma secuencial en cada una de ellas, estando activo en sólo una piconet cada vez.

En la *figura 1.6* se aprecia un esquema ilustrativo de las topologías mencionadas, así como las relaciones jerárquicas existentes entre los dispositivos implicados en las mismas.

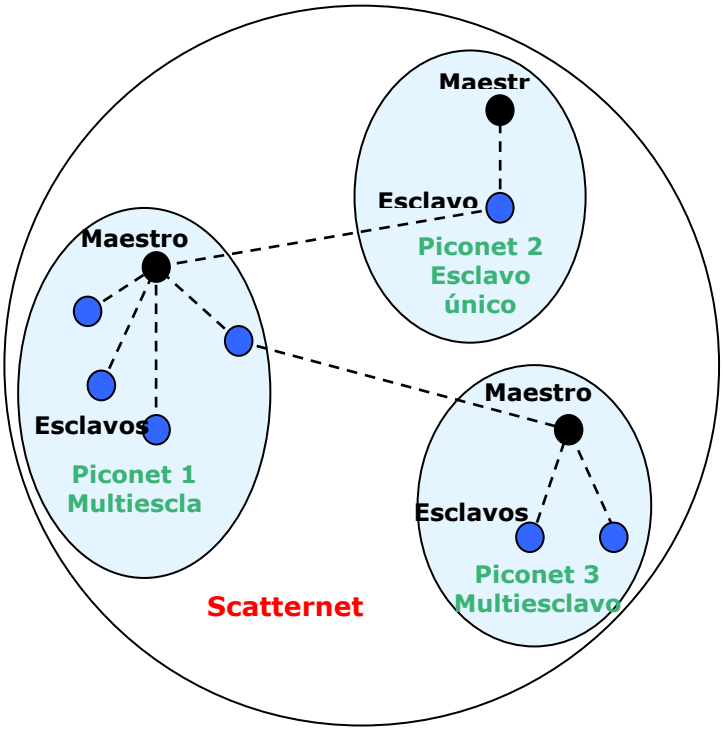


Figura 1.6: Topología y dependencias en una red Bluetooth

1.4. Normas y referencias

1.4.1. Disposiciones legales y normas aplicadas

No es de aplicación ninguna disposición legal o norma para la ejecución del presente proyecto.

1.4.2. Programas de cálculo

No ha sido precisa la realización de cálculos complejos para la ejecución del presente proyecto, por lo que tampoco se han utilizado programas para ello.

1.4.3. Plan de gestión de calidad aplicado durante la redacción del proyecto

Para la redacción del proyecto se ha seguido la norma *UNE 157001:2014* sobre *Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico.*

1.4.4. Bibliografía

Se ha utilizado y consultado la siguiente bibliografía para la ejecución y redacción del presente proyecto:

1. Bruce Hopkins, Ranjith Antony. “Bluetooth for Java”. Ed. Apress, 2003
2. Gimeno Brieba, Alberto. “Bluetooth desde Java”. 2004 [En línea] *http://jderobot.org/index.php?title=Special:Upload&wpDestFile=PFC_Miguel_II_Articulo_Bluetooth_JSR-82.pdf* (Última visita: septiembre de 2014)

1.4.5. Otras referencias

Se han utilizado y consultado las siguientes referencias para la ejecución y redacción del presente proyecto:

1. Página oficial de la tecnología Bluetooth. [En línea] *http://www.bluetooth.com (*Última visita: septiembre de 2014).
2. Wikipedia [En línea] *http://es.wikipedia.org/wiki/Bluetooth* (Última visita: septiembre de 2014).
3. Web oficial de la API BlueCove. [En línea]. *http://bluecove.org/* (Última visita: septiembre de 2006).
4. Web de la API JSR-82 oficial de Java. [En línea] http://docs.oracle.com/javame/config/cldc/opt-pkgs/api/bluetooth/jsr082/ (Última visita: septiembre de 2014).

5. Página oficial de Java. [En línea] *https://www.java.com/es/* (Última visita: septiembre de 2014).
6. Página oficial de NetBeans. [En línea] *https://netbeans.org/* (Última visita: septiembre de 2014).

1.5. Definiciones y abreviaturas

Puesto que la mayoría de abreviaturas utilizadas a lo largo de la memoria corresponden a protocolos o partes de ellos, habiendo sido la mayoría acompañadas de su significado, únicamente resumiremos en este apartado aquellas que se han utilizado con mayor recurrencia:

- ACL: Asynchronous Connectionless
- API: Application Programming Interface
- Banda ISM: Industrial, Scientific and Medical
- FHSS: Frequency Hopping Spread Spectrum
- HCI (Host Controller Interface
- ISO: Internacional Standard Organization
- J2ME: Java 2 Micro Edition
- J2SE: Java Platform, Standard Edition
- JCP: Java Community Process
- JRE: Java Runtime Environment
- JSR: Java Specification Requests
- L2CAP: Logical Link Control and Adaptation Protocol
- LMP: Link Manager Protocol
- OBEX: Object Exchange Protocol
- OSI: Open Systems Interconnection
- PAN: Personal Area Network
- RFCOMM: Radio Frequency Communication
- SCO: Synchronous Connection-Oriented
- SDK: Software Developer's Kit
- SDP: Service Discovery Protocol
- SIG: Special Interest Group
- TCS-Binary: Telephony Control Specification
- WAP: Wireless Application Protocol

1.6. Requisitos de diseño

El cliente nos plantea la necesidad de la elaboración de un software para acometer el envío de ficheros desde la sede de su establecimiento comercial hacia los dispositivos móviles de los clientes que se encuentren en las inmediaciones. Los ficheros a enviar representarán las ofertas comerciales que se crearán, debiendo estos enviarse mediante tecnología Bluetooth.

La aplicación deberá ser automatizable, en el sentido de que deberá poder realizar búsquedas de dispositivos y envíos de ofertas de manera autónoma, simplemente con ponerla en funcionamiento.

Los condicionantes de hardware serán básicos. La aplicación deberá correr sobre un PC con Sistema Operativo Windows, el cual estará dotado de un adaptador Bluetooth mediante USB. Así mismo, la aplicación deberá tener una interfaz de usuario amigable para facilitar su utilización.

1.7. Análisis de soluciones

A lo largo de este apartado describiremos las herramientas y posibles soluciones a seguir para la consecución de los objetivos marcados en el proyecto. Se hará una introducción a las herramientas actuales para el desarrollo de software destinado a controlar dispositivos Bluetooth, donde se verá las ventajas que nos ofrece Java y el protocolo para el desarrollo de estas aplicaciones gracias a su API JSR-82, desarrollada por el JCP (*Java Community Process*). Se expondrán sus posibles implementaciones a utilizar, características, ventajas y desventajas, limitaciones, etc... Se realizará un análisis de los distintos paquetes que componen la API que finalmente utilizaremos para el desarrollo de la aplicación y sus clases más importantes.

1.7.1. Bluetooth y Java

La programación de dispositivos Bluetooth es relativamente nueva, aunque actualmente ya existen diferentes alternativas a la hora de desarrollar este tipo de aplicaciones.

En el año 2000, el *Java Community Process* (JCP), lanzó la API **JSR-82**, que estandarizó la forma de desarrollar aplicaciones Bluetooth usando Java. Gracias a esta API, es posible centrarse en el desarrollo de aplicaciones para controlar dispositivos Bluetooth sin necesidad de sumergirnos en la complejidad y detalles de su protocolo. Fue

la compañía Motorota la que, junto con el *Expert Group* (grupo de expertos) de Java, crearon el que hoy es la **API oficial de Java para Bluetooth**.

De esta forma, utilizando la API JSR-82, el programador queda a un nivel superior del que quedaría en el caso de utilizar otros medios para el desarrollo de una aplicación.

El objetivo de esta especificación era definir una API estándar abierta, no propietario que pudiera ser usado en todos los dispositivos que implementasen la plataforma J2ME (*Java 2 Micro Edition*).

Las APIs JSR-82 son muy flexibles, ya que permiten trabajar tanto con aplicaciones nativas Bluetooth como con aplicaciones Java Bluetooth.

A pesar de esto, ***¿qué ventajas presenta el uso de la API Bluetooth de Java frente a otros lenguajes nativos como C/C++?***

INDEPENDIENTE DEL HARDWARE Y LA PILA DE PROTOCOLOS

Una de las principales razones es que la API JSR-82 es independiente de la pila y del hardware Bluetooth. Esto nos da la capacidad de escribir aplicaciones sin ningún conocimiento del hardware Bluetooth subyacente o de la pila de protocolos utilizada. De esta forma podemos escribir código estándar de Java sin tener que utilizar ningún método nativo, pudiendo hacerlo funcionar prácticamente en cualquier plataforma hardware y en cualquier sistema operativo.

API ESTANDARIZADA

Al ser una API estandarizada, todos los fabricantes están sujetos a cumplir con el estándar definido para la API de Java. Así pues, independientemente del SDK (Software Developer's Kit) para Bluetooth que se utilice, la metodología de programación será la misma, cosa que no ocurre con los demás lenguajes como C/C++, en los que al no existir ningún estándar para un SDK Bluetooth basado en ellos, cada vendedor es libre de nombrar a las funciones y métodos como prefiera.

La API JSR-82 de Java, fue desarrollada orientado a dispositivos que cumplan las siguientes características:

- Al menos 512 k de memoria libre (ROM y RAM) (las aplicaciones necesitan memoria adicional).
- Conectividad a la red inalámbrica Bluetooth.
- Que tengan una implementación del J2ME CLDC.

Existen algunos **casos en los que no se recomienda el uso de Java para programar dispositivos Bluetooth**. Esto es cuando se precise programar:

- Un Indicador de fuerza de una señal.
- Aplicaciones de voz.
- Un medidor de distancias.

La API intenta ofrecer las siguientes capacidades:

- Registro de servicios.
- Descubrimiento de dispositivos y servicios.
- Establecer conexiones RFCOMM, L2CAP y OBEX entre dispositivos.
- Usar dichas conexiones para mandar y recibir datos (las comunicaciones de voz no están soportadas).
- Manejar y controlar las conexiones de comunicación.
- Ofrecer seguridad a dichas actividades.

1.7.2. Implementaciones de la API JSR-82

En un principio, se puede suponer, que la API JSR-82, al ser una API de Java, es totalmente portable a cualquier tipo de plataforma y sistema operativo. Sin embargo, esta API ha sido desarrollado orientado a su utilización sobre dispositivos móviles que lleven implementada la plataforma J2ME con soporte para CLDC (*Connected Limited Device Configuration*).

En el caso de que quisiéramos desarrollar una aplicación para establecer una comunicación Bluetooth entre varios dispositivos móviles como teléfonos o PDAs no tendríamos ningún problema a la hora de utilizar la API original de Java. Sin embargo el propósito de este proyecto no es el de realizar una conexión entre teléfonos móviles, sino entre ellos y un PC que implemente la plataforma Standard de Java (J2SE).

Entonces, ***¿qué ocurre si queremos desarrollar una aplicación Java para comunicarnos desde un PC con dispositivos móviles?*** Si buscamos información en Internet sobre utilizar la API JSR-82 sobre J2SE, por ejemplo en los foros de la página oficial de Java, veremos que mucha gente lo ha intentado sin éxito.

Si queremos realizar una conexión Bluetooth desde un ordenador común, éste deberá tener instalado un adaptador Bluetooth. Normalmente estos adaptadores se conectan directamente al puerto USB, aunque también los hay que se conectan a otras interfaces como el puerto serie. El mayor problema radica en que al utilizar la API de Java, éste debe de poder comunicarse con el dispositivo local instalado en la máquina en la que se ejecuta el programa.

Sin embargo, por si solo la API de Java no es capaz de realizar esta función, precisando por tanto una serie de librerías nativas que funcionen de puente entre la API y

los controladores Bluetooth utilizados por el sistema operativo en el que se ejecute la aplicación. Así pues, dependiendo del sistema operativo y del controlador Bluetooth que utilice el dispositivo local, se deberá utilizar una API JSR-82 específica.

Este fue uno de los mayores problemas que se nos presentó a la hora de comenzar el desarrollo de la aplicación que propone este proyecto. Existe muy poca documentación en castellano, prácticamente ninguna, que hable sobre la utilización de la API JSR-82 sobre J2SE. En multitud de foros sobre programación Java aparece gente solicitando ayuda con frases como *"No puedo inicializar mi dispositivo local"* o *"cómo programar un dispositivo Bluetooth USB"*.

Actualmente, existen varias implementaciones de la API JSR-82 adaptadas para su utilización sobre J2SE. La mayoría de estas implementaciones son de pago y algunas de ellas presentan serias limitaciones con respecto a la API original de Java, como por ejemplo el no soportar el protocolo de intercambio de objetos OBEX.

Las dos implementaciones más conocidas y utilizadas para el desarrollo de aplicaciones Bluetooth sobre J2SE son *AvetanaBluetooth* y *BlueCove*. Mientras que AvetanaBluetooth es de licencia gratuita únicamente para sistemas operativos Linux, BlueCove lo es tanto para Linux como MAC OS y Windows.

Aunque la licencia de Avetana para Windows es bastante económica (25 €), el fabricante entrega la API compilada con las direcciones Bluetooth de los dispositivos del usuario, de manera que no sería posible utilizar la aplicación con otros dispositivos adquiridos posteriormente. A pesar de ello, la licencia de Avetana para Windows incorpora varios cambios gratuitos de las direcciones Bluetooth cuando el usuario lo estime oportuno.

No obstante, debido a su gratuitidad y mayor tendencia de utilización será BlueCove la API seleccionada para el desarrollo de la aplicación.

1.7.3. BlueCove

BlueCove es una implementación de código abierto del API JSR-82 sobre J2SE para sistemas operativos *Windows, Linux y MAC OS*. Esta implementación se encuentra actualmente en la versión 2.1.0 y está disponible para descarga en su *web oficial*[1]. Actualmente soporta hasta la versión 64bits de Windows 7, pero igualmente es compatible con versiones anteriores como Windows XP SP2.

Dependiendo de la pila de protocolos utilizada, BlueCove da soporte a más o menos componentes de la API original de Java, además, existen diferentes bugs y errores que aún están por corregir dependiendo de la pila utilizada, los cuales pueden ser consultados en la web oficial de la API. Por ejemplo, en la pila de protocolos de Microsoft

[1] Web oficial de la API BlueCove. http://bluecove.org/

para Windows (*Microsoft Bluetooth Stack*) no están soportadas las comunicaciones *l2cap* y presenta algún que otro error esporádico a la hora de descubrir dispositivos. A pesar de esto, la API BlueCove para Windows posee la suficiente funcionalidad y estabilidad para poder desarrollar nuestro programa, teniendo además la ventaja de ser gratuito y poder ejecutar la aplicación sobre Windows XP o 7.

Aunque en un principio no será necesario, pues utilizaremos la pila de Microsoft, si deseamos utilizar una pila de protocolos diferente, deberemos copiar la librería oportuna en el directorio *"Windows\System32"* de la unidad donde tengamos instalado el sistema operativo para que la API funcione correctamente. Por lo tanto, es importante cerciorarnos de que la pila de protocolos utilizada por el sistema operativo para controlar el dispositivo Bluetooth es la pila de Microsoft. Esta verificación es importante ya que muchos dispositivos suelen venir con un CD de *drivers* para su instalación. En este caso, si se instalan estos *drivers*, los que utiliza el Sistema Operativo Windows quedarán anulados, por lo que la API dejaría de funcionar.

1.7.4. Aspectos básicos de la API JSR-82

NOTA: Las descripciones siguientes se realizarán de acuerdo con la API original de JAVA. BlueCove, al igual que otras implementaciones de la API, presenta ciertas diferencias y limitaciones con respecto al original, las cuales fueron descritas en el apartado 1.7.3.

Como ya se mencionó anteriormente, la API JSR-82 define un estándar para la programación de dispositivos Bluetooth utilizando Java. Esta API **está compuesta por dos paquetes**, los cuales dependen de la configuración CLDC de J2ME:

- **Paquete javax.bluetooth**: Este paquete define las clases e interfaces básicas para el descubrimiento de dispositivos, descubrimiento de servicios, establecimiento de conexiones y gestión de la comunicación. La programación utilizando este paquete se realiza a bajo nivel. La comunicación se establece mediante flujos de datos y arrays de bytes.
- **Paquete javax.obex**: Utilizando este paquete podemos manejar el protocolo de alto nivel OBEX. Este protocolo es muy parecido a HTTP y es utilizado comúnmente para operaciones de intercambio de ficheros. Al ser un protocolo de alto nivel, el utilizarlo facilita muchísimo la tarea del programador a la hora de manejar las comunicaciones, por ejemplo, a la hora de enviar un fichero, con este paquete bastaría con crear el objeto pertinente y enviarlo al dispositivo destino con el comando PUT, en cambio, con el paquete javax.bluetooth, la transferencia del fichero se debería implementar a un

nivel inferior, debiendo previamente avisar al dispositivo destinatario y poniéndose de acuerdo con él para la transacción.

Para el desarrollo de la aplicación se utilizarán ambos paquetes dependiendo de la función específica a realizar por el programa en cada momento, así pues se utilizará el paquete *javax.bluetooth* para realizar las búsquedas de dispositivos y el paquete *javax.obex* para realizar el envío de los ficheros a los dispositivos descubiertos.

Los dispositivos implicados en una comunicación Bluetooth, podrán participar en la misma en el rol de servidor o de cliente. Las funciones que desempeña cada uno de ellos son las siguientes:

Funciones de un cliente en una comunicación Bluetooth:

- Búsqueda de dispositivos en el radio de alcance que se encuentren en modo conectable.
- Búsqueda de servicios en los dispositivos encontrados.
- Establecimiento de la conexión con el dispositivo que ofrece el servicio deseado.
- Comunicación con el servicio, una vez que la conexión se ha establecido.

Funciones de un servidor en una comunicación Bluetooth:

- Crear una conexión servidora para que los clientes puedan detectarla y conectarse a ella.
- Especificar los atributos del servicio creado.
- Atender las conexiones entrantes de los clientes.

Es importante matizar que en nuestro programa, el *servidor de ofertas* que envía los ficheros hará las funciones de *cliente* en la comunicación, mientras que los *clientes* que reciban las ofertas harán las funciones de *servidores*, pues serán ellos quienes ofrezcan la conexión servidora que rastreará el programa y a la que se conectará para hacer la transferencia.

1.7.5. El paquete javax.bluetooth

Este paquete está compuesto por una serie de clases, bien definidas y detalladas. Utilizando estas clases y los métodos que poseen, podemos gestionar una comunicación Bluetooth de comienzo a fin.

El primer paso para crear una comunicación Bluetooth será el de inicializar el dispositivo local (dispositivo Bluetooth USB en nuestro caso). Una vez inicializado el

dispositivo local, obteniendo el objeto con el que nos referiremos a él. A través de este objeto podremos seguir realizando el resto de operaciones.

El paso siguiente dependerá de si el usuario desea crear una conexión servidora o si por el contrario desea lanzar un cliente para conectarse a un servicio previamente creado.

Si el usuario desea establecer una conexión servidora, únicamente deberá crear dicha conexión y permanecer a la espera de las conexiones por parte de los clientes. En este caso no será necesaria la realización de búsqueda de dispositivos, ya que serán los clientes los que deberán encontrar al servidor y no al contrario.

Si por el contrario el usuario desea lanzar un cliente, después de la inicialización del dispositivo local el usuario deberá realizar una búsqueda de dispositivos. Una vez obtenida una lista de dispositivos encontrados, se deberán buscar los servicios que ofrecen cada uno de estos dispositivos, para que finalmente el usuario pueda seleccionar el servicio y establecer la comunicación con el servidor y servicio seleccionados.

A continuación se describen brevemente las clases principales que componen este paquete. En la documentación de la *API original de Java* [2] aparece la lista completa de clases y métodos que lo compone, por lo que aquí solamente se definirán los aspectos principales del mismo sin entrar en detalles, dotando al lector de una visión global del funcionamiento y metodología de programación de la API.

Clase *LocalDevice*

Esta clase será el punto de partida de prácticamente cualquier operación que queramos realizar utilizando esta API. Esto se realizará obteniendo un único objeto de esta clase, lo cual se hará de la forma *LocalDevice ld = getLocalDevice().*

Una vez obtenido este objeto podremos aplicarle diferentes métodos para obtener información relativa al mismo como por ejemplo el nombre o *friendly-name* que tiene establecido, la dirección Bluetooth del dispositivo, el estado de conectividad, etc. Esto se obtiene mediante los métodos *ld.getFriendlyName(), ld.getBluetoothAddress y ld.getDiscoverable()* (aplicándolos al objeto obtenido anteriormente).

Si queremos que nuestro dispositivo pueda ser detectado por otros dispositivos, por ejemplo cuando queramos programar un servidor, deberemos establecer la conectividad del mismo, pudiendo elegir entre conectividad ilimitada, limitada o sin conectividad. Esto se hace con el método *ld.setDiscoverable(CONSTANTE),* donde *CONSTANTE* especifica el tipo de conectividad.

[2] Web de la API JSR-82 oficial de Java. http://docs.oracle.com/javame/config/cldc/opt-pkgs/api/bluetooth/jsr082/

En nuestro programa, al utilizar una API que funciona bajo Windows, el estado de la conectividad del dispositivo es establecido por el controlador del mismo, de modo que no nos será posible establecer el estado de conectividad desde nuestra aplicación. Para poder establecer el estado de conectividad en nuestro dispositivo, deberemos especificarlo en la opción adecuada del controlador del dispositivo entrando en *Panel de Control* en la parte de Bluetooth o directamente haciendo doble clic en el icono Bluetooth de la barra de tareas (este icono aparece al conectar un dispositivo Bluetooth).

Al intentar inicializar el dispositivo local se puede producir una excepción del tipo *BluetoothStateException*. En el caso de que esto se produjese significaría que no se pudo inicializar el dispositivo local.

Clase *DeviceClass*

Esta clase nos ofrece el método *getDeviceClass()*, mediante el cual podremos saber si el dispositivo sobre el que aplicamos dicho método es un ordenador, un teléfono, etc. En la especificación de la API oficial de Java aparece una lista con los posibles valores que devuelven los métodos de esta clase y el significado que tienen.

Clase *UUID*

La clase *UUID (Universally Unique Identifier)* define una serie de números enteros que identifican protocolos y servicios. Estos identificadores serán usados para buscar y crear servicios determinados. En la documentación de la clase se incluye una tabla con los protocolos y servicios más comunes y sus UUIDs asignados. Un servicio quedará identificado por su UUID asociado y por una serie de atributos que especifican ciertas características del mismo, variantes, etc.

Clase *DiscoveryAgent*

Mediante un objeto de esta clase, un cliente, podrá realizar búsquedas de dispositivos y servicios. Este objeto será único y se obtendrá a partir del objeto de la clase *LocalDevice*, obtenido anteriormente, de la siguiente manera:

DiscoveryAgent discoveryAgent = ld.getDiscoveryAgent()

Entre los métodos que nos ofrece esta clase, se encuentra el método retrieveDevices(CONSTANTE), el cual, sin realizar ninguna búsqueda de dispositivos, nos devuelve una lista de dispositivos que el usuario haya configurado como dispositivos conocidos o dispositivos encontrados en búsquedas anteriores.

Para realizar búsquedas de dispositivos deberemos aplicar el método *startInquiry(argumentos)* sobre el objeto de la clase obtenido anteriormente. A este

método se le pasan una serie de argumentos para especificar, entre otras cosas, la conectividad que han de tener los dispositivos Bluetooth que se han de encontrar en la búsqueda. Para cancelar la búsqueda se llamará al método *cancelInquiry().*

Interfaz *DiscoveryListener*

La clase desde la que se realice la búsqueda de dispositivos y servicios deberá implementar la interfaz DiscoveryListener. Esto se hace incluyendo la cláusula de implementación de la interfaz en la declaración de la clase así como los métodos que esta posee dentro de la misma, tal y como se indica a continuación:

```
public class Ejemplo implements DiscoveryListener{
        public static void main(String args[ ]){
        …….
       }//Fin del método main
  Aquí se incluirían los métodos de la interfaz DiscoveryListener
}//Fin de la clase
```

Los métodos que posee esta interfaz son los siguientes:

- **public void deviceDiscoverd(RemoteDevice rd, DeviceClass c):** Este método será automáticamente invocado durante la búsqueda de dispositivos cada vez que un nuevo dispositivo sea descubierto.
- **public void inquiryCompleted(int c):** Este método será automáticamente invocado durante la búsqueda de dispositivos cuando esta finalice.
- **public void servicesDiscoverd(int transID, ServiceRecord[ ] sr):** Este método será automáticamente invocado durante la búsqueda de servicios en cada dispositivo cuando sea descubierto el servicio que se está buscando en dicha búsqueda. Al encontrar el servicio podemos obtener automáticamente la dirección de conexión del mismo mediante la función getConnectionURL() para posteriormente poder conectarnos con el mismo.
- **public void servicesSearchCompeted(int transID, int respCode):** Este método será automáticamente invocado durante la búsqueda de servicios cuando esta finalice.

Comunicaciones

Utilizando el paquete javax.bluetooth se pueden establecer conexiones mediante dos mecanismos diferentes: SPP y L2CAP. Las conexiones de tipo SPP (*Serial Port*

Profile) se establecen mediante flujos de datos de tipo *InputStram* y *OutputStream*. En cambio, las conexiones establecidas mediante L2CAP (Logical Link Control and Adaption Protocol) se efectúan mediante arrays de bytes.

Comunicación por parte del cliente

Una vez que el cliente ha realizado la búsqueda de dispositivos y servicios, estará en disposición de establecer la conexión con el servidor mediante la URL obtenida en la búsqueda. Para realizar la conexión se usará el método C*onnector.open(URL)* tal y como se indica a continuación:

```
StreamConnection conexion = (StreamConnection) Connector.open(URL);
OutputStream salida = connexion.openOutputStream();
InputStream entrada = connexion.openInputStream();
        o también
StreamConnection conexion = (StreamConnection) Connector.open(URL);
DataOutputStream salida = connexion.openDataOutputStream();
DataInputStream entrada = connexion.openDataInputStream();
```

Dependiendo del tipo de datos que se deseen manejar.

Una vez establecida la conexión con el servidor podremos leer y escribir de ella con métodos como *in.readUTF()*, *out.writeUTF()*, etc.

En el caso de que se tratase de una conexión L2CAP, al realizar la llamada al método *Connector.open()*, éste nos devolvería un objeto de tipo *L2CAPConnection*, a través del cual podríamos leer y enviar bytes mediante los métodos *recive()* y *send()* respectivamente.

Comunicación por parte del servidor

Después de establecer el modo de conectividad del servidor como *visible*, procederemos a la creación de una conexión servidora de forma que los clientes que la detecten puedan conectarse.

Para establecer una conexión servidora deberemos construir primeramente una cadena con la dirección del servicio que vamos a crear. Esta cadena deberá comenzar por *"btssp://localhost:"* o *"btl2cap://localhost:"* dependiendo del tipo de comunicación que se desea implementar. A continuación deberemos concatenar el UUID del servicio a crear

y posterior a este el nombre y parámetros del servicio. A continuación se muestra un ejemplo para crear una dirección de conexión servidora de tipo SPP:

```
StringBuffer URL = new StringBuffer ("btssp://localhost:");
URL.append((new UUID(0x12345).toString());
URL.append(";name=Nombre de mi servicio;authorize=true");
```

Una vez construida la dirección para el servicio a ofertar, establecemos la conexión servidora a partir de ella de forma similar a como lo hacíamos con el cliente. Al llamar al método *Connector.open()* se nos devolverá un *notificador*, que en el caso de tratarse de una conexión SPP, será un objeto de la clase *StreamConnectionNotifier*, y en el caso de L2CAP, será un objeto de tipo *L2CAPConnecionNotifier*. Por ejemplo, para abrir una conexión servidora a partir de la URL obtenida anteriormente, tendríamos:

```
StreamConnectionNotifier notificador=null;
notificador= (StreamConnectionNotifier) Connector.open(URL.toString());
```

1.7.6. El paquete javax.obex

Como ya hemos mencionado, el protocolo OBEX es muy parecido al HTTP, y es comúnmente utilizado para operaciones de transferencia de ficheros. Está basado en un modelo cliente – servidor de intercambio de mensajes. Estos mensajes consisten en un conjunto de cabeceras de mensaje y un cuerpo opcional del mismo.

En una comunicación OBEX, el cliente envía comandos al servidor junto con algunas cabeceras de mensaje y en ocasiones, un cuerpo de mensaje (dependiendo del comando).

En OBEX, las cabeceras se encapsulan en un objeto de la clase *HeaderSet*. El cuerpo de los mensajes se leen o escriben mediante un *InputStream* o un *OutputStream*.

Cuando un servidor recibe los comandos OBEX por parte del cliente, responde a los mismos con un código de respuesta indicando el éxito o no de la petición. Al igual que los mensajes provenientes de los clientes, los mensajes de respuesta del servidor también incluyen cabeceras y en ocasiones un cuerpo de mensaje (dependiendo del comando).

Los distintos comandos de operación que el cliente puede enviar al servidor son:

- **CONNECT:** Para iniciar la comunicación con el servidor.
- **PUT:** Para enviar datos al servidor.
- **GET:** Para solicitar datos al servidor.

- **DELETE:** Para eliminar un recurso del servidor.
- **SETPATH:** Para crear directorios y navegar por ellos.
- **CREATE-EMPTY:** Para crear un fichero vacío en el servidor.
- **ABORT:** Para cerrar la conexión de forma prematura.
- **DISCONNECT:** Para cerrar la conexión.

En sentido contrario, el servidor responderá a los clientes OBEX, dependiendo del éxito o no de la operación, con los siguientes comandos:

- **SUCCESS:** Este tipo de paquete es devuelto por el servidor cuando la operación finaliza bajo normales circunstancias.
- **FAILURE:** Si se ha producido algún problema durante la operación.
- **CONTINUE:** Cuando un cliente le envía múltiples paquetes.

Así pues, la programación a través del paquete *javax.obex* supone trabajar a un nivel bastante superior al paquete *javax.bluetooth*, sin tener que preocuparnos de las dificultades que supone la programación a bajo nivel. De esta forma se podrían desarrollar aplicaciones más eficientes, sencillas y manejables que con el paquete anterior.

A continuación se describen brevemente las clases principales que componen este paquete. En la documentación de la *API original de Java*[3], aparece la lista completa de clases y métodos que la compone, por lo que aquí solamente se definirán los aspectos principales del mismo sin entrar en detalles, dotando al lector de una visión global del funcionamiento y metodología de programación de la API.

Interfaz *ClientSession*

Se trata de una subclase de la clase *javax.microedition.io.Connection*. Su definición representa una conexión de tipo OBEX desde el punto de vista del cliente. A través de esta clase se definirán las cabeceras para poder realizar operaciones como las vistas anteriormente (CONNECT, PUT, GET, etc.). Para obtener una instancia de la interfaz lo haremos de la siguiente manera:

```
ClientSession cs = (ClientSession) Connector.open(url)
```

[3] Web de la API JSR-82 oficial de Java. http://docs.oracle.com/javame/config/cldc/opt-pkgs/api/bluetooth/jsr082/

Interfaz *HeaderSet*

La interfaz *HeaderSet* define los métodos para obtener y establecer valores para cabeceras de comunicaciones a través del protocolo OBEX. Para crear una instancia de de la interfaz lo haremos a partir del objeto *ClientSession* obtenido anteriormente utilizando el método *createHeaderSet()*, de la siguiente manera:

```
HeaderSet hsOperation = cs.createHeaderSet();
hsOperation.setHeader(HeaderSet.NAME, fi.getName());
hsOperation.setHeader(HeaderSet.LENGTH, new Long(filebytes.length));
```

Interfaz *Operation*

Cuando deseamos realizar una operación OBEX de tipo GET o PUT, además de crear las cabeceras oportunas con los datos del envío, es necesario completar la operación enviando o recibiendo los datos correspondientes a la operación. La interfaz *Operation* define los métodos para completar la operación a realizar. A continuación vemos un ejemplo de cómo completaríamos una operación de tipo PUT utilizando la interfaz Operation:

```
Operation putOperation = cs.put(hsOperation);
OutputStream os = putOperation.openOutputStream();
try {   ByteArrayInputStream is = new ByteArrayInputStream(filebytes);
        byte[] buffer = new byte[0x400];
        int i = is.read(buffer);
                while (i != -1) {
                        os.write(buffer, 0, i);
                        i = is.read(buffer);
                }
        os.close();
        putOperation.close();
}catch (Exception e) { }
```

Interfaz SessionNotifier

Si en lugar de establecer una conexión OBEX como cliente, lo que deseamos es tomar el papel de servidor y ofrecer un servicio, deberemos implementar esta interfaz. Para ello utilizaremos el método *acceptAndOpen()* que lanzará el servicio y quedará a la espera de conexiones entrantes, las cuales vendrán representadas mediante instancias

de la subclase *RequestHandler*. A continuación vemos un ejemplo de cómo utilizar esta interfaz:

```
SessionNotifier serverConnection = (SessionNotifier) Connector.open( "btgoep://localhost:"+ serverUUID + ";name=ObexExample");
RequestHandler handler = new RequestHandler();
serverConnection.acceptAndOpen(handler);
```

1.8. Resultados finales

En este apartado trataremos de describir los resultados obtenidos una vez elegidos los componentes a utilizar para el desarrollo de la aplicación. Se describirá la aplicación desarrollada, sus diferentes funcionalidades así como la metodología de programación utilizada para su desarrollo, las adversidades superadas durante el mismo, etc.

Por otro lado, se han elaborado una serie de manuales para las partes de usuario y mantenimiento. Éstos están incluidos en la documentación del proyecto como anexos y forman parte del alcance del proyecto. En ellos se describe al detalle la manera de instalar, mantener y utilizar la aplicación de forma correcta. En la parte de mantenimiento se describirán las funciones principales que desempeñan cada clase (Java) que compone el programa, quedando estas plenamente identificadas para su posible ampliación, mejora, corrección, etc.

Se incluyen además varios anexos con diagramas de flujo que ayudarán a entender el funcionamiento de la aplicación, así como un esquema de la arquitectura de clases utilizadas por la misma.

Por último se analizarán las conclusiones extraídas durante el desarrollo del proyecto y las líneas de futuro aplicables al mismo.

1.8.1. Descripción de la aplicación

Tal y como ya comentamos en el apartado *1.1*., la aplicación desarrollada consiste en un software que se ejecuta sobre un PC con Sistema Operativo Windows, el cual dispone de un adaptador Bluetooth USB. La aplicación realiza el papel de un servidor de ofertas comerciales, pudiendo realizar envíos bajo demanda o de manera automática a los dispositivos móviles de los clientes que se encuentren dentro del radio de cobertura del servidor. El envío de una oferta consiste en la transferencia de un fichero desde el servidor al dispositivo móvil del cliente, para lo cual éste deberá tener su adaptador Bluetooth activado.

Después de comenzar las pruebas con BlueCove, fuimos obteniendo algunos resultados. Primeramente conseguimos inicializar el dispositivo local y extraer información del mismo, como el nombre y la dirección MAC. Una vez inicializado con éxito el dispositivo local, el siguiente paso era evidentemente el poder realizar búsquedas de dispositivos y servicios, lo cual también se consiguió sin mayores dificultades. Una vez comprobada la buena dirección que seguía el desarrollo de la aplicación en su parte más delicada, como era la interacción con el dispositivo Bluetooth, era la hora de plantear la estructura del programa, las funcionalidades pretendidas y diseñar su interfaz gráfica correspondiente.

FUNCIONALIDADES

A continuación, se enumeran las distintas funcionalidades de las que consta la aplicación, las cuales definen de manera conjunta el objetivo de la aplicación desarrollada:

- **Gestión de ofertas**: La aplicación podrá crear, modificar y eliminar ofertas. Cada oferta constará de un nombre, un periodo de validez y un fichero asociado. El periodo de validez quedará definido por una fecha de inicio de validez y una fecha de fin. Por otro lado el fichero asociado a cada oferta deberá ser del tipo *jpg, pdf* o *txt*.
- **Búsqueda manual de clientes**: Desde la aplicación se podrán hacer a petición del administrador búsquedas de dispositivos Bluetooth conectables. Si durante la búsqueda se obtienen resultados (dispositivos encontrados), estos serán registrados automáticamente en la base de datos de la aplicación. Los dispositivos a encontrar deberán ser aquellos que admitan el envío de ficheros.
- **Búsqueda automática de clientes**: Se podrán hacer búsquedas periódicas de dispositivos Bluetooth conectables. Estas búsquedas se repetirán constantemente cada cierto tiempo, siendo el administrador del programa quien determine el tiempo entre ellas además de decidir cuándo han de comenzar y terminar. De igual manera, los resultados obtenidos en cada búsqueda deberán registrarse en la base de datos o actualizar los existentes.
- **Gestión de clientes**: Sobre los clientes registrados en la base de datos, el administrador de la aplicación podrá realizar acciones de eliminación, para aquellos casos en que se realice un registro erróneo, o bien activar o desactivar el envío de ofertas a aquellos clientes que así lo manifiesten.
- **Envío manual de ofertas**: A petición del administrador, la aplicación podrá realizar un envío puntual de aquellas ofertas que se encuentren vigentes en ese instante. Las ofertas se enviarán de manera secuencial a cada dispositivo registrado, siempre y

cuando éste se encuentre en el radio de cobertura, tenga activado el envío de ofertas y no las haya recibido con anterioridad.

- **Envío automático de ofertas**: Al igual que se hace con las búsquedas, el administrador podrá iniciar un envío periódico de ofertas. Estos ciclos de envío se repetirán constantemente cada cierto tiempo, siendo el administrador del programa quien determine el tiempo entre ellos además de decidir cuándo han de comenzar y terminar.

INTERFAZ DE USUARIO

Puesto que el administrador de la aplicación deberá interactuar frecuentemente con la misma creando ofertas, modificándolas, iniciando búsquedas, etc., se hizo preciso el desarrollo de una interfaz amigable para facilitar esta tarea al administrador, cubriendo así las funcionalidades principales de la aplicación. De esta manera, el diseño de la interfaz de usuario realizado ha sido acorde a las funcionalidades que se pretenden, resultando un diseño bastante intuitivo, sencillo y ordenado. Así pues, la interfaz consta de tres secciones bien diferenciadas, una para la gestión de búsquedas y envíos, otra para la gestión de ofertas y otra para la gestión de clientes.

Cada una de las secciones dispone de distintos elementos (campos de texto, botones, etc.) que interactúan con el usuario (administrador de la aplicación). En el manual de usuario de la aplicación (apartado *2.2.)* se describen con detalle cada uno de los elementos de los que se dispone en cada sección así como su manera de utilizarlos.

Por otro lado, con el objeto de dotar a la aplicación de un cierto grado de **estabilidad**, se han creado una serie de controles y reglas cubriendo algunos aspectos o posibles circunstancias que se pueden dar durante la ejecución de la misma, para que el usuario (administrador) pueda realizar acciones que por un motivo u otro no puedan ser realizadas o puedan poner en peligro la correcta ejecución del programa. Estas son, entre otras, las siguientes:

- Los elementos que interactúan con el dispositivo Bluetooth permanecerán deshabilitados hasta que el dispositivo Bluetooth USB sea debidamente inicializado.
- Al crear una oferta, el formulario que recoge sus datos deberá estar debidamente cumplimentado. Los ficheros seleccionados se han restringido a los propuestos para su envío (jpg, txt y pdf). Se realiza un control de las fechas escogidas para el período de validez, de manera que la fecha fin deberá ser igual o mayor a la de inicio.
- El tiempo entre búsquedas automáticas se ha limitado a un mínimo de 2 minutos y un máximo de 60.
- El tiempo entre envíos automáticos se ha limitado a un mínimo de 2 minutos y un máximo de 60.

- Se ha establecido un *timeout* configurable para la espera de respuesta por parte de los clientes. Dicho timeout será como mínimo de 10 segundos y como máximo de 60.
- Si un cliente no es alcanzable o no responde en el timeout configurado, la aplicación lo ignorará hasta el próximo ciclo de envío, de manera que no quede colgada esperado o intente consecutivamente realizar envíos fallidos.
- Mediante el uso de *hilos (Threads)* se ha independizado la ejecución de la interfaz gráfica de la ejecución de las funciones de envío y búsqueda. De esta manera, la interfaz no queda congelada o colgada esperando cuando se inicie algunas de las funciones mencionadas.

1.8.2. Desarrollo de la aplicación

En el apartado anterior se han descrito las funciones principales de la aplicación desarrollada y sus características constructivas más importantes. Como ya se comentó, para el desarrollo de la aplicación se ha recurrido a varias herramientas como la programación por hilos, interfaz gráfica, etc. Este apartado trata de introducir mediante algunos ejemplos cómo se llevan a cabo algunas de las funciones que desempeña la aplicación desarrollada así como citar las herramientas utilizadas para el desarrollo de la misma, la cual ha sido bautizada con el nombre de ***BlueSend***.

ENTORNO DE EDICIÓN Y COMPILACIÓN

A la hora de emprender el desarrollo de una aplicación de ciertas dimensiones, resulta muy interesante el uso de un IDE (*Integrated development environment)*, es decir, un entorno integrado de desarrollo. Un IDE puede hacernos el trabajo mucho más sencillo, sobre todo si nuestra aplicación va manejando un buen número de clases Java. Además estos entornos nos permiten mucha más versatilidad para depurar nuestros programas puesto que tienen debbugers mucho más avanzados, cosa que facilita muchísimo la detección y corrección de errores.

NetBeans es el IDE utilizado para el desarrollo de nuestra aplicación. Se trata de un IDE de código abierto de fácil instalación y multitud de funcionalidades. Hay más herramientas similares de código abierto disponibles, sin embargo, NetBeans posee una de las mejores relaciones *calidad - facilidad*, siendo especialmente adecuado para el desarrollo de aplicaciones Java. Su instalación es muy sencilla. Basta con descargarse el instalador de su *página oficial*[4] y seguir el asistente de instalación. En la *figura 1.7* se muestra una captura de pantalla del IDE NetBeans.

[4] Página oficial de NetBeans. https://netbeans.org/

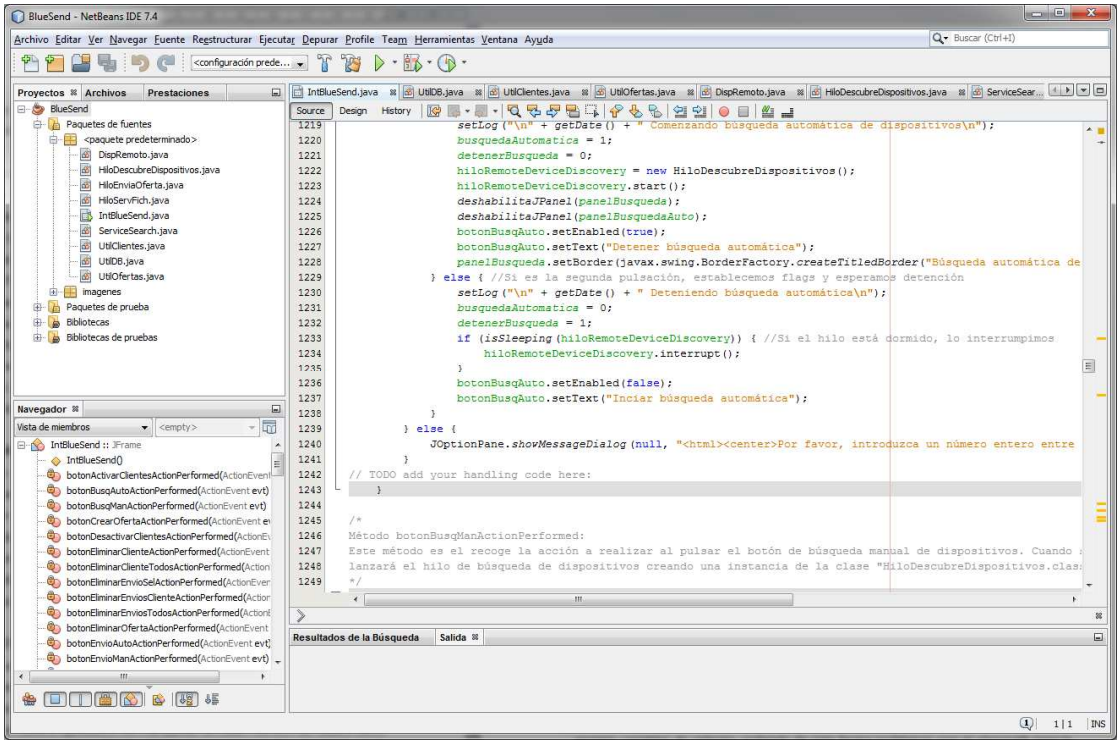


Figura 1.7: IDE NetBeans

Una de las características clave de NetBeans es su modularidad, lo cual permite a los desarrolladores extender sus aplicaciones a otros desarrollando módulos propios para el IDE. Por otro lado, NetBeans posee un entorno para el desarrollo de interfaces gráficas especialmente útil, simplificando y facilitando enormemente el desarrollo y programación de la interfaz gráfica en sí.

Al crear el proyecto de nuestra aplicación en el IDE NetBeans, es necesario añadir a la biblioteca de librerías la librería correspondiente a la API utilizada, en nuestro caso ***bluecove-2.1.1-SNAPSHOT.jar***. Otra de las ventajas de NetBeans es que crea sus propias variables de entorno, evitando de esta forma problemas con el *classpath* para la compilación y ejecución del programa.

INTERFAZ GRÁFICA

Como hemos comentado anteriormente, NetBeans es especialmente útil para el diseño y desarrollo de interfaces gráficas en Java. Esto es gracias a que posee dos modos de edición, uno es el modo *fuente (source)*, en el que realmente vemos todo el código que compone la clase que estemos editando en ese momento. El otro modo de edición, y el que nos interesa para el desarrollo de la interfaz, es el modo *diseño (design).* En este modo podemos crear una interfaz gráfica de manera “relativamente” sencilla,

simplemente hay que ir seleccionando los elementos de una barra de herramientas e insertándolos en un panel contenedor, que se encuentra dentro de un marco de ventana.

NOTA: En el CD del proyecto se incluye la versión 7.4. de NetBeans en castellano, que es la utilizada para el desarrollo de la aplicación.

Se pueden insertar multitud de elementos, como etiquetas, botones, listas, puntos de selección, menús desplegables, etc. Lo único (que no poco) que tendrá que realizar el programador será la programación de los eventos que controlan la interfaz. En nuestro caso, los eventos manejados han sido muy numerosos, como deshabilitar elementos ante ciertas circunstancias, ejecutar acciones dependiendo del botón pulsado, cambiar textos de etiquetas dependiendo de algunas variables, etc.

En la *figura 1.8* se aprecia una captura del modo diseño de NetBeans en el que se observan algunos de los elementos de que dispone.

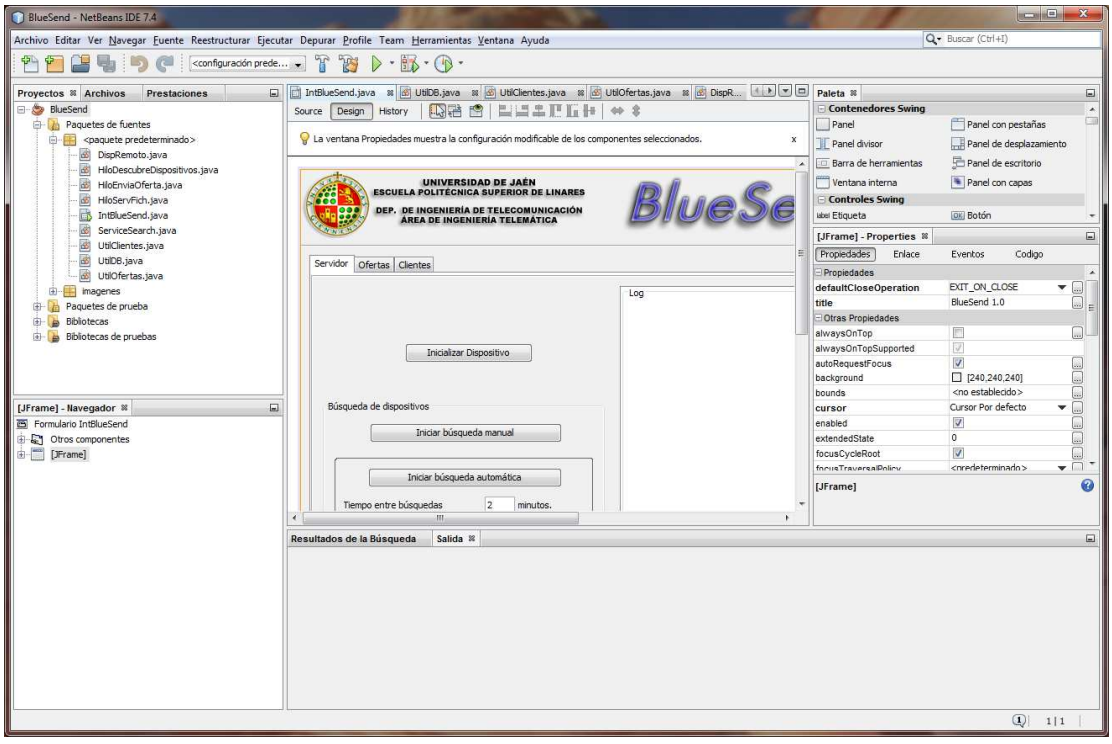


Figura 1.8: NetBeans en modo diseño

Su modo de funcionamiento es sencillo. Cuando el desarrollador inserta un elemento en la ventana de la interfaz gráfica de la aplicación, automáticamente NetBeans generará las líneas de código oportunas para implementar los cambios realizados. Sin embargo, como ya se ha comentado, el programador si deberá escribir las líneas de código referentes al manejo de eventos de la interfaz.

A modo de ejemplo, a continuación se presenta un fragmento de código extraído del código fuente de la aplicación en el que se aprecia el manejo del evento relacionado con la inicialización del dispositivo local en la clase *IntBlueSend.class*.

```
private void botonInicioDispActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        miDispositivo = LocalDevice.getLocalDevice(); // Obtenemos el dispositivo local
    } catch (Exception e) {
        miDispositivo = null;
    }
    //Si el código de inicialización del dispositivo es incorrecto, mostramos mensaje de error
    //y cerramos la aplicación
    if (miDispositivo == null) {
        JOptionPane.showMessageDialog(null, "<html><center>No se pudo iniciar el dispositivo"
                + " bluetooth.<P><P>Asegúrese de que su dispositivo está conectado correctamente e"
                + " inténtelo de nuevo.</center></html>", "Error", JOptionPane.ERROR_MESSAGE);
        setLog("No se pudo iniciar el dispositivo local.\n");
    } else { //Si hay éxito obtenemos los datos del dispositivo y los mostramos en la interfaz.
             //Habilitamos los paneles oportunos.
        local_name = miDispositivo.getFriendlyName();
        local_address = miDispositivo.getBluetoothAddress();
        etiquetaEstado.setIcon(new ImageIcon(getClass().getResource("/imagenes/On.png")));
        etiquetaEstado.setText("<html><b>Estado: Iniciado</b><br>Nombre del dispositivo: " +
                local_name + "<br>Dirección bluetooth del dispositivo: "
                + local_address + "</html>");
        setLog(getDate() + " Dispositivo local iniciado.\n  Nombre del dispositivo: " + local_name +
                "\n  Dirección bluetooth del dispositivo: " + local_address + "\n\n");
        habilitaJPanel(panelBusqueda);
        habilitaJPanel(panelServidor);
        habilitaJPanel(panelBusquedaAuto);
        habilitaJPanel(panelServidorAuto);
        botonInicioDisp.setEnabled(false);
    }
```

En el fragmento de código anterior se puede observar que el manejo de eventos resulta intuitivo una vez se conocen los nombres de las funciones que lo manejan. Por ejemplo, para mostrar u ocultar un elemento, se utiliza la función *elemento.setVisible(boolean arg)*, de si *arg* es *true* querrá decir que se muestre y si es *false* que se oculte. Para establecer un texto determinado en una etiqueta o botón se utiliza la función *elemento.setText("Texto del elemento")*. Las funciones para establecer estados o parámetros de los elementos suelen comenzar por la palabra clave *set*, mientras que las que nos devuelven datos de los mismos suelen comenzar por la palabra clave *get.* En la documentación oficial del paquete *javax.swing* de Java se puede encontrar la lista completa de funciones y demás aspectos relacionados con los elementos de la interfaz gráfica.

MANEJO DE HILOS O THREADS

Otro de los elementos claves de la aplicación desarrollada es la *programación concurrente o multihilo*, gracias a la cual el servidor es capaz de realizar al mismo tiempo las tareas de búsqueda de dispositivos y envío de ofertas sin interferir entre ambas ni con

la interfaz de usuario. Así pues, cada vez que se inicie una búsqueda o comience un ciclo de envío, la aplicación lanzará un hilo independiente para cada tarea siendo su ejecución independiente y paralela al resto del programa.

La interfaz de Java y las clases que incluyen funcionalidades sobre hilos son las siguientes:

- **Thread**: Es la clase responsable de producir hilos funcionales para otras clases. Para añadir la funcionalidad de hilo a una clase simplemente se deriva la clase de Thread y se incluye el método run. Es en este método run donde el procesamiento de un hilo tendrá lugar, es decir, el cuerpo del hilo.
- **Runnable**: La interfaz Runnable proporciona la capacidad de añadir la funcionalidad de un hilo a una clase simplemente implementando la interfaz, en lugar de derivándola de la clase Thread.
- **Object Thread**: Esta clase proporciona algunos métodos cruciales dentro de la arquitectura multihilo de Java. Estos métodos son wait(), notify() y notifyAll(). El método wait() hace que el hilo de ejecución espere en estado dormido hasta que se le notifique que continúe. Del mismo modo, el método notify() informa a un hilo en espera de que continúe con su ejecución. El método notifyAll() es similar a notify() excepto que se aplica a todos los hilos en espera.

A modo de ejemplo, a continuación se presenta un fragmento de código extraído del código fuente de la aplicación en el que se aprecia cómo se declara y se lanza un hilo, concretamente el hilo para descubrir dispositivos, implementado en la clase *HiloDescubreDispositivos.class*

```
public HiloDescubreDispositivos hiloRemoteDeviceDiscovery = null;
hiloRemoteDeviceDiscovery = new HiloDescubreDispositivos();
hiloRemoteDeviceDiscovery.start();
```

En este caso, la clase *HiloDescubreDispositivos.class* deriva de la clase *Threads* y como el método *run()* contiene las sentencias que componen el cuerpo del hilo.

La creación y lanzamiento del hilo que contiene la clase se realiza desde la clase asociada a la interfaz gráfica llamada *IntBlueSend.class*, y estaría dentro del evento de pulsación del botón asociado a las búsquedas de dispositivos.

1.8.3. Conclusiones

Ante los resultados obtenidos tras el desarrollo de la aplicación, hemos podido comprobar que la programación de dispositivos Bluetooth ofrece muchas posibilidades, pudiendo desarrollarse programas de muy diversos usos y con una gran estabilidad en su funcionamiento.

Hemos comprobado como Java nos ofrece un entorno de programación de dispositivos Bluetooth bastante cómodo y manejable gracias a la API JSR-82, quedando patente de que se trata de una API con enormes posibilidades, pudiendo el usuario emprender el desarrollo de una aplicación totalmente personalizada para controlar dispositivos Bluetooth de forma sencilla y eficiente.

En cuanto a la tecnología Bluetooth, ha quedado constancia de su versatilidad de uso y ventajas que presenta. Las enormes posibilidades de aplicación y su gran estandarización son indicios de que será una tecnología con gran futuro en cualquiera de sus ámbitos de uso.

Actualmente, la gran aceptación que presenta la tecnología Bluetooth en la sociedad, presente prácticamente en cualquier modelo de Smartphone o Tablet, la convierte en una de las tecnologías de comunicación inalámbrica de mayor utilización a nivel global, siendo una apuesta firme de los principales fabricantes y cada vez más presente en la electrónica de consumo en general.

1.8.4. Líneas de futuro

Una línea de futuro interesante podría ser el desarrollo de una aplicación para los dispositivos clientes para su ejecución sobre uno de los sistemas operativos móviles más actuales, como Android.

Con esta aplicación, podría conseguirse que fuesen los clientes quienes, a demanda, solicitasen el envío de una oferta en concreto, o bien manifestasen su deseo de no recibir más.

La corrección de posibles *bugs* y una mejora de la usabilidad de la interfaz podrían ser otra de las posibles mejoras a emprender en un futuro para esta aplicación.

1.9. Planificación

Para la ejecución y redacción del presente proyecto, se ha llevado a cabo una planificación en 5 semanas, cada una de ellas conlleva una fase diferente. En la siguiente figura se aprecia un cronograma de la planificación seguida y sus distintas fases:

	Semana 1	**Semana 2**	**Semana 3**	**Semana 4**	**Semana 5**
Documentación y estudio de la solución a implementar.	x				
Inicialización del dispositivo local y realización de búsquedas.		x			
Parte de gestión de ofertas y clientes.			x		
Envío de ficheros y automatización de búsquedas y envíos.				x	
Redacción de documentos.					x

Figura 1.9: Tabla de planificación

1.10. Orden de prioridad entre los documentos

Ante posibles discrepancias, se ha establecido el siguiente orden de prioridad para los documentos básicos que componen el presente proyecto:

1. Pliego de condiciones
2. Anexos
3. Memoria
4. Presupuesto

2. Anexos

2.1. Manual de instalación

REQUISITOS MÍNIMOS

Hardware:

- Procesador Intel Pentium II o compatible a velocidad mínima de 266 MHz.
- 128 MB de memoria RAM.
- 200 MB de espacio libre en disco duro.
- Adaptador Bluetooth USB genérico V1.2 o superior.

Software:

- Sistema Operativo de 32 bits Windows 7 o XP Professional SP2.
- Java Runtime Environment 7 o superior.

MANUAL

Antes de instalar la aplicación comprobaremos que el equipo tiene instalada la máquina virtual de Java. Para ello podemos comprobarlo desde el *Panel de Control* del Sistema Operativo, o bien abriendo un terminal en el PC y escribiendo la palabra *java*. Si la consola no reconoce el comando, querrá decir que no tenemos instalado el entorno de tiempo de ejecución de Java (*Java Runtime Environment* o JRE), procediendo entonces según el siguiente apartado:

Instalación del entorno en tiempo de ejecución de Java (JRE)

Para la instalación del entorno en tiempo de ejecución de Java, simplemente hay que hacer doble clic en el fichero *"jre-7u67-windows-i586.exe"* incluido en el directorio *"\Software\"* del CD del proyecto. Una vez abierto el instalador basta con seguir los pasos del asistente haciendo clic en el botón *aceptar*, tal y como se muestra en la siguiente ilustración:

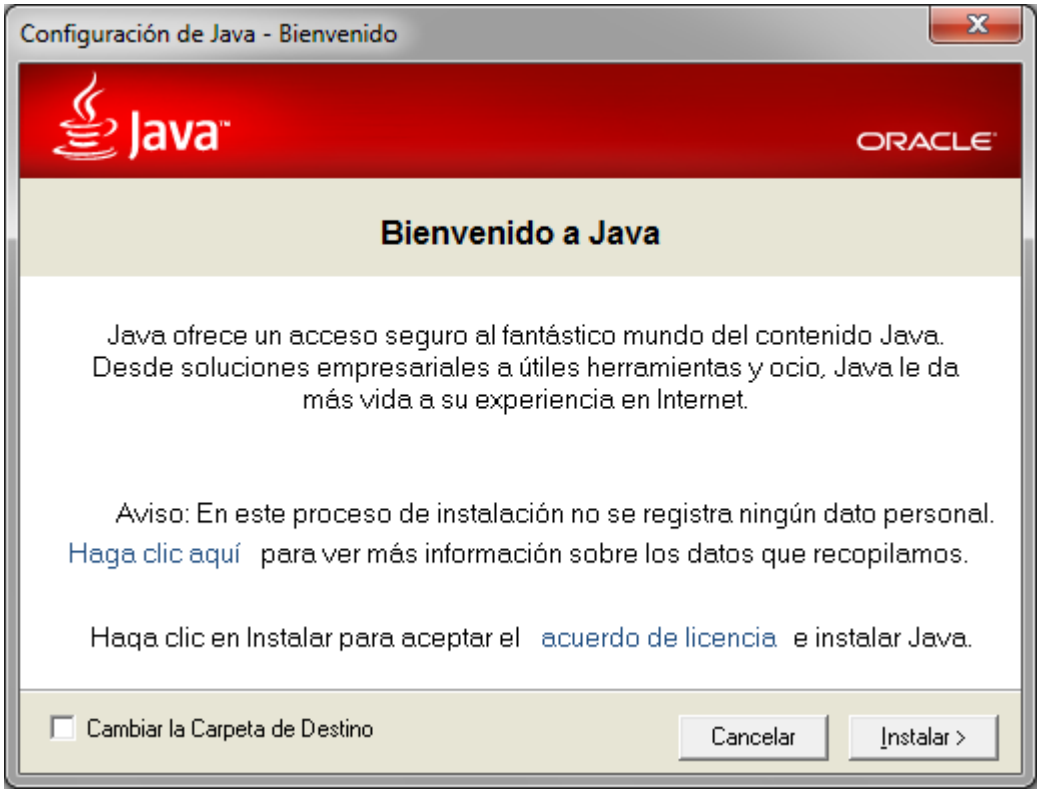


Figura 2.1: Instalación de Java Runtime Environment

Una vez instalado el entorno de ejecución en tiempo real de Java, pasamos a instalar la aplicación desarrollada.

Instalación de BlueSend 1.0

Puesto que la aplicación ha sido compilada en un **único fichero jar autoejecutable**, no será necesario realizar una instalación como tal, siendo la aplicación totalmente portable y ubicable en cualquier directorio del PC.

Así pues, bastará con copiar el directorio *"\BlueSend 1.0\"* incluido en del CD del proyecto a cualquier directorio del disco duro del PC donde vayamos a ejecutar la aplicación. En dicho directorio se ubica el fichero jar mencionado, denominado "*BlueSend.jar",* junto con un subdirectorio llamado *"lib".* En el subdirectorio *lib* se encuentran las librerías auxiliares utilizadas en la aplicación. Estas son:

- **/lib/bluecove-2.1.1-SNAPSHOT.jar**: Correspondiente a la implementación de la API JSR-82 BlueCove utilizada.
- **/lib/h2-1.4.180.jar**: Correspondiente a la base de datos H2 utilizada por la aplicación.
- **/lib/jcalendar-1.3.3.jar**: Se trata de una librería auxiliar para el lanzamiento de un calendario de selección de fechas en la interfaz.

El subdirectorio *lib* deberá estar situado a la misma altura que el fichero ejecutable *BlueSend.jar* para que así las librerías puedan ser referenciadas desde la aplicación.

Una vez copiado el directorio de la aplicación, bastará con hacer doble click sobre el fichero *BlueSend.jar* y la aplicación se ejecutará mostrando la interfaz de usuario. Por otro lado, la primera vez que se ejecute la aplicación, se creará otro subdirectorio llamado *ficherosOfertas* el cual contendrá los ficheros de las ofertas que se vayan creando durante la utilización del programa.

Instalación del dispositivo Bluetooth

Puesto que el programa ha sido desarrollado utilizando la pila de protocolos de Microsoft para controlar el dispositivo Bluetooth (*Microsoft Bluetooth Stack*), es necesario asegurarse de que el dispositivo sea correctamente instalado y utilice el controlador adecuado.

Así pues, a la hora de instalar los ***drivers*** del dispositivo, es **importante** NO instalar los *drivers* del fabricante y dejar que Windows 7 o Windows XP Professional SP2 instale sus propios controladores.

Para asegurarnos de que se están utilizando los controladores adecuados, podemos abrir el *Administrador de dispositivos de Windows*, y observar el controlador instalado. En la *figura 2.2* se aprecia una captura del aspecto que debe mostrar el dispositivo Bluetooth instalado en el *Administrador de dispositivos de Windows*.

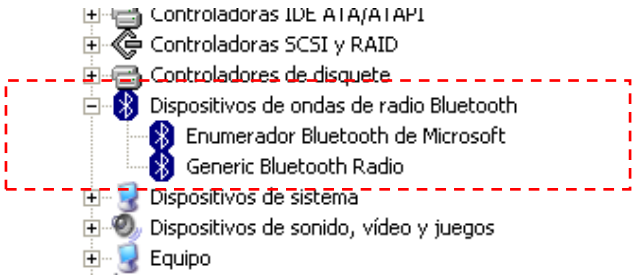


Figura 2.2: Controladores Bluetooth (Microsoft Bluetooth Stack)

Si por error se instala un controlador diferente, no se utilizará la pila de protocolos de *Microsoft Bluetooth Stack*, por lo que no se garantiza el buen funcionamiento de la aplicación. Para corregir esto bastaría con desinstalar los controladores existentes haciendo clic en el botón *"Quitar"* y volver a instalar el dispositivo dejando que Windows se encargue de reconocer los controladores e instalarlos.

2.2. Manual de usuario

En este manual se recogen las nociones básicas de uso de la aplicación. Se describirá cómo se comporta dependiendo de la interacción del usuario y de los eventos ocurridos durante su funcionamiento.

La interfaz de usuario de BlueSend consta de tres secciones principales, a las cuales se accede desde un menú de pestañas situado en la parte superior de la aplicación. Cada sección dispone de ciertos elementos para realizar sus propias funciones. Por otro lado, existen elementos comunes, como son la cabecera de títulos y la barra de estado del dispositivo Bluetooth. En la *figura 2.3* se aprecia una captura de la ventana principal de la aplicación y de las diferentes partes que componen la interfaz de usuario.

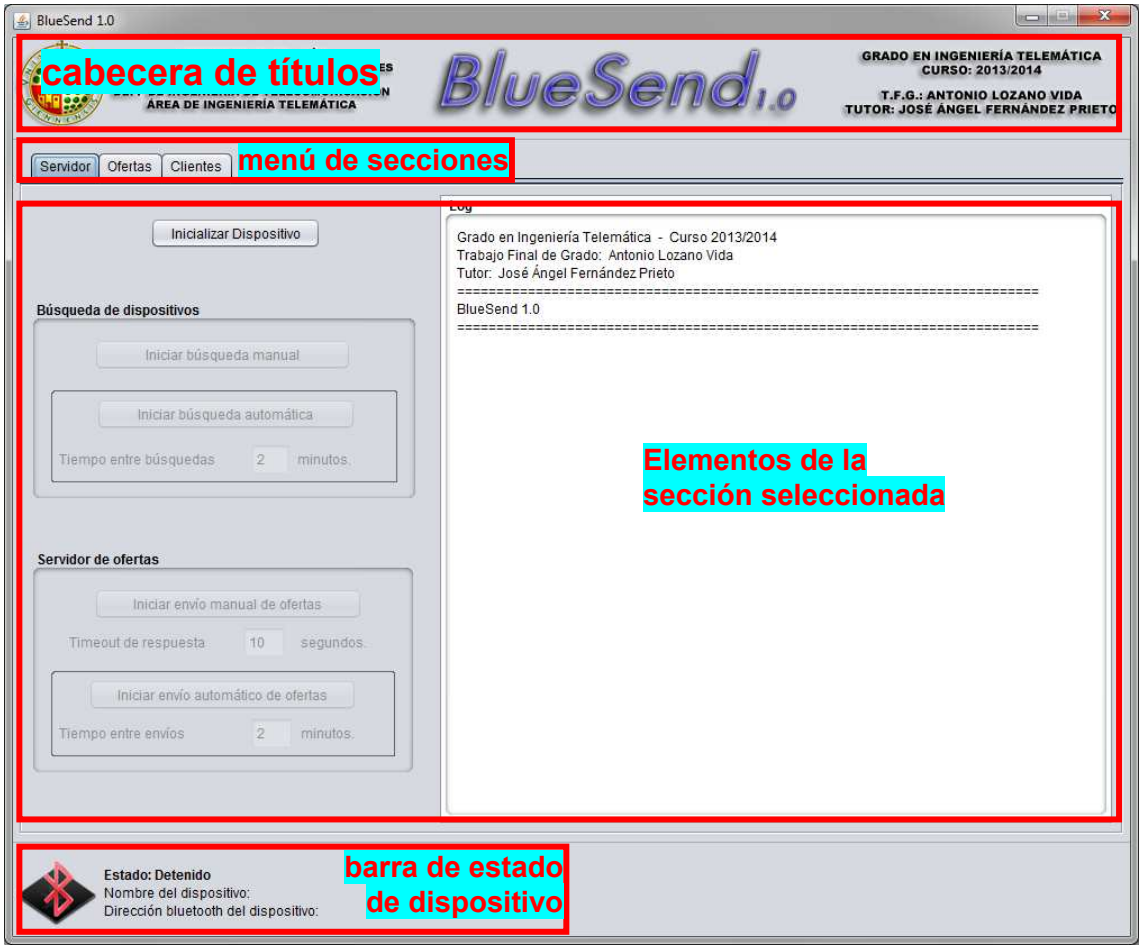


Figura 2.3: Interfaz de usuario BlueSend 1.0

A continuación detallaremos cada una de las secciones que componen la aplicación, sus elementos y funciones.

OFERTAS

Aunque se trata de la segunda de las secciones que aparece en el menú, la expondremos con anterioridad a las demás, pues es preciso crear ofertas previamente para poder enviarlas. En esta sección, el usuario dispone de varios elementos que nos proporcionan los mecanismos para **crear, modificar o eliminar las ofertas** que se enviarán a los dispositivos.

Creación de ofertas

Para la creación de ofertas se dispone de un **formulario de alta**, el cual incorpora una serie de campos que recogen los datos necesarios para añadir una nueva oferta al sistema. Una oferta constará de **nombre** o título**, periodo de inicio de validez, periodo de fin de validez** y un **fichero asociado**. El fichero asociado a cada oferta será el que se envíe a los clientes. Una vez introducidos los datos, se podrá dar de alta la oferta haciendo click en el **botón *crear oferta***.

A la hora de introducir los datos, es preciso que estén todos los campos debidamente cumplimentados para que la aplicación nos permita dar de alta la oferta. Por defecto, las fechas de inicio y fin de validez tomarán los valores de la **fecha actual** como fecha de inicio **y una semana después** como fecha de fin, aunque estos valores podrán cambiarse a petición del usuario, respetando siempre que la fecha de fin sea igual o posterior a la de inicio. Por otro lado, los ficheros seleccionables para asociar a la oferta serán ficheros de tipo ***jpg, txt o pdf***. No se han establecido restricciones respecto al tamaño de los mismos, pero se recomienda que éstos sean lo más ligeros posibles (200 KB o menos).

Modificación y eliminación de ofertas

Una vez creada la oferta, ésta se incorporará a una **tabla**, la cual mostrará una lista de las distintas ofertas creadas. Seleccionando cualquiera de las ofertas (filas) de la tabla, se nos habilitarán las opciones de **modificación** o **eliminación** para la oferta seleccionada. El **mismo formulario** se utilizará **para modificar o eliminar** las ofertas ya creadas.

En la *figura 2.4* se aprecia la sección de ofertas con los campos del formulario de alta debidamente cumplimentados, así como el mensaje que indica la correcta creación de la misma tras pulsar el botón de creación. En la *figura 2.5* se aprecia el listado tras la creación de varias ofertas y estando seleccionada una de ellas, permitiendo entonces la modificación o eliminación de la misma.

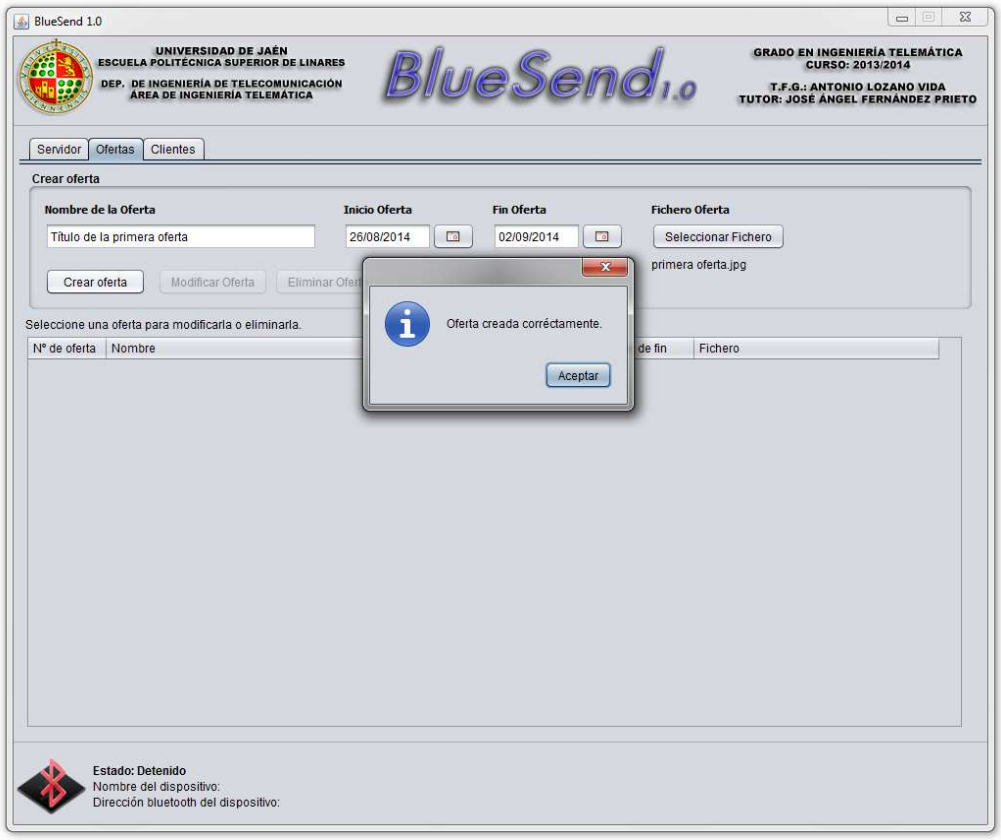


Figura 2.4: Creación de nueva oferta

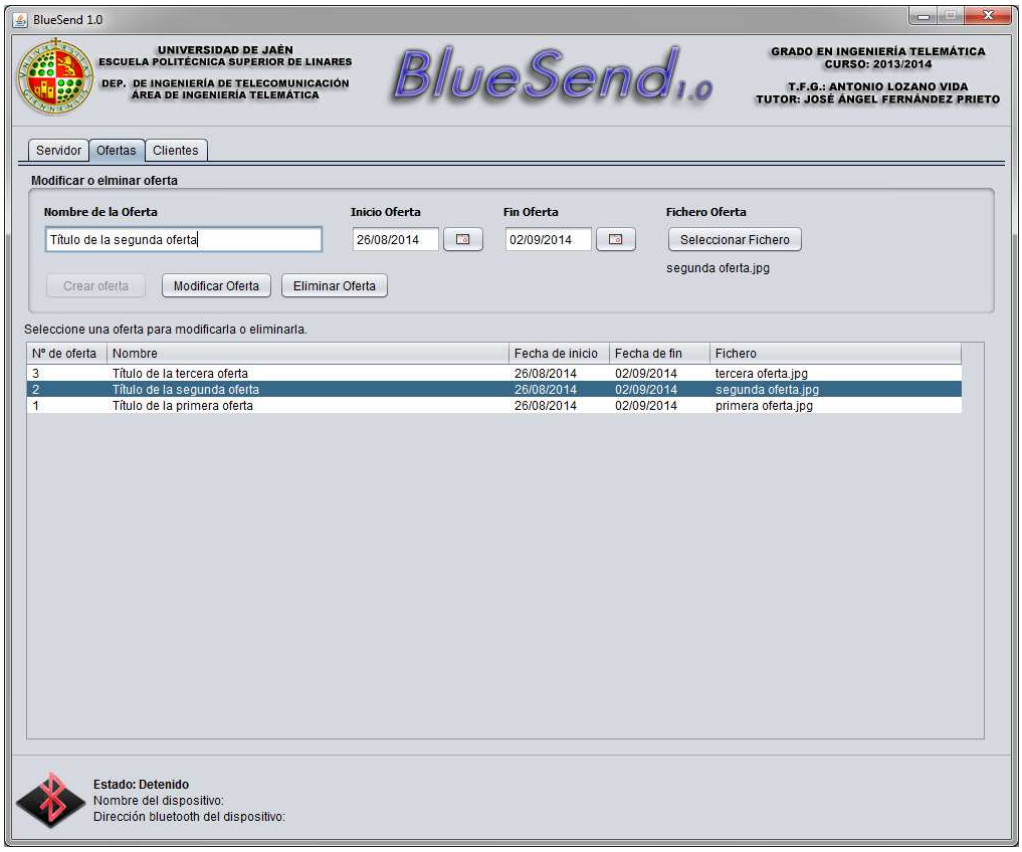


Figura 2.5: Varias ofertas creadas, una seleccionada.

CLIENTES

Se trata de la tercera de las secciones que aparecen en el menú. En ella el usuario podrá visualizar y eliminar los clientes encontrados en las búsquedas, así como los registros de envíos realizados a los mismos.

La sección está dividida en dos partes bien diferenciadas, izquierda y derecha. En la izquierda se nos muestra una tabla con un **listado de los clientes**, los cuales se irán añadiendo de manera automática a partir a partir de las búsquedas efectuadas en la sección *Servidor*. Seleccionando cualquiera de ellos se nos habilitarán las opciones de **eliminación** y de **activación/desactivación de los envíos** para dicho cliente (para el caso en que así lo manifiesten). Adicionalmente se dispone de acciones propias para eliminar todos los clientes y activar/desactivar el envío de ofertas para la totalidad de los mismos. La parte derecha corresponde a la **gestión de los envíos** realizados a los clientes a través del servidor de ofertas. Dependiendo del cliente que seleccionemos en la parte izquierda, se nos mostrará una relación de las ofertas que han sido enviadas con éxito a dicho cliente, las cuales se irán registrando de manera automática una vez enviadas. En este apartado de la sección se nos presentan opciones para eliminar un envío en concreto del cliente seleccionado, todos los envíos realizados al cliente seleccionado, o bien todos los envíos de todos los clientes. Esto puede ser útil si un cliente desea recibir nuevamente una oferta previamente enviada.

Gracias al registro de los envíos realizados a cada cliente, la aplicación únicamente tratará de enviar a un cliente las ofertas que aún estén pendientes de recibir por su parte, omitiendo aquellas que ya se hayan enviado con anterioridad o bien no estén dentro del periodo de validez fijado para su envío.

En la *figura 2.6* se aprecia la sección de clientes mostrando sendos listados de clientes y envíos realizados, así como las distintas opciones de eliminación y activación/desactivación.

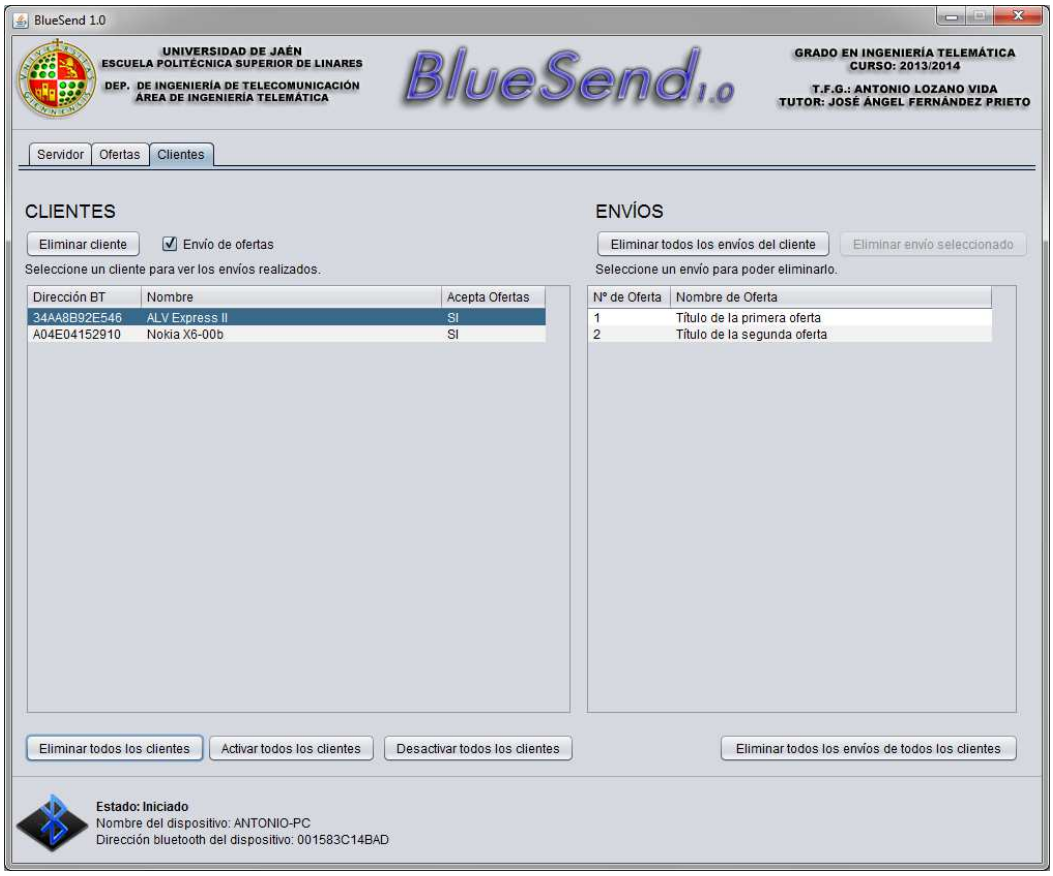


Figura 2.6: Sección clientes. Listado de clientes y envíos.

SERVIDOR

Es la sección principal de la aplicación, pues mediante ella el usuario podrá realizar **búsquedas de dispositivos** y realizar el **envío de ofertas** a los dispositivos encontrados. También es la sección que aparece por defecto activa al abrir la aplicación. A continuación detallaremos las funciones que se pueden realizar desde esta sección de la interfaz:

Inicialización del dispositivo

Se trata de un **paso previo e indispensable** para poder interactuar posteriormente con el dispositivo Bluetooth, bien sea para hacer búsquedas o para realizar envíos. La inicialización consiste en la detección del dispositivo Bluetooth local instalado en el equipo, así como sus características principales como el nombre y dirección MAC. Para este fin se dispone de un botón titulado *Inicializar Dispositivo*, ubicado en la parte superior izquierda de la sección. El resto de elementos de la sección permanecerán deshabilitados hasta que el dispositivo sea inicializado con éxito. En la parte derecha se dispone de una ventana de Log, en la cual se escribirá información de los distintos eventos que vayan

teniendo lugar en la aplicación, tales como búsquedas, envíos, altas de ofertas, modificaciones, etc.

Además de la ventana de Log, se dispone de una barra de estado de dispositivo ubicada en la parte inferior de la interfaz, estando ésta presente en toda la aplicación. Si el dispositivo es inicializado con éxito, se habilitarán el resto de elementos que componen la sección. Por otro lado se registrará un mensaje en la ventana de Log, a la vez que cambiará el icono de la barra de estado, indicando las propiedades del mismo. En caso contrario, la aplicación dará un error y registrará un mensaje indicando lo acontecido en la ventana de Log.

En las figuras 2.7 y 2.8, se aprecia respectivamente el estado que presenta la interfaz antes y después de inicializar el dispositivo local.

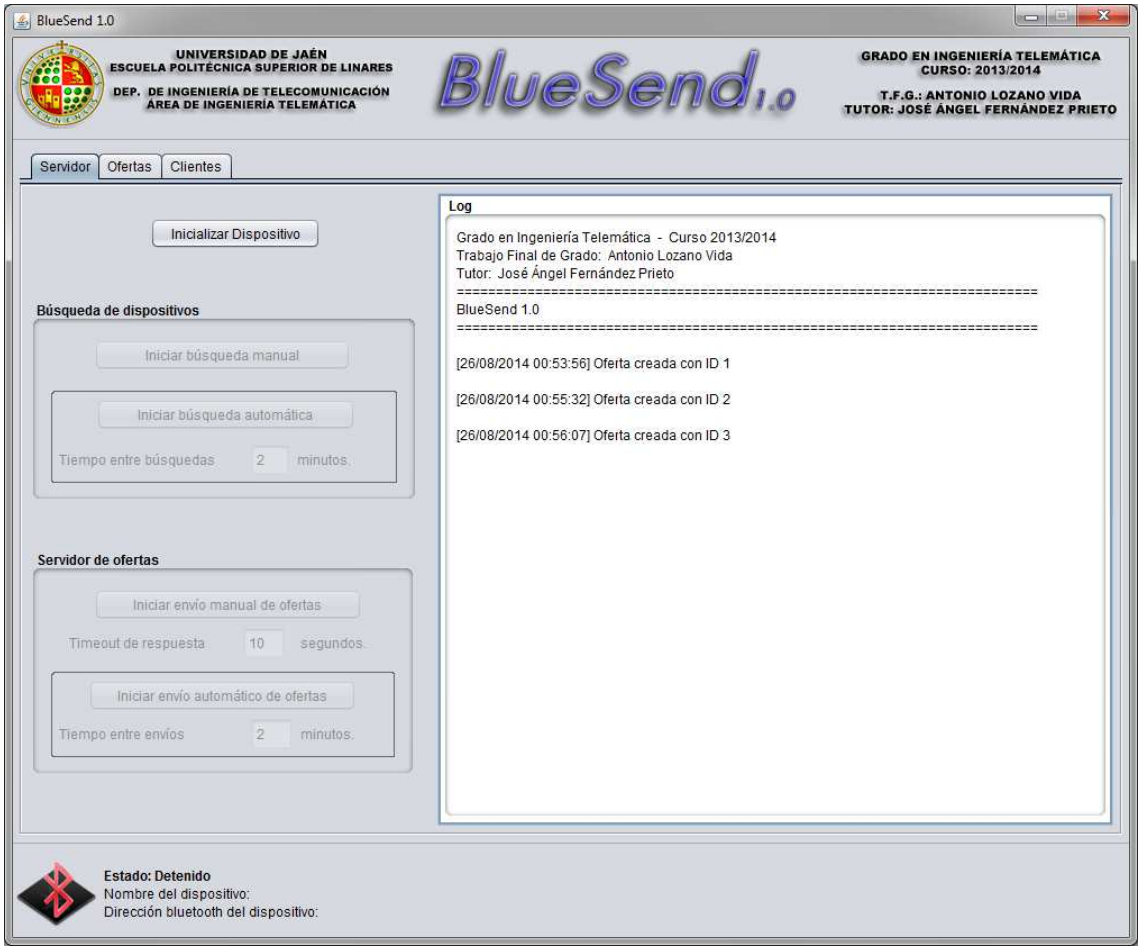


Figura 2.7: Sección Servidor antes de inicializar el dispositivo local.

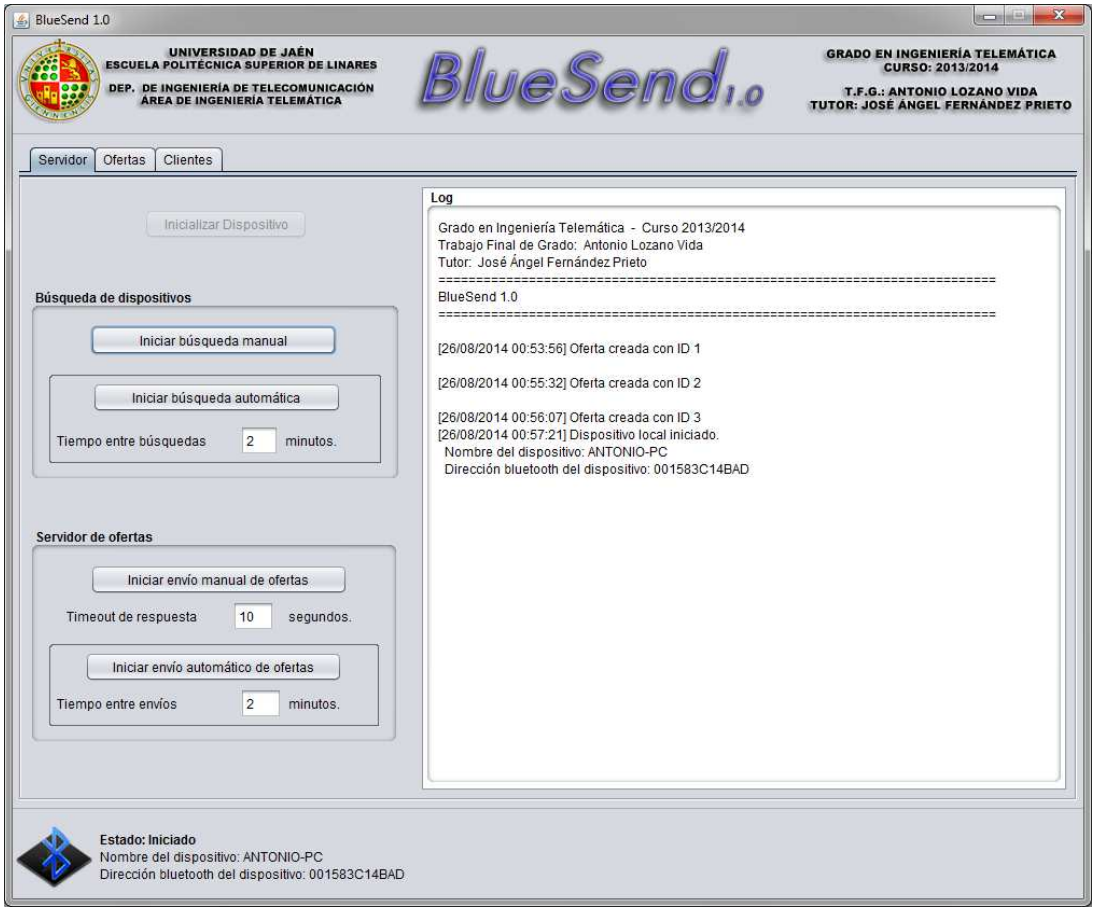


Figura 2.8: Sección Servidor después de inicializar el dispositivo local.

Búsqueda de dispositivos

Justo debajo del botón de inicialización nos encontramos con el ***panel de búsquedas***. Mediante los elementos de dicho panel se lanzarán las búsquedas de los dispositivos Bluetooth conectables para el envío de las ofertas. Se dispone en primer lugar de un botón titulado ***Iniciar búsqueda manual***, el cual una vez pulsado, lanzará una única búsqueda de dispositivos. Al iniciar la búsqueda se deshabilitarán por completo los elementos que componen el panel hasta que ésta finalice. Una vez finalizada se mostrarán los dispositivos encontrados, si los hubiere, a través de un mensaje en la ventana de Log, así como a través de la tabla de clientes en la *sección clientes.*

Por otro lado, en el mismo panel de búsquedas se dispone de otro botón titulado ***Iniciar búsqueda automática***, mediante el cual el usuario podrá iniciar un ciclo continuo de búsquedas, las cuales se irán repitiendo de manera automática cada cierto tiempo, hasta que se dé la orden de detención pulsando de nuevo el mismo botón. El tiempo entre búsquedas es configurable mediante el campo de texto ubicado en el mismo panel titulado *Tiempo entre búsquedas*, debiendo estar el valor comprendido entre 2 y 60 minutos. En las figuras 2.9 y 2.10, se aprecia respectivamente el estado que presenta la interfaz justo después de pulsar el botón de búsqueda manual y tras finalizar la misma.

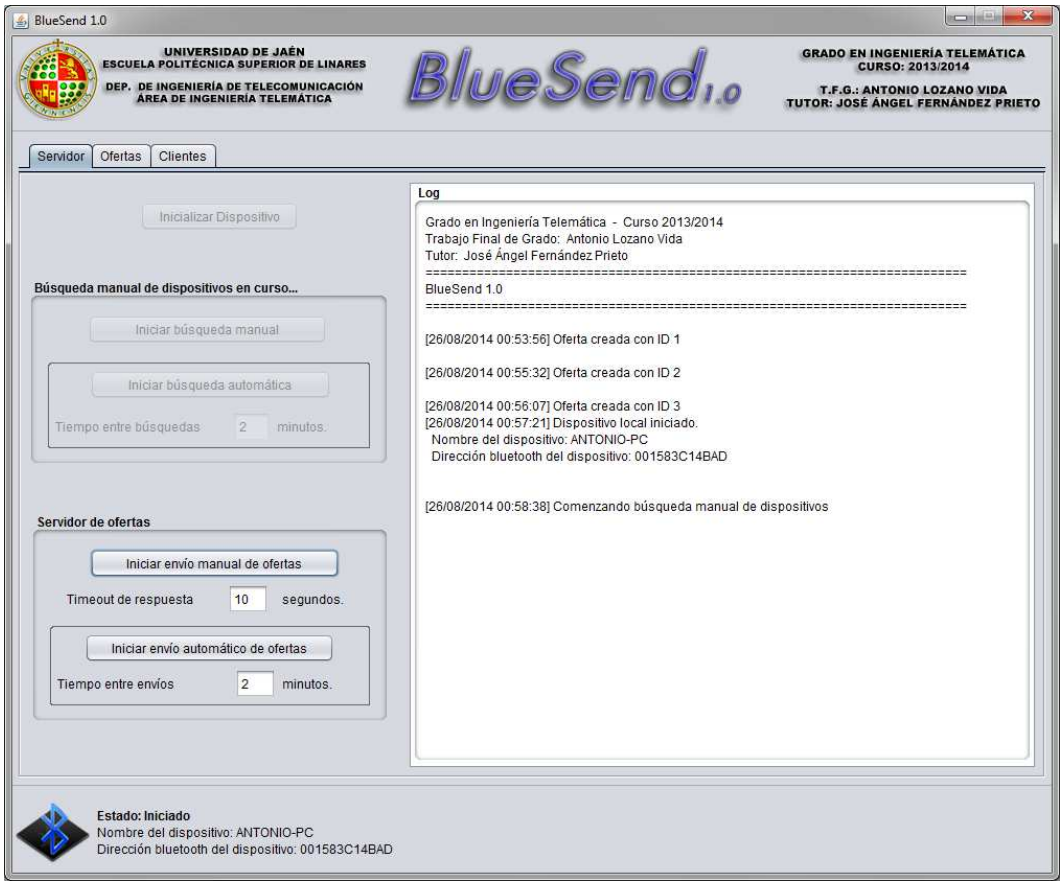


Figura 2.9: Sección Servidor al iniciar una búsqueda manual de dispositivos.

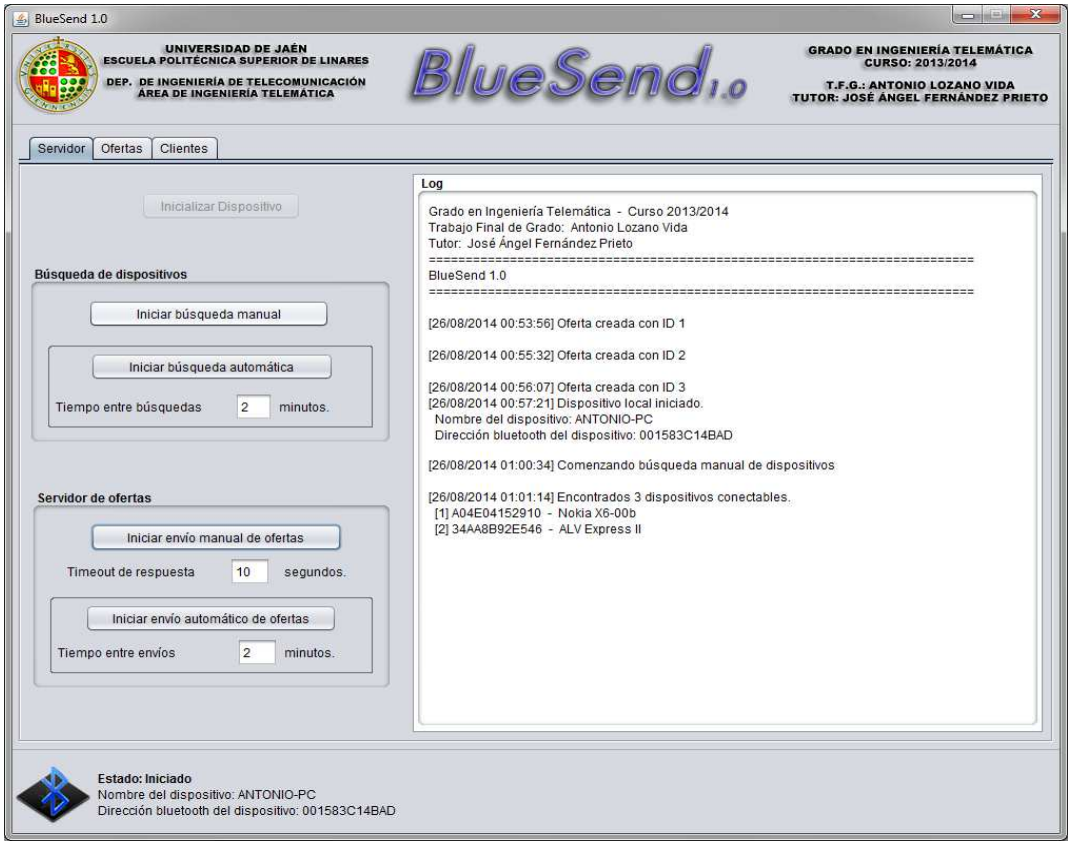


Figura 2.10: Sección Servidor al finalizar la búsqueda manual de dispositivos.

Servidor de ofertas

Debajo del *panel de búsquedas,* se encuentra el ***panel del servidor de* ofertas.** Posiblemente se trate de la parte más importante de la aplicación, pues mediante él se realizarán los envíos de las ofertas a los dispositivos previamente descubiertos. Al igual que ocurría con el *panel de búsquedas*, el usuario podrá iniciar un único ciclo de envío o un ciclo periódico continuo cada cierto tiempo.

Se dispone por tanto de un botón titulado ***Iniciar envío manual de ofertas***, mediante el cual el programa iniciará un único **ciclo de envío** de aquellas ofertas que se encuentren vigentes en ese momento. Al iniciar el ciclo de envío el programa tratará de enviar todas y cada una de las ofertas vigentes a todos y cada uno de los dispositivos registrados. Este envío se realizará de manera secuencial enviando oferta tras oferta a un mismo cliente y pasando posteriormente al siguiente. Sin embargo, en el caso de que un cliente no esté alcanzable (o bien no tenga activado su dispositivo Bluetooth), rechace una petición de envío o simplemente no la acepte en un tiempo determinado será ignorado para el resto de envíos del ciclo. De esta manera la aplicación será mucho más eficiente y ágil a la hora de enviar las ofertas, obviando esperas innecesarias.

El tiempo de espera para que el cliente acepte la petición de envío antes de que éste sea ignorado es configurable mediante el campo de texto titulado *Timeout de respuesta*, ubicado en el mismo panel. El valor que ha de tomar el parámetro deberá estar entre 10 y 60 segundos. Para cada uno de los envíos realizados con éxito y tras finalizar el ciclo completo se mostrará un mensaje en la ventana de Log indicado el fichero enviado y el cliente que lo recibió. Por otro lado, estos envíos serán asociados a cada cliente y registrados en la tabla de envíos de la sección *clientes.*

Por último, y de igual manera que ocurría con las búsquedas, se dispone en el mismo panel de un botón titulado ***Iniciar envío automático de ofertas***, mediante el cual el usuario podrá iniciar un ciclo continuo de envíos periódicos, los cuales se irán repitiendo de manera automática cada cierto tiempo, hasta que se dé la orden de detención pulsando de nuevo el mismo botón. El tiempo entre envíos es también configurable mediante el campo de texto ubicado en el mismo panel titulado *Tiempo entre envíos*, debiendo estar el valor comprendido entre 2 y 60 minutos.

En la *figura 2.11* se aprecia el estado que presenta la interfaz tras realizar un envío manual de ofertas, apreciándose en la ventana de Log dos ficheros enviados con éxito a un cliente. Por otro lado, en la *figura 2.12* se aprecian unas capturas del mensaje de aviso en el equipo cliente al recibir el fichero de una oferta, junto con el resumen de archivos recibidos, coincidiendo con los mostrados en la ventana de Log.

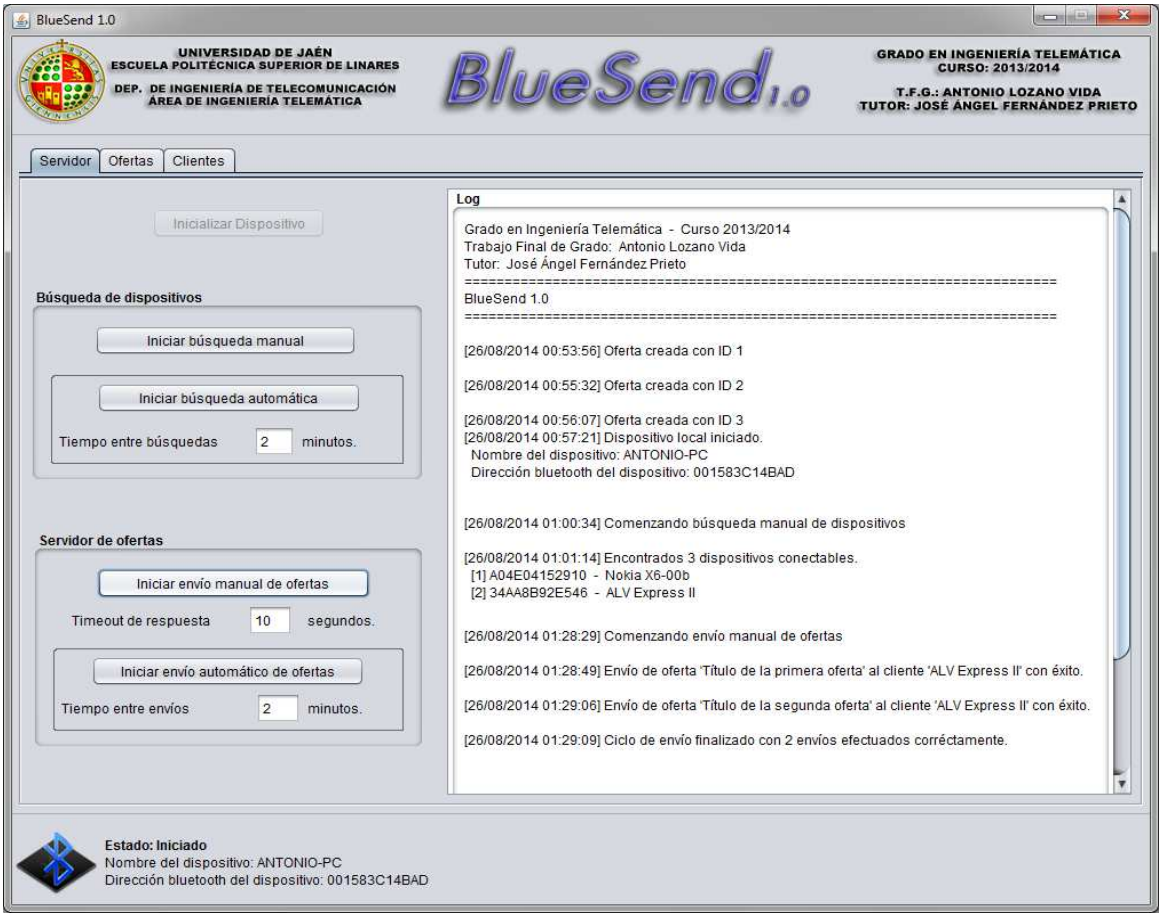


Figura 2.11: Sección Servidor tras realizar un envío manual de ofertas.

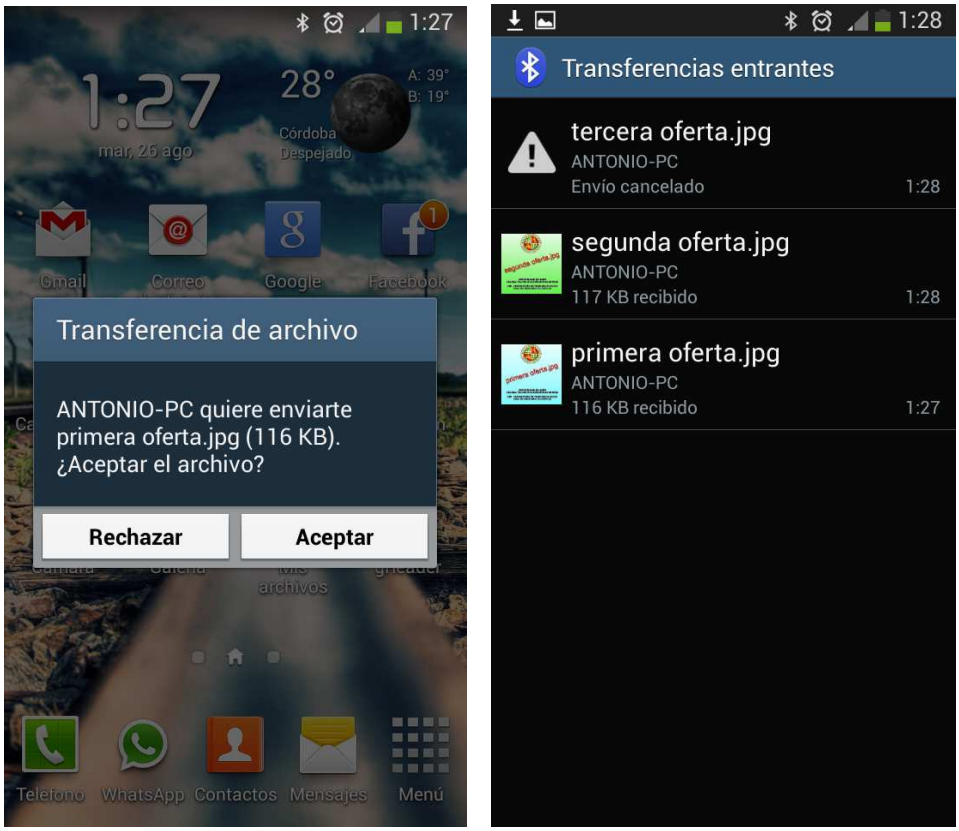


Figura 2.12: Aviso de envío de oferta en disp. cliente y lista de ficheros recibidos.

2.3. Manual de mantenimiento

En este manual se profundiza en los aspectos más relevantes de la estructura interna de la aplicación facilitando al administrador una visión global del funcionamiento de la misma. De igual manera, servirá como punto de partida a la hora de corregir posibles errores o acometer alguna mejora. Se hará una descripción de las funciones que desempeña cada una de las clases que la componen, además de de una introducción a la base de datos utilizada y sus características.

2.3.1. Funcionalidades de las clases

En estas líneas se describe una visión general las funciones que realiza cada clase en el conjunto total de la aplicación desarrollada. Los métodos que componen cada una de ellas están descritos en detalle en los comentarios que acompañan al código fuente de la aplicación.

Para una mayor comprensión del funcionamiento de la aplicación, se han incluido en los anexos una serie de esquemas y diagramas de flujo. Estos diagramas se corresponden con las líneas principales del código del programa, por lo que se facilita bastante la tarea de buscar fragmentos de código en la totalidad del mismo para futuras correcciones, mejoras y adaptaciones. Así pues:

En el ***anexo 2.4*** se puede apreciar un diagrama ilustrativo de la arquitectura de clases que intervienen en la aplicación y sus principales relaciones según el flujo del programa. Cada una de estas clases está comentada y explicada en el manual de mantenimiento (*apartado 2.3*).

En el ***anexo 2.5*** se incluye un diagrama de flujo general de la aplicación.

En los ***anexos 2.6 y 2.7*** se incluyen unos diagramas de flujo un poco más detallados sobre la parte de búsqueda de dispositivos y envío de ofertas.

CLASE IntBlueSend.class

Se trata de la clase contenedora de los elementos de la interfaz gráfica desde la cual el usuario interactuará con la aplicación. Al ejecutarse el método *main* de la clase, se creará una instancia de la misma, de forma que una vez creada dicha instancia se visualizarán los distintos elementos que contiene. Estos son, entre otros, un objeto de la clase *JFrame*, que se corresponde con el marco principal de la ventana de la aplicación. Dicho objeto contendrá a su vez a otro de la clase *JPanel*, dentro del cual quedan incrustados los demás elementos como las etiquetas, botones, imágenes, etc...

Dentro del panel principal, se encuentran otros tres paneles. Dos de ellos, del tipo *JPanel* serán los utilizados como cabecera y pié de la aplicación. El otro panel,

denominado *panelSecciones* es del tipo *JTabbedPanel* y es el panel que realmente incorpora todos los elementos que gestionan la aplicación. Éste incorpora a su vez otros tres paneles, uno por sección de la aplicación (servidor, ofertas y clientes).

- **Sección de servidor:** Esta sección es la que aparece por defecto activa al iniciarse la aplicación. Mediante ella se realizan las gestiones propias del servidor de ofertas. En primer lugar se dispone de un botón denominado *BotonInicioDisp*, cuya acción se encargará de inicializar el dispositivo Bluetooth local. Si la inicialización tiene éxito entonces activará el resto de elementos de la sección, los cuales precisan del dispositivo mismo para actuar. Los elementos que se activarán tras la correcta inicialización del dispositivo local serán el panel de búsquedas de dispositivos junto con el panel del servidor de envío de ofertas.
    - El **panel de búsquedas** dispone de dos botones, uno para realizar manualmente búsquedas puntuales de dispositivos Bluetooth conectables, y el otro para realizar búsquedas automáticas cada cierto tiempo. El tiempo a esperar entre búsquedas aparece en el panel como un campo de texto del tipo *JTextField*. Al pulsar uno u otro botón se lanzará un hilo paralelo independiente, que será el encargado de realizar la búsqueda de dispositivos conectables. De esta manera se consigue separar la ejecución de la interfaz gráfica de la ejecución de la búsqueda, evitando que la interfaz quede colgada esperado que ésta termine su ciclo. El hilo que realiza la búsqueda corresponde a una instancia de la clase *HiloDescubreDispositivos.class*, la cual realizará en primer lugar una búsqueda de dispositivos visibles, y en segundo una búsqueda del servicio de intercambio de ficheros OBEX en cada uno encontrado. Una vez finalizada la búsqueda realizará el almacenamiento de los dispositivos conectables en la base de datos de la aplicación. En caso de tratarse de una búsqueda automática, el hilo se dormirá después de cada ciclo de búsqueda el tiempo establecido, comenzando una nueva a continuación, y observando un flag que le indicará cuando ha de parar, el cual se activará al pulsar de nuevo el botón de búsquedas automáticas.
    - El **panel del servidor** de ofertas es similar al de búsquedas. Dispone de dos botones, uno para realizar un ciclo de envío de forma manual, y otro para iniciar ciclos de envío de forma automática cada cierto tiempo. Los parámetros configurables de este panel son el tiempo a esperar entre ciclos de envío así como el tiempo de respuesta máximo aceptable para cada intento de envío. Al igual que ocurre con las búsquedas, los envíos

se realizarán mediante el lanzamiento de un hilo independiente, correspondiente a la clase *HiloServFich.class*. Este hilo realizará una consulta a la base de datos obteniendo un listado de las ofertas que han de enviarse a cada cliente, procediendo a continuación a su envío correspondiente.

- Por último, en la parte derecha de esta sección se dispone de una **ventana de Log**, en la cual se escribirá información de los distintos eventos que vayan teniendo lugar en la aplicación, tales como búsquedas, envíos, altas de ofertas, modificaciones, etc.

- **Sección de ofertas:** es la segunda de las secciones de las que consta el panel principal. En ella se realiza la gestión de las ofertas por parte del administrador del programa. Los métodos relacionados con la gestión de las ofertas están recogidos en la clase *UtilOfertas.class* y serán llamados desde la interfaz según la acción que el usuario desee realizar.
- **Sección de clientes:** Se trata de la última sección de la aplicación, en la que se reúnen las funciones para la gestión de los clientes por parte del administrador del programa. Por otro lado, desde esta sección también se realiza la gestión de los envíos realizados con éxito a cada cliente. Los métodos relacionados con la gestión de los clientes y los envíos están recogidos en la clase *UtilClientes.class* y serán llamados desde la interfaz según la acción que el usuario desee realizar.

CLASE UtilOfertas.class

Esta clase contiene los métodos relacionados con la gestión de las ofertas. Los métodos creados en esta clase serán utilizados por las demás clases de la aplicación que los necesiten. Entre otros, se incluyen métodos para: dar de alta nuevas ofertas, rellenar y vaciar las tablas, modificar y eliminar ofertas, etc.

CLASE UtilClientes.class

Esta clase contiene los métodos relacionados con la gestión de los clientes, así como de los envíos asociados a los mismos. Los métodos creados en esta clase serán utilizados por las demás clases de la aplicación que los necesiten. Entre ellos, se incluyen métodos para: dar de alta nuevos clientes y envíos, rellenar y vaciar las tablas, modificar y eliminar registros, etc.

CLASE UtilDB.class

Esta clase contiene los métodos relacionados con la inicialización de la base de datos, así como los parámetros globales que se utilizarán para la conexión con la misma.

CLASE HiloDescubreDispositivos.class

Esta clase contiene un hilo que será lanzado desde la interfaz gráfica (clase *IntBlueSend.class*). Dicho hilo será el encargado de realizar las búsquedas de dispositivos conectables, para posteriormente proceder a la búsqueda del servicio de intercambio de ficheros OBEX. Los dispositivos encontrados (objetos *RemoteDevice*) se almacenarán en una lista (clase *Vector*) al mismo tiempo que se vayan encontrando. Una vez finalizada la búsqueda de dispositivos, se procederá a la búsqueda del servicio OBEX mediante la clase *ServiceSearch.class*.

CLASE HiloDescubreDispositivos.class

Esta clase contiene un hilo que será lanzado desde la interfaz gráfica (clase *IntBlueSend.class*). Dicho hilo será el encargado de realizar las búsquedas de dispositivos conectables, para posteriormente proceder a la búsqueda del servicio de intercambio de ficheros OBEX. Los dispositivos encontrados (objetos *RemoteDevice*) se almacenarán en una lista (clase *Vector*) al mismo tiempo que se vayan encontrando. Una vez finalizada la búsqueda de dispositivos, se procederá a la búsqueda del servicio OBEX mediante la clase *ServiceSearch.class*.

CLASE ServiceSearch.class

Esta clase contiene los métodos necesarios para realizar la búsqueda de servicios una vez finalizada la de dispositivos. La búsqueda de servicios será lanzada mediante el método *descubreServicios()*, el cual será llamado desde el hilo de la clase *HiloDescubreDispositivos.class* una vez finalizada la búsqueda.

CLASE DispRemoto.class

Los objetos de esta clase serán utilizados para almacenar conjuntamente los objetos de la clase *RemoteDevice* que representan a los dispositivos remotos encontrados en las búsquedas junto con sus respectivas propiedades, como son el nombre, la dirección Bluetooth y la url de conexión para el servico OBEX. Estos objetos serán almacenados y leídos durante las búsquedas de dispositivos y servicios, pudiendo acceder con mayor facilidad a cualquier dato de un dispositivo remoto.

CLASE HiloServFich.class

Esta clase contiene un hilo que será lanzado desde la interfaz gráfica (clase *IntBlueSend.class*). Dicho hilo será el encargado de realizar las veces de servidor de los ficheros asociados a las ofertas creadas. El ciclo de envío de ficheros constará en primer lugar de una consulta a la base de datos, la cual nos devolverá un listado de las ofertas a enviar atendiendo a los filtros marcados en ellas y en los clientes (periodo de validez, activación o desactivación de envío, etc), así como a la tabla de registros de envíos, de manera que una oferta no sea enviada dos veces al mismo usuario. Una vez obtenida la lista de ofertas a enviar se tratará de realizar el envío de cada una de ellas de manera secuencial, teniendo en cuenta que si un cliente no está conectable, rechaza el envío, o simplemente no lo acepta en un tiempo dado, el servidor lo ignorará durante ese ciclo y pasará al siguiente cliente. Esto es importante ya que de lo contrario el servidor podría quedar colgado esperando respuesta de un cliente ausente. Si el envío resulta exitoso se registrará en la base de datos para tenerlo en cuenta en el siguiente ciclo. Dependiendo del botón pulsado en la interfaz, el envío constará de un único ciclo, o bien de ciclos periódicos cada cierto tiempo.

CLASE HiloEnviaOferta.class

Esta clase contiene un hilo que será lanzado desde el hilo del servidor de ficheros (clase *HiloServFich.class*). Este hilo será el encargado de realizar el envío de un fichero determinado a un cliente determinado. El resultado de dicho envío (éxito o fracaso) lo indicará mediante un flag del hilo anterior. Aunque en principio el envío de los ficheros se podría realizar en el mismo hilo del servidor, debido a la naturaleza de la aplicación desarrollada, en la que se realizarán multitud de envíos en secuencia a diferentes clientes, resulta más eficiente realizar los envíos en hilos independientes para que, en caso de fallo en el envío, podamos interrumpir rápidamente el proceso y así liberar el socket creado al instante. De lo contrario los fallos en los envíos ralentizarían la secuencia de envío que maneja el servidor de forma significativa.

2.3.2. Base de datos H2

Tal y como se ha definido, el programa desarrollado ha de gestionar los datos referentes a:

- Los **clientes descubiertos** y su dirección de conexión.
- Las **ofertas creadas** por el administrador el programa.
- Los **envíos realizados correctamente** a los clientes.

Puesto que los datos a gestionar no son de gran envergadura, no tendría sentido utilizar una base de datos compleja o como un servicio independiente en el equipo, por lo

que se ha optado por el uso de una base de datos embebida en la aplicación. Actualmente existen multitud de opciones gratuitas para tal fin, siendo la base de datos *H2* la opción elegida, pues cubre sobradamente los requisitos de la aplicación y está especialmente desarrollada para Java. Este tipo de base de datos consiste en un fichero *jar* que es importado e instanciado desde el programa, y que lanzará de manera interna una interfaz para la gestión de los datos mediante lenguaje SQL. Así pues no tendremos que instalar ningún servidor independiente en el equipo, simplificando la aplicación y facilitando su instalación.

De esta manera, se han creado tres tablas, las cuales definen la estructura de datos que contendrá la información que se gestiona desde la aplicación. Es la misma aplicación la que directamente creará las tablas al ejecutarse por primera vez en caso de no estar creadas. A continuación se expone el código SQL encargado de crear la estructura de datos:

```
CREATE TABLE IF NOT EXISTS CLIENTE (
  dirCliente            VARCHAR(12) PRIMARY KEY,
  nombreCliente         VARCHAR(50) NOT NULL,
  urlCliente            VARCHAR(100) NOT NULL,
  activoCliente         BOOLEAN NOT NULL,
  );

CREATE TABLE IF NOT EXISTS OFERTA (
  idOferta              NUMERIC(5) PRIMARY KEY AUTO_INCREMENT,
  nombreOferta          VARCHAR(100) NOT NULL,
  ficheroOferta         VARCHAR(100) NOT NULL,
  fechaInicioOferta     DATE NOT NULL,
  fechaFinOferta                DATE NOT NULL
);

CREATE TABLE IF NOT EXISTS CLIENTEOFERTA (
  dirCliente    VARCHAR(12),
  idOferta      NUMERIC(5),
  CONSTRAINT CLIENTEOFERTA_PK PRIMARY KEY (dirCliente,idOferta),
  CONSTRAINT CLIENTE_FK FOREIGN KEY (dirCliente) REFERENCES
CLIENTE (dirCliente) ON DELETE CASCADE,
  CONSTRAINT OFERTA_FK FOREIGN KEY (idOferta) REFERENCES OFERTA
(idOferta) ON DELETE CASCADE);
```

Como se observa, se han definido tres tablas: La primera de ellas llamada ***CLIENTE*** almacenará los datos de los clientes detectados que ofrezcan el servicio de intercambio de ficheros OBEX utilizado para el envío de las ofertas. Cada cliente se identificará por su dirección MAC, que será la clave primaria de la tabla. Además, para cada cliente también se almacenará su nombre, que será el *friendly-name* del dispositivo, su url de conexión y una opción que representará su deseo de recibir o no las ofertas a enviar. La segunda tabla definida llamada ***OFERTA*** representa las ofertas creadas por el administrador. Cada oferta constará de un identificador numérico único que se generará de manera secuencial automáticamente y que será la clave primaria de la tabla. Además, para cada oferta se dispondrán de datos como un nombre, un periodo de validez y un fichero asociado, el cual será el que finalmente se envíe a los clientes. Para el almacenamiento de los ficheros asociados a cada oferta se creará un directorio junto a los ficheros que componen la aplicación, conteniendo en él los ficheros asociados a las ofertas creadas en ese momento. Por último, se define la tabla ***CLIENTEOFERTA***, la cual relaciona las dos anteriores y que almacenará los envíos realizados con éxito a los clientes, lo cual nos servirá para no repetir envíos previamente realizados.

Además del acceso interno que realiza el programa a la base de datos, ésta posee una interfaz de gestión web que nos brinda un acceso independiente para consulta o modificación de la misma. Para lanzar dicha interfaz, debemos hacer doble click sobre el fichero *h2-1.4.180.jar* incluido en el subdirectorio *lib* del programa. Al hacerlo se nos abrirá un navegador web, en el que debemos introducir los datos de la base de datos creada, siendo estos los siguientes:

- URL: *jdbc:h2:~/dbBlueSend/dbBlueSend*
- Usuario: *BlueSend*
- Password: *ujaen*

Es preciso que para poder acceder de manera externa se tenga cerrada la aplicación *BlueSend*, pues la base de datos no permite el acceso simultáneo a más de dos usuarios. A pesar de lo descrito, no se recomienda realizar modificaciones sobre las tablas y registros creados, pues podrían ocasionar un mal funcionamiento de la aplicación. En cualquier caso, ante un error irrecuperable, bastaría con eliminar las tablas y el directorio *ficherosOfertas*, ya que al abrir de nuevo el programa éste crearía de nuevo las tablas tal y como están configuradas por defecto. En las *figuras 2.13* y *2.14* se aprecian respectivamente una captura de la interfaz web de ingreso y de gestión de la base de datos:

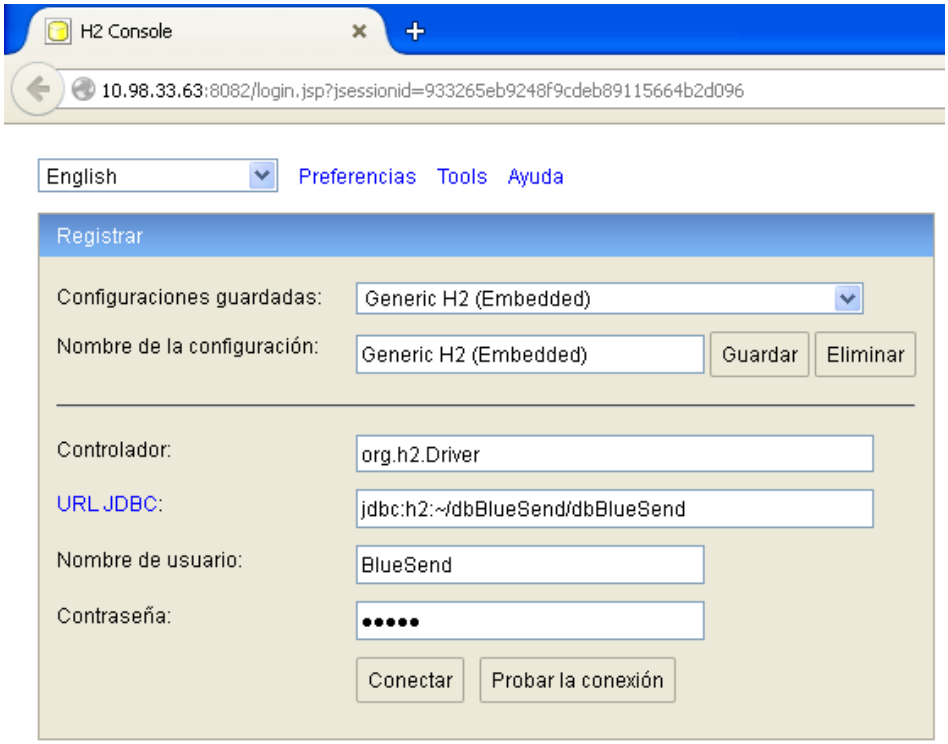


Figura 2.13: Interfaz de ingreso exterior de la base de datos H2

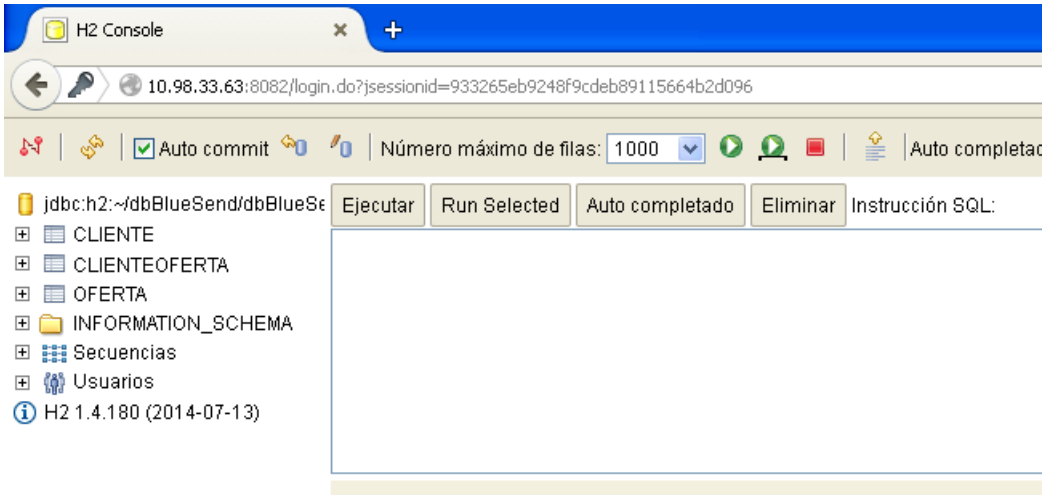
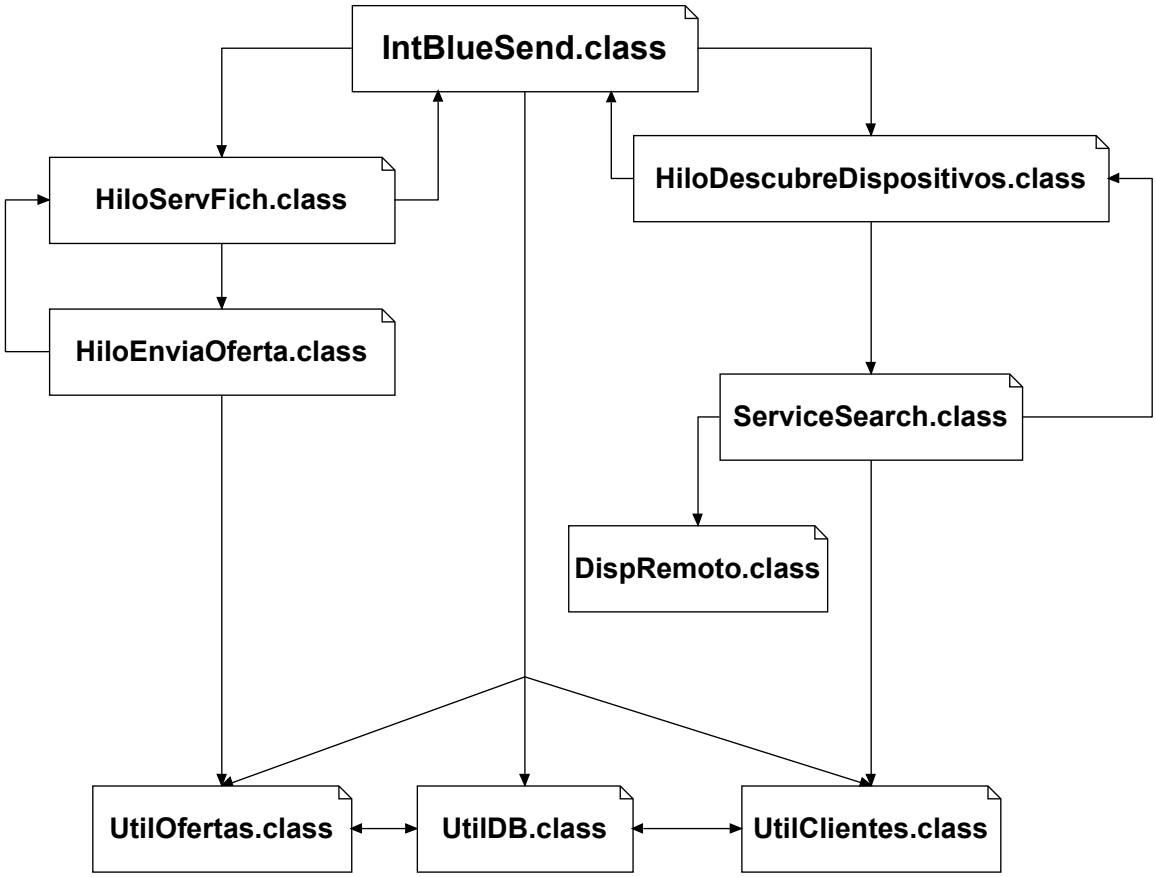
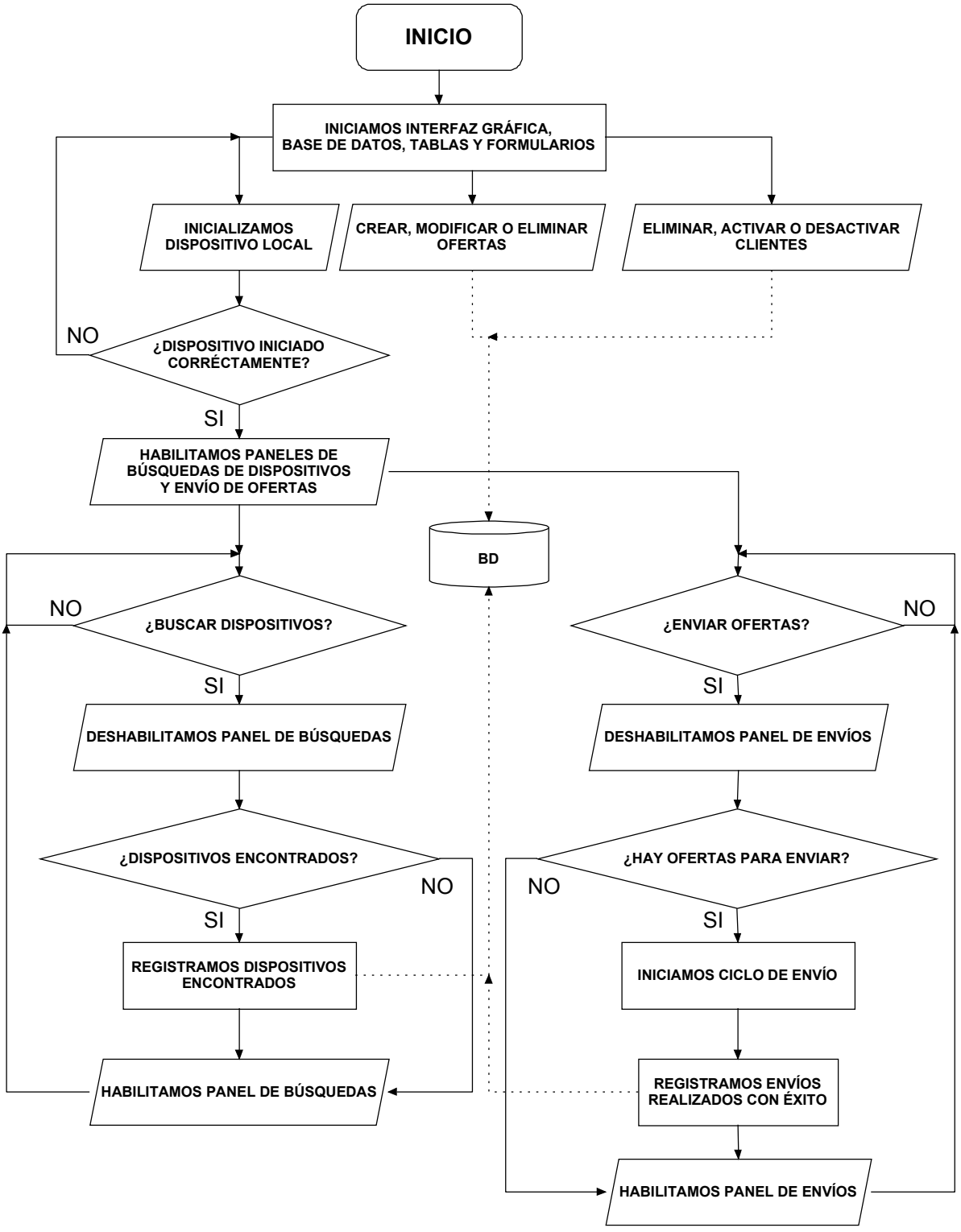


Figura 2.14: Interfaz de gestión exterior de la base de datos H2

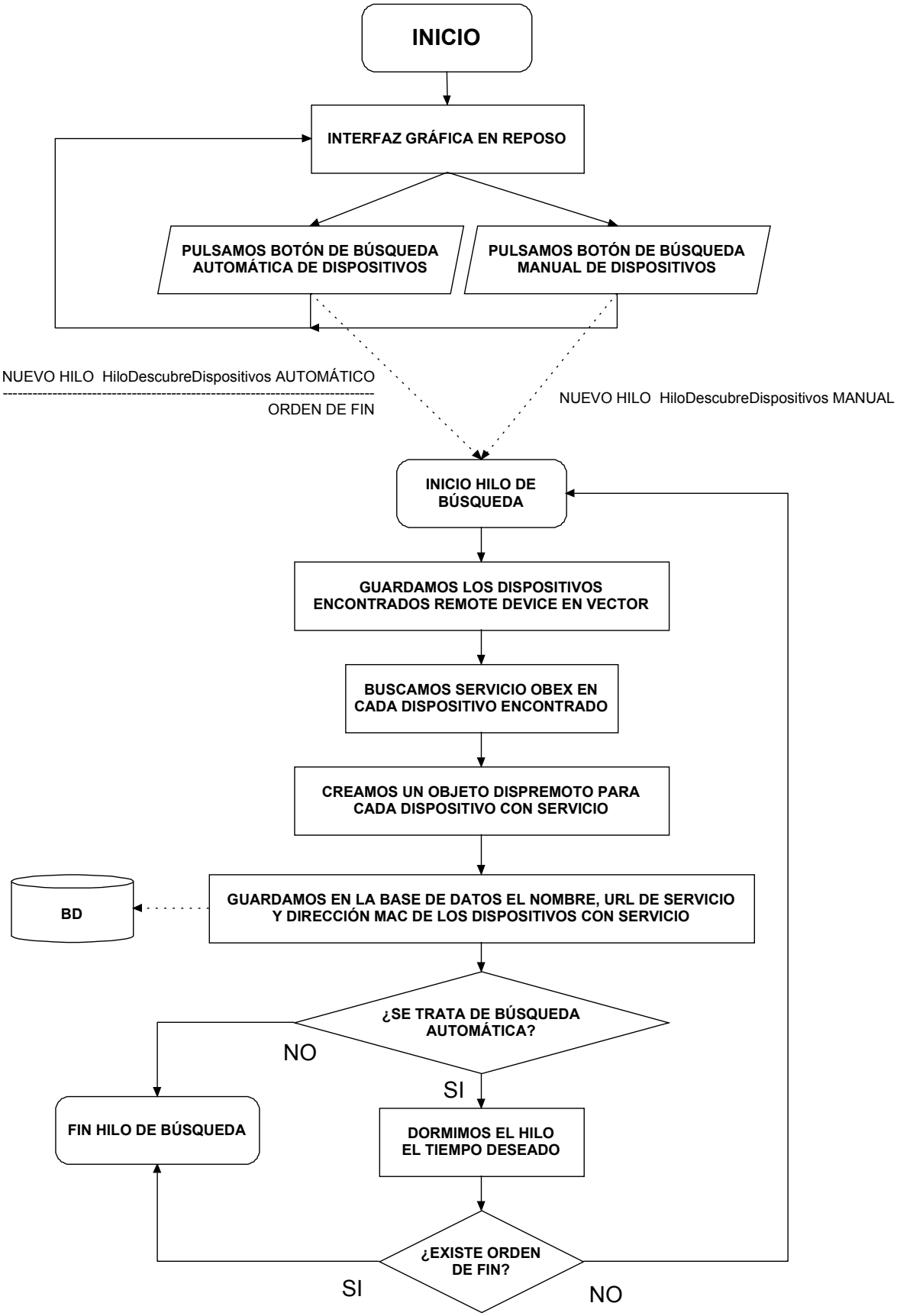
2.4. Arquitectura de clases que intervienen en la aplicación



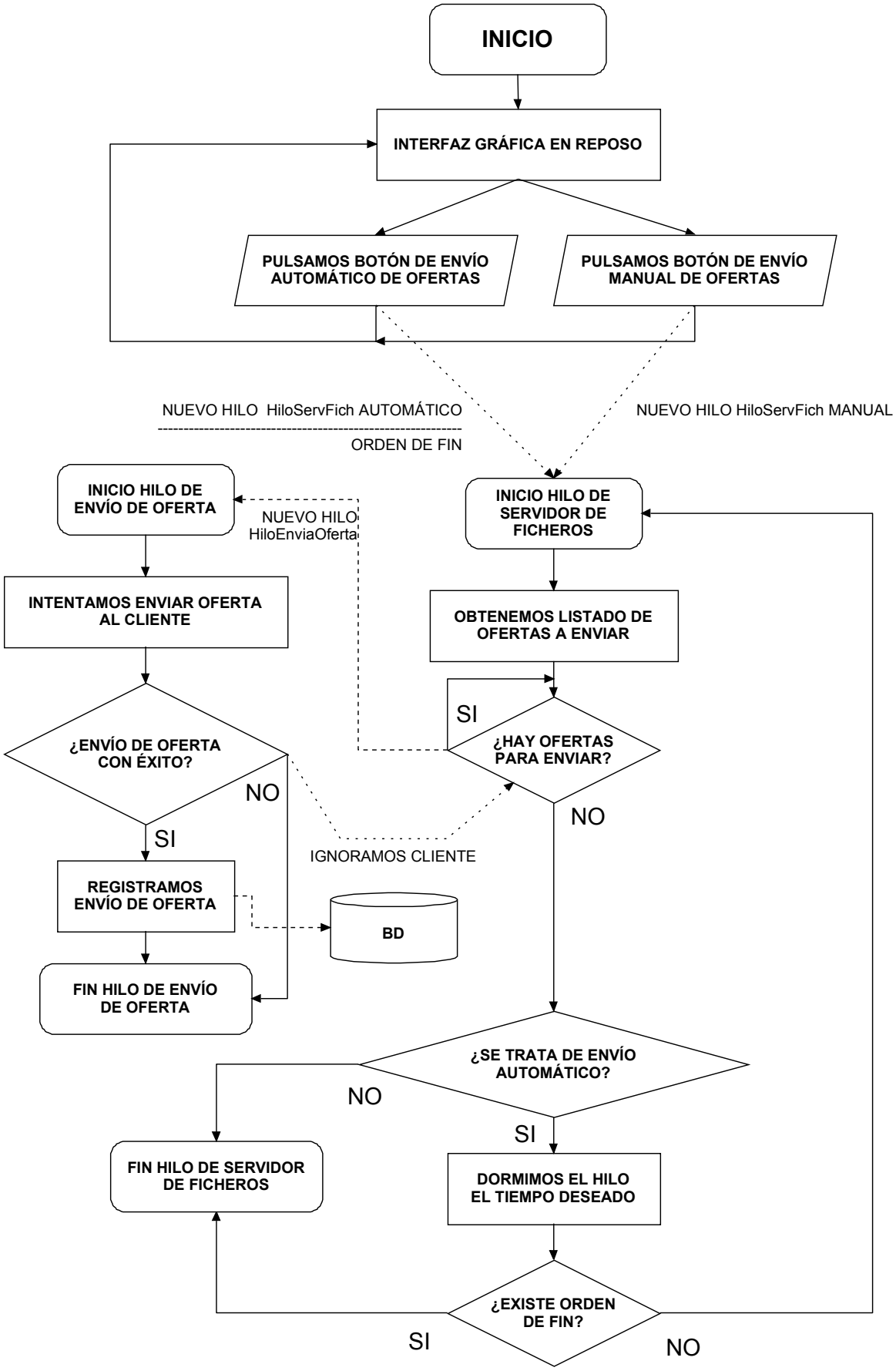
2.5. Diagrama de flujo general de la aplicación



2.6. Diagrama de flujo de descubrimiento de dispositivos



2.7. Diagrama de flujo de envío de ofertas



3. Pliego de condiciones

Los condicionantes técnicos necesarios para la correcta implementación del proyecto desarrollado se limitan a los requisitos mínimos de hardware y software que ha de disponer el equipo en el que se ha de instalar la aplicación.

Estos condicionantes han sido incluidos adicionalmente en el manual de instalación (apartado 2.1) y se resumen a continuación:

Parte Hardware

- Procesador Intel Pentium II o compatible a velocidad mínima de 266 MHz.
- 128 MB de memoria RAM.
- 200 MB de espacio libre en disco duro.
- Adaptador Bluetooth USB genérico V1.2 o superior.

Parte Software

- Sistema Operativo de 32 bits Windows 7 o XP Professional SP2.
- Java Runtime Environment 7 o superior.

4. Presupuesto

La siguiente tabla presenta el presupuesto necesario para la realización del proyecto:

CONCEPTO	COSTE	CANTIDAD	TOTAL
Horas de programación	45 €	240	10800 €
Horas de redacción	12 €	60	720 €
TOTAL (sin IVA)			11520 €
+ 21% IVA			2419.20 €
TOTAL (IVA incluido)			**13939.20 €**

Asciende el presente Presupuesto de Trabajo Final de Grado de *"Implementación de un servidor de ficheros vía Bluetooth"* a la cantidad de **TRECE MIL NOVECIENTOS TREINTA Y NUEVE EUROS CON VEINTE CÉNTIMOS**.