ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Programación de un sistema de Mandos de Respuesta basado en el uso de terminales móviles con Android

AUTOR: Julio Alberto Ruiz Navarro
DIRECTORES: Juan Carlos Sánchez Aarnouste
Pilar Manzanares López

Septiembre **/ 2011**

Autor	Julio Alberto Ruiz Navarro
E-mail del Autor	Julio_arn19@hotmail.com
Director(es)	Juan Carlos Sánchez Aarnouste, Pilar Manzanares López
E-mail del Director	juanc.sanchez@upct.es
Codirector(es)	Pilar.manzanares@upct.es
Título del PFC	Programación de un sistema de Mandos de Respuesta basado en el uso de terminales móviles con Android
Descriptores	

Resumen

Sistemas comerciales como EduClick ofrecen al entorno educativo diferentes herramientas interactivas que permiten fomentar la participación de los alumnos en el proceso de enseñanza/aprendizaje, a la vez que potencian la retención y facilitan información sobre la evolución de la actividad docente al profesor.

Entre estas herramientas se encuentran los sistemas de Mandos de Respuesta, que permiten realizar preguntas colectivas y recoger las respuestas emitidas mediante mandos electrónicos.

Android es un Sistema Operativo (S.O.) que está tomando mucha importancia recientemente. Este S.O. puede ejecutarse, entre otros, en smartphones (teléfonos móviles de última generación) basado en el núcleo Linux.

Una de las ventajas principales de este S.O. es que permite realizar aplicaciones por terceros (personas ajenas a las empresas que gestionan el S.O. y el teléfono en sí). Además, existe un completo entorno de desarrollo (SDK) así como un emulador muy potente para probar las aplicaciones desarrolladas antes de lanzarlas en el dispositivo final.

Dadas las características de los Smartphone (conexión a redes de datos de telefonía móvil, GPS, acelerómetro, brújula digital, etc.), así como la potencia ofrecida por el S.O., es posible crear aplicaciones en las que el límite lo pone nuestra imaginación.

Titulación	Ingeniería Técnica de Telecomunicación, esp. Telemática
Intensificación	
Departamento	TIC
Fecha de Presentación	Septiembre - 2011

Índice General:

Índice de Figuras:

Índice de Tablas:

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS.

1. Introducción

En las siguientes líneas se hace una breve introducción al presente proyecto, exponiendo cuál es su motivación, los contenidos ofrecidos en la memoria que nos ocupa, y los objetivos que persigue.

1.1. Introducción

En la actualidad, la tecnología está muy presente en la vida de todas las personas. Se ha convertido en una parte importante de la vida de cada una de esas personas que pueden interactuar con ella. Desde que se inventase el primer computador hasta hoy, se ha avanzado mucho en este camino, pues en aquellos días un ordenador ocupaba toda una sala. Gracias al avance de la investigación y de las técnicas de fabricación se ha hecho posible la creación de dispositivos con mejores prestaciones, en menos espacio, y a un precio más asequible.

Un ejemplo de esto son los Smartphone, dispositivos móviles inteligentes que incorporan todo tipo de prestaciones, así como GPS, Internet, Wifi, Bluetooth, etc. Actualmente se encuentran en auge, y cada día se sigue investigando para crear un Smartphone un poco más pequeño, más potente, con mejores prestaciones. Disponer de uno de estos dispositivos móviles resulta todo un gozo por las numerosas posibilidades que ofrece.

La docencia también ha visto un aliado en la tecnología, pues aporta una nueva forma de compartir conocimientos, haciendo las clases más dinámicas, permitiendo atraer el interés de los alumnos, y ayudándose de ésta para desarrollar su labor de forma más eficiente.

Actualmente hay muchas empresas en el mercado ofreciendo productos educativos vasados en la tecnología, y que sin duda son aceptadas por el nuevo enfoque que se da a la enseñanza.

1.2. Objetivos

En este proyecto se pretende desarrollar un sistema completo que permita realizar encuestas en el aula empleando teléfonos móviles de última generación. Para ello se debe desarrollar los siguientes apartados que se pueden considerar como objetivos principales del proyecto:

- Programar una aplicación para teléfonos móviles con S.O. Android que permita,
    - conectarse mediante Bluetooth con un servidor central
    - Identificarse frente a dicho servidor
    - Recibir las preguntas tipo test del servidor
    - Por último, enviar al servidor la respuesta seleccionada por el usuario
- Por otro lado, también será un objetivo de este trabajo crear un servidor central que realice las siguientes tareas:
    - Realizar una búsqueda de los dispositivos Bluetooth conectados (independientemente de si tienen S.O. Android)
    - Establecer conexión con todos los que pueda
    - Enviar las preguntas tipo test a todos los dispositivos conocidos
    - Recibir dichas respuestas y almacenarlas correctamente en una base de datos

Para conseguir todos estos objetivos, se consideran los siguientes objetivos adicionales:

- Aprender sobre el sistema operativo *Android*, el modo de programación en ese sistema operativo en dispositivos móviles y su entorno de desarrollo, ya que está en verdadero auge.
- Indagar en la programación sobre dispositivos móviles inteligentes o *Smartphone*, pues ofrecen un mundo de posibilidades y funcionalidades al alcance de la mano.
- Incrementar la formación sobre la programación orientada a objetos, puesto que se trata de lenguajes más cercanos al programador y que ofrecen más posibilidades.
- Aprender un nuevo lenguaje de programación, C#, famoso entre las aplicaciones de escritorio por su facilidad de diseñar componentes gráficos muy amigables y de diseño, y su eficiencia en la ejecución.
- De forma añadida, formarse en la utilización de entornos de desarrollo debido a que ofrecen posibilidades de depuración realmente interesantes, sobre todo en el

caso de *Visual Studio*, válido también para programar en otros lenguajes de programación.

1.3. Estructura de la Memoria

A continuación se explica brevemente el contenido de cada uno de los capítulos.

En el *CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS***.**, se expone la motivación del presente proyecto, una visión general de los contenidos de la memoria, y los objetivos que se persiguen.

En el *CAPÍTULO 2. TECNOLOGÍAS EMPLEADAS*, se explican las diversas tecnologías que se han usado en el proyecto.

En el *CAPÍTULO 3. DESARROLLO DEL SISTEMA*, se describe de forma detallada la el diseño inicial, problemas surgidos, solución propuesta, implementación del servidor y del cliente con sus clases, funcionalidad…

En el *CAPÍTULO 4. APLICACIÓN SERVIDOR*, se detalla el funcionamiento y se repasa la funcionalidad de la aplicación servidor.

En el *CAPÍTULO 5. APLICACIÓN CLIENTE*, se especifica el funcionamiento y funcionalidad de la aplicación cliente.

En el *CAPÍTULO 6. CONCLUSIONES Y LÍNEAS FUTURAS*, se reflexiona sobre los objetivos conseguidos y las dificultades encontradas, y se indican nuevas líneas de desarrollo.

CAPÍTULO 2. TECNOLOGÍAS EMPLEADAS

El desarrollo del siguiente proyecto ha originado la utilización de distinta tecnología para llevar a cabo los objetivos del mismo, y que ha propiciado la convivencia de todos estos para conseguir un objetivo común.

Gracias a la estandarización, se consigue que un PC y un dispositivo móvil puedan comunicarse sin problemas; en este proyecto, ha sido gracias a la tecnología *Bluetooth*, que ha proporcionado un medio de comunicación inalámbrico entre el PC empotrado y los diversos dispositivos móviles que se encuentren por la sala.

2.1. Android

Android es una plataforma de software y un sistema operativo (SO) diseñado para dispositivos móviles, es decir, teléfonos inteligentes o *Smartphone*, aunque su desarrollo se ha ampliado para soportar otros dispositivos tales como *tablets*, *netbooks*, PC'S, lectores de libros electrónicos, reproductores de audio MP3, … Este SO está basado en *GNU*/*Linux*, la mayor parte de la plataforma está disponible bajo la licencia de software libre *Apache* (y otras licencias de código abierto), y en la actualidad pertenece a *Google*. Tras de sí tiene una gran comunidad de desarrolladores de aplicaciones para extender la funcionalidad de los dispositivos que incorporan este SO, haciendo, en el ejemplo de los dispositivos móviles, que lideren las listas de ventas. Se encuentra en continuo desarrollo, existiendo a día de hoy numerosas actualizaciones, situándose su última versión estable en *2.3.6 Gingerbread* para dispositivos móviles.

La estructura del SO Android se compone de aplicaciones que se ejecutan en un framework Java de aplicaciones orientadas a objetos sobre el núcleo de bibliotecas de Java en una máquina virtual con compilación en tiempo de ejecución (*Dalvik*). Las bibliotecas, escritas en lenguaje de programación C, incluyen un administrador de interfaz gráfica, un *framework OpenCore*, una base de datos relacional *SQLite*, una API gráfica *OpenGL ES 2.0 3D*, un motor de renderizado *WebKit*, un motor gráfico SGL, SSL y una biblioteca estándar de *C Bionic*.

Las aplicaciones para este SO se desarrollan en Java, gracias a un SDK que puede ser integrado en entornos de desarrollo (como *Eclipse IDE*), y que proporciona las herramientas necesarias para crear y desarrollar aplicaciones para dispositivos con este SO. Además, también se pueden escribir aplicaciones en otros lenguajes de programación, pero este método no está soportado oficialmente por *Google*. Un ejemplo de esto es desarrollar las aplicaciones en lenguaje C y posteriormente se compilan en código nativo ARM para su ejecución.

2.1.1. Características de Android

Entre las características de Android se encuentran:

- La amplia variedad de diseños (VGA, librerías de gráficos 2D y 3D).
- Almacenamiento de datos en bases de datos (BBDD) *SQLite*.
- Conectividad (*GSM/EDGE, CDMA, EV-DO, UMTS, Bluetooth, Wifi, Wimax*).
- Mensajería (*SMS* y *MMS*).
- Navegador Web.
- Máquina virtual Java.
- Soporte de formatos (*MPEG-4, H.264, MP3, AAC, OGG, AMR, JPEG, PNG, GIF*).
- Soporte para hardware adicional (cámaras de video, pantallas táctiles, *GPS*, acelerómetros, …).
- Entorno de desarrollo (emulador, herramientas de depuración, perfiles de memoria y funcionamiento, plugin para Eclipse IDE).

2.1.2. Arquitectura de Android

La arquitectura de Android es la que aparece en la Figura 1, y como se puede ver se compone de una serie de componentes para llevar a cabo su ejecución de forma eficiente.

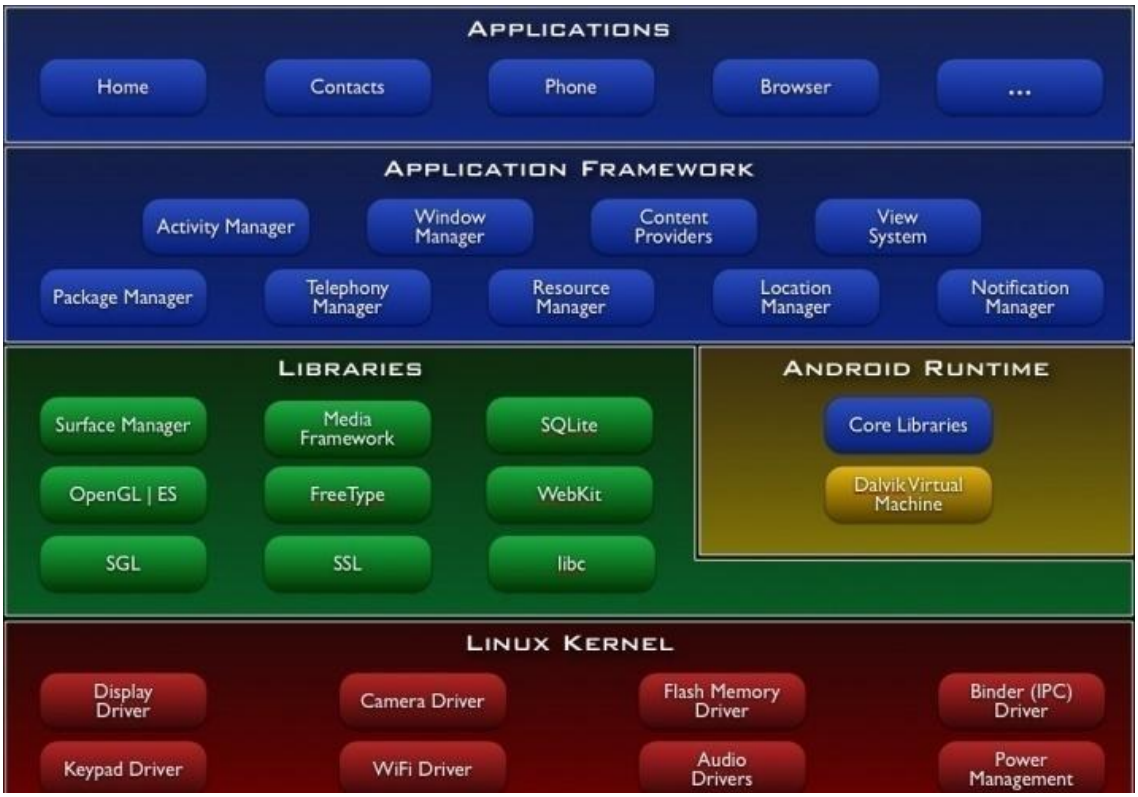

Figura 1: Arquitectura de Android

A continuación se detallarán los componentes principales de *Android*:

- **Aplicaciones:** *Android* dispone de una serie de aplicaciones base tales como un cliente de correo electrónico, aplicación de SMS, calendario, mapas, navegador, contactos, …, entre otras, desarrolladas todas ellas en lenguaje de programación Java.
- ***Framework* de las aplicaciones:** los desarrolladores de aplicaciones tienen acceso completo a los mismos APIs del *framework* empleados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes, de forma que cualquier aplicación puede publicar sus capacidades y otra aplicación distinta a ésta hace uso de las mismas (sujetas por supuesto a unas reglas de seguridad del *framework*). Además, este mismo mecanismo permite que los componentes sean remplazados por el usuario.
- **Librerías:** *Android* dispone de un set de librerías C/C++ usadas por varios componentes del sistema, las cuales se exponen a los desarrolladores a través del marco del *framework* de aplicaciones de *Android*, el cual interactúa con las librerías mediante *JNI* (*Java Native Interfaz*).
- ***Runtime* de Android:** set de librerías base que proporcionan la mayor parte de las funciones disponibles en las librerías base del lenguaje de programación Java. Cada aplicación corre su propio proceso, con su propia instancia de la máquina virtual *Dalvik*. Éste sido desarrollado de forma que un dispositivo pueda correr múltiples máquinas virtuales de forma eficiente, pues está optimizado para memoria mínima.
- **Núcleo Linux:** *Android* depende de un *Linux* versión 2.6 para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red, y modelo de *drivers*. También actúa como una capa de abstracción entre el *hardware* y el resto de la pila.

2.1.3. Android SDK

El kit de desarrollo de software (*software development kit* o *SDK*) incluye un conjunto de herramientas de desarrollo, tales como un *debugger*, librerías, documentación, códigos de ejemplo, tutoriales y un emulador (basado en *QEMU*). Está soportado en SO *Windows*, *Linux* y *Mac*. El entorno de desarrollo (*Integrated Development Environment* o *IDE*) oficialmente soportado es *Eclipse* conjuntamente con el *plugin ADT* proporcionado por *Google*.

2.1.4. Entorno de desarrollo

El entorno de desarrollo empleado para desarrollar la aplicación *android* ha sido el IDE Eclipse, un entorno de desarrollo integrado de código abierto multiplataforma.

A la hora de crear una aplicación, hay que tener en cuenta que el aspecto gráfico se hace por un lado, en lenguaje *xml* dada la pobreza del editor gráfico, y el código que ejecuta la aplicación por otro. De esta forma, todo lo referente a componentes gráficos se desarrolla y guarda en la carpeta *res*, mientras que el código fuente se guarda en *src*.

Aunque pueda parecer confuso esta separación, una vez trabajas un poco con la herramienta no supone ningún problema.

2.1.5. Smartphone

En la actualidad, los dispositivos móviles o Smartphone que implementan el sistema operativo *android* están en lo más alto del mercado, pues disponen de buenas prestaciones de por sí solos, además de la experiencia que supone para el usuario utilizar este sistema operativo, pues exprime las capacidades del terminal ofreciendo las mayores prestaciones, y manteniendo un consumo con respecto a otros Smartphone muy bueno.

2.2. Bases de Datos

Se puede definir *Base de Datos* como un conjunto de datos que pertenecen a un mismo contexto, y que se almacenan de forma sistemática para su uso posterior. Normalmente este término siempre se asocia a bases de datos en formato digital, pues con el desarrollo de la informática y la electrónica, se ha incrementado mucho el uso de éstas en el formato mencionado debido a la gran variedad de soluciones que ofrece a la hora de almacenar datos. En prácticamente todos los escenarios donde se requiere el almacenamiento de información se utilizan bases de datos.

Como se ha mencionado, el uso de bases de datos en formato digital está muy extendido debido a la eficacia y rapidez con la que poder almacenar, recuperar y tratar datos almacenados gracias a programas desarrollados para tal fin, denominados *Sistemas Gestores de Bases de Datos* (*SGBD*), tales como *Microsoft Access*, *SQLite*, *MySQL*, *Open Access*, *Oracle*…

2.2.1. MySQL

Para el presente proyecto se ha empleado la base de datos *MySQL* porque es una de las más populares, pues es de código abierto ofreciendo soporte para más de 20 plataformas como *Linux*, *Windows*, *NetWare*, *OS/X*, … y se pueden usar infinidad de lenguajes de programación para acceder a *MySQL*. Existen interfaces para *C*, *C++*, *PHP*, *C#*, …, y para los lenguajes que no poseen interfaz es posible acceder a través de *ODBC*. Es multihilo, por lo que es capaz de trabajar con más de un procesador simultáneamente. Ofrece un alto rendimiento, fiabilidad y facilidad de uso, proporcionando además una amplia gama de herramientas de bases de datos. Debido a estas características, es la base de datos de elección para muchas empresas.

2.2.2. Librería MySQL para C#

Para trabajar con bases de datos *MySQL* desde *C#* se dispone de amplia información así como ejemplos, e incluso conectores, de forma que instalando el paquete correspondiente, se incrusta en el entorno de desarrollo ofreciéndote nuevas

posibilidades. Además, también instala las librerías necesarias para desarrollar aplicaciones que utilicen bases de datos, como la que nos ocupa.

Las librerías añadidas a la aplicación servidor son:

- *mysql.data.dll*
- *mysql.data.entity.dll*
- *mysql.visualstudio.dll*

2.2.3. Diseño de base de datos adoptado

Se ha optado por no inundar la base de datos con tablas, y tener los datos de esta forma más compactos, aunque dependiendo de la envergadura que ganen, las consultas a la base de datos pueden ser muy costosas.

Así, en la base de datos solo podrá haber una tabla de alumnos. Esta tabla estará formada por cinco campos:

- *MAC*: dirección MAC del dispositivo móvil del alumno. Tipo *text*.
- *Nombre*: nombre del alumno. Tipo *text*.
- *Apellidos*: apellidos del alumno. Tipo *text*.
- *DNI*: DNI o NIF. Tipo *text*.
- *identificador*: identificador único de alumno. Tipo *int*. *Auto-increment*. Este valor lo asigna la base de datos.

Los alumnos se agregan de maneara automática a esta tabla al contestar a una pregunta, por lo que no será necesario incorporarlos de forma manual.

Para las preguntas, éstas se organizan por temas, es decir, se creará una tabla para las preguntas del tema1, otra tabla para las preguntas del tema2, … Los campos de esta tabla serán:

- *NumPreg*: número o identificador de pregunta en la tabla del tema al que se refiere. Este número es elegido por el usuario, no por la base de datos. Es de tipo *int*.
- *Pregunta*: de tipo *text*, almacenará el enunciado de la pregunta.
- *Respuesta Correcta*: de tipo *text*, contiene la respuesta correcta a la pregunta *formulada*.
- *Respuesta1*, *Respuesta2*, …: de tipo *text*, contendrá las posibles respuestas para la pregunta propuesta.

Según se ha hecho en el proyecto, cuando se quiere cargar una pregunta, siempre se ha buscado por el número de pregunta dentro del número de tema, es decir, tema1 pregunta3. La consulta se realizará extraordinariamente rápida, pues solo tiene que buscar un campo dentro de una tabla.

Cuando se crea la tabla de preguntas, automáticamente se crea también la tabla de respuestas. Habrá una tabla de respuestas para las preguntas del mismo tema, de forma

que cuando se propone una pregunta, si no dispone ya de columna, se añade la columna con el número de pregunta. Los campos para esta tabla son:

- *identificador*: de tipo *int*, en esta columna se guarda el identificador de usuario que ha contestado a la pregunta correspondiente.
- *Pregunta1*, *Pregunta3*, …: de tipo text, recoge la respuesta que el alumno ha dado a esa pregunta.

Así, en una fila disponemos de, en la primera columna, el identificador de usuario, y en el resto de la fila las respuestas que ha dado a cada una de las preguntas.

Las tablas se relacionan por el identificador, es decir, la relación existente entre la tabla de alumnos y la tabla de respuestas es el identificador.

2.3. C# y Entorno de desarrollo

C # (también conocido como C Sharp) es un lenguaje de programación orientado a objetos. Este lenguaje ha sido desarrollado por Microsoft como parte de su plataforma .NET. Su sintaxis básica deriva de C/C++, y utiliza el modelo de objetos de la plataforma .NET, similar a la de Java, aunque incluye mejoras derivadas de otros lenguajes. Dispone de multitud de librerías, y es asombroso lo mucho que se puede llegar a hacer en pocas líneas de código, pues es un lenguaje potente.

Además, para desarrollar aplicaciones en este lenguaje se dispone de un IDE (entorno de desarrollo integrado), *Visual Studio*, que mejora aún más la experiencia, si cabe. Ayuda mucho a la hora de crear aplicaciones gráficas, pues para diseñarlas solo hay que limitarse a arrastrar los componentes a la ventana de diseño, y modificar las propiedades oportunas; y todo esto sin interactuar con el código, porque *Visual Studio* automáticamente te genera todo el código de la interfaz gráfica, quedando para el programador el desarrollo de la funcionalidad que quiera dar a todos esos componentes, que no es poco.

A la hora de desarrollar aplicaciones, este entorno de desarrollo ofrece sugerencias y ayuda, es decir, se crea un objeto y aporta información de todos los métodos y variables que tiene, con la explicación de cada uno de ellos, indicando los parámetros de entrada y salida, posibles excepciones, ...

Y cuando se tiene desarrollada la aplicación y llega la hora de hacer pruebas y comprobar el funcionamiento, ofrece un mundo de posibilidades, pues permite depuración, vista del valor de cada variable, permite realizar edición de código sin tener que parar la ejecución del mismo, … Sin duda, toda una experiencia para el desarrollador.

2.4. Bluetooth

Bluetooth es una tecnología que permite la conexión entre dispositivos de forma inalámbrica. Emplea ondas de radio de corto alcance, de 2,4 a 2,48 GHz de la banda ISM, lo que le da una compatibilidad universal entre dispositivos Bluetooth, puesto que la banda ISM está disponible a nivel mundial. Puede alcanzar distancias de 10 metros manteniendo un consumo bajo. Puede llegar hasta los 100 m si se incrementa mucho el consumo de batería.

Para evitar interferencias entre dispositivos y protocolos que operan en la misma banda de frecuencias, Bluetooth emplea la técnica de salto de frecuencias (FHSS) consistente en dividir la banda en 79 canales de 1 MHz y llevar a cabo 1600 saltos por segundo.

Los dispositivos Bluetooth pueden clasificarse en tres clases, dependiendo de la distancia que cubren, o la potencia transmitida.

Clase	Distancia (m)	Potencia tx (mW)
clase 1	100	100
clase 2	10	2,5
clase 3	1	0,1

Tabla 1: clases de dispositivos Bluetooth

También pueden clasificarse según el ancho de banda:

Versión	Ancho de Banda
Versión 1.2	1 Mbit/s
Versión 2.0 + EDR	3 Mbit/s
Versión 3.0 + HS	24 Mbit/s

Tabla 2: clasificación Bluetooth por ancho de Banda

A diferencia de otras tecnologías LAN inalámbricas, como Wi-fi, diseñadas para dispositivos que se hallen dentro o alrededores de un mismo edifico, los dispositivos que utilizan rede PAN inalámbricas IEEE 802.15 podrán comunicarse en cualquier parte del mundo.

Varios dispositivos Bluetooth pueden conectarse entre sí cuando uno esté dentro del radio de cobertura del otro. Pueden comunicarse hasta ocho dispositivos entre ellos y formar una red *Piconet* o *Picorred*. La unión de varias de estas Picones se denomina *Scatternet* o *Red Dispersa*. Dentro de una red Piconet, los dispositivos pueden comportarse como maestro o esclavo, pero solo puede haber un dispositivo maestro en la red Piconet, por lo que el resto serán esclavos.

Este dispositivo maestro se utiliza para sincronizar al resto de dispositivos maestros con su reloj y patrón de saltos. Todos los dispositivos de una red piconet están sincronizados desde el punto de vista del tiempo y de la secuencia de saltos entre

canales. Cada unidad dispone de un reloj de sistema interno que determina la temporización y la secuencia de saltos que debe seguir el transceptor.

En cuanto a velocidad de transmisión se refiere, Bluetooth soporta con la especificación actual 3 Mbps para el traslado de datos. Está orientado sobre todo a aplicaciones de voz.

2.4.1. Librería Bluetooth para C#

La librería empleada para el manejo de dispositivos Bluetooth ha sido la librería *InTheHand.Net.Personal*, puesto que se trata de una librería gratuita, a diferencia de otras, que cumple con las necesidades de la aplicación.

Dispone de multitud de objetos y tipos de datos de Bluetooth para llevar a cabo un amplio abanico de posibilidades.

CAPÍTULO 3. DESARROLLO DEL SISTEMA

3.1. Arquitectura

El sistema con el que se da solución al proyecto es un sistema formado por un servidor central, ejecutado en un PC, encargado de emitir información (preguntas a responder) y recoger la que envíen los clientes (respuesta introducida por el usuario para la pregunta a responder), tratándola de forma adecuada (guardarlas en una base de datos); y la aplicación cliente, que será la que se ejecutará en los dispositivos móviles, y que recogerá la pregunta emitida por el servidor central, recogerán la respuesta del usuario y la enviarán de vuelta al servidor central, recogiendo éste la respuesta emitida. Y todo esto sin mantener la conexión, es decir, el dispositivo móvil se conecta, envía/recibe la información, y se desconecta. Por tanto, la arquitectura con el que se da solución al proyecto es una arquitectura *cliente-servidor*, en la que los dispositivos móviles se conectan al servidor central para recoger la pregunta o para enviar la respuesta, según proceda, y el servidor central atiende a los clientes que se conecten a él. En la siguiente figura (*Figura 2*) se muestra la arquitectura seguida.

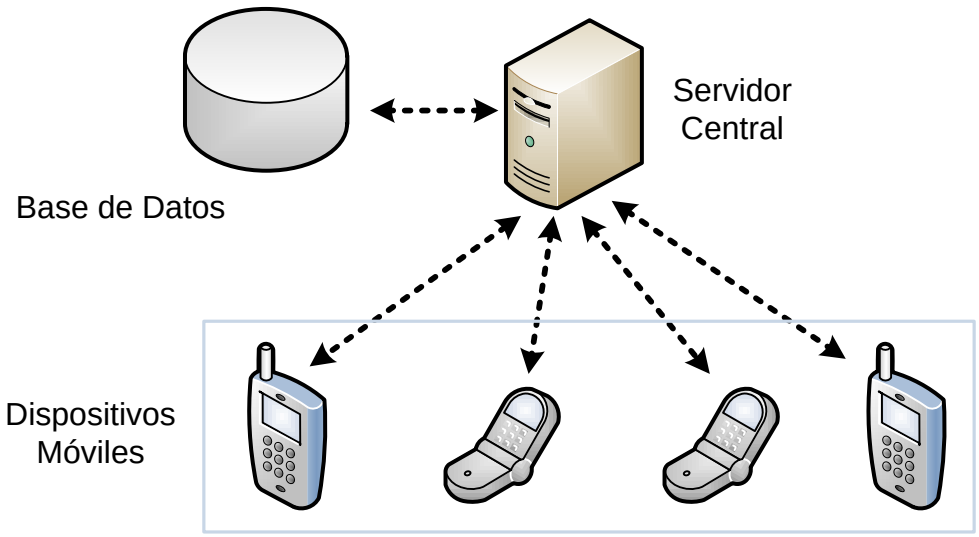

Figura 2: Arquitectura *cliente-servidor*

Los datos que se intercambiarán entre servidor central y dispositivo móvil será el que muestra la siguiente figura (*Figura 3*).

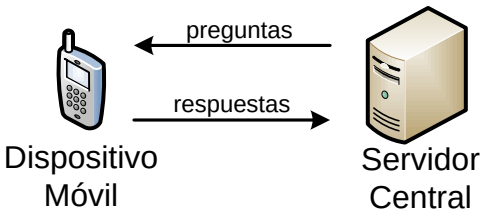

Figura 3: intercambio de datos

3.2. Problemas

El primero de los problemas surge en el planteamiento de la aplicación, pues, como se ha explicado en el capítulo de *Bluetooth* (*2.4. Bluetooth*), conforme aumenta el número de conexiones simultáneas se tarda más en realizar la conexión, además de que no es posible conectar más de 8 dispositivos simultáneos, por lo que, si normalmente en una clase hay alrededor de 25 o 30 alumnos, se tardará demasiado tiempo en atenderles, y no se podrán mantener tantas conexiones simultáneas.

Durante el desarrollo del proyecto surgió un problema que llevó a, manteniendo la arquitectura *cliente-servidor*, cambiar el sentido de ésta. Es decir, mientras que en la arquitectura inicial se pretendía que el servidor central hiciese de servidor, desde el punto de vista del desarrollo del sistema no es así, pues hubo un problema con la función que se utiliza en *android* para conectar con otros dispositivos que dispongan de *Bluetooth*, y no se conseguía que el dispositivo móvil iniciase la comunicación con el servidor. La solución propuesta para salvar este problema se explica en el capítulo que sigue (*3.3. Solución Adoptada*).

3.3. Solución Adoptada

Para solucionar el primero de los problemas planteados se opta por no mantener las conexiones, es decir, conectarse, servir al cliente, y desconectarse, pues de esta forma se permite la atención a numerosos clientes.

Para solucionar el problema de comunicación *Bluetooth* en los dispositivos móviles, se ha cambiado el modo en el que se conectan los dispositivos móviles al servidor. Es decir, es el servidor el que inicia la conexión con el dispositivo móvil, y este último el que espera a que se conecten a él. Para hacer este proceso de modo eficiente sin que el servidor tenga que molestar a los dispositivos móviles continuamente para ver si esperan una pregunta o quieren enviar una respuesta, se juega con las posibilidades de visibilidad e identificación de servicios de *Bluetooth*, explicados en el capítulo *2.4. Bluetooth*. Los detalles sobre cómo se ha seguido este esquema de modo eficiente se pueden encontrar en los capítulos de desarrollo de la aplicación servidor (*3.4. Desarrollo de Aplicación Servidor*) y de la aplicación cliente en el dispositivo móvil (*3.5. Desarrollo de Aplicación Cliente*).

3.4. Desarrollo de Aplicación Servidor

La labor del servidor será, como se ha mencionado en capítulos previos, servir a los clientes (dispositivos móviles) que lo soliciten, enviándole la pregunta o recogiendo las respuesta. Esta es una descripción muy simplificada de la labor del servidor, pues también dispone de multitud de opciones para crear la información, guardarla y tratarla. A continuación se explican los detalles de desarrollo e implementación del servidor.

La aplicación servidor es una aplicación de escritorio, es decir, una aplicación gráfica en la que el usuario interactúa con la aplicación de forma más amigable que a través de consola, permitiendo así poder visualizar de forma más estética y ordenada la información tratada, por ejemplo, la pregunta que se muestra en la aplicación (y que enviará a los clientes), información guardada en la base de datos, etc.

Para llevar a cabo una estructura más clara en el desarrollo de la aplicación completa, se han implementado una serie de clases que pretenden encapsular el funcionamiento de determinadas opciones de la aplicación. Así, la relación entre clases es la que se detalla a continuación (

Figura **4**). Se omiten los atributos y métodos de las clases, pues serán explicadas con detalle a continuación.

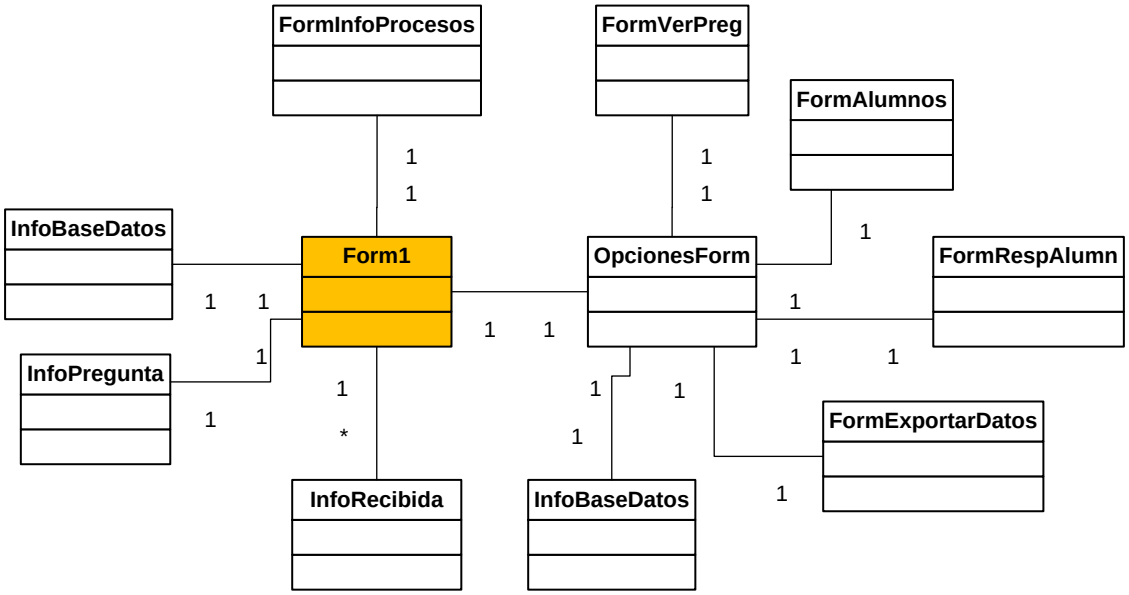

Figura 4: Clases implementadas y relación entre ellas

La clase principal es la clase *Form1* resaltada en la figura anterior. En ella residen multitud de variables y métodos accesibles desde otras clases (lo cual quiere decir que esos métodos y variables serán públicos) para poder controlar multitud de parámetros de forma sencilla y centralizada al residir en una sola clase.

A continuación se explicará cada una de las clases desarrolladas con detalle, explicando cada una de las variables y métodos implementados para llevar a cabo su labor.

3.4.1. Clase infoBaseDatos

Esta clase se ha diseñado con la misión de guardar la información de conexión con la base de datos, es decir, aquellos parámetros que es necesario indicar para realizar la conexión con la base de datos y posteriores consultas, tales como dirección IP, usuario, etc. En la descripción de variables de la clase que nos ocupa se indican los necesarios.

Variables:

Las variables de la clase, como se ha mencionado, son las necesarias para realizar la conexión con la base de datos. Éstas variables serán privadas (*private*) de forma que solo puedan ser accesibles desde la presente clase (aunque se han declarado métodos para poder acceder a ellos). Dichas variables son:

- *String address*: variable tipo *String* que contendrá la dirección IP de la máquina donde se encuentre el servidor, puesto que no tiene por qué ser la local.
- *String basedeDatos*: variable tipo *String* que albergará el nombre de la base de datos a la que se quiere realizar la conexión.
- *String Usuario*: variable tipo *String* donde se almacenará el nombre de usuario para realizar autenticación en la conexión con la base de datos.
- *String Contraseña*: variable tipo *String* que guardará la contraseña para realizar autenticación en la conexión con la base de datos.
- *String Puerto*: variable tipo *String* que indicará el número de puerto en el servidor donde se encuentra la base de datos para realizar la conexión.
- *Boolean inicializada*: variable *Boolean* para conocer el estado de esta clase, es decir, si se ha inicializado o no. Toma el valor *false*.

Constructor:

No se ha declarado ningún constructor, por lo que permanece el constructor por defecto, es decir, el nombre de la clase sin pasar ningún parámetro.

Métodos Get/Set:

Para hacer las variables accesibles se ha optado por implementar métodos *get* para consultar el valor de estas variables, y métodos *set* para establecer un nuevo valor. Los métodos *get* son los siguientes:

- *public Boolean getInicializ()*
- *public String getAddr()*
- *public String getBaseDatos()*
- *public String getUsuario()*
- *public String getPass()*

- *public String getPuerto()*

Por otra parte, los métodos *set* son:

- *public void setInicializ(Boolean b)*
- *public void setAddr(String addr)*
- *public void setBaseDatos(String BD)*
- *public void setUsuario(String User)*
- *public void setPass(String Pass)*
- *public void setPuerto(String puerto)*

3.4.2. Clase infoPregunta

Para guardar la información de la pregunta a la que se tiene que responder se ha implementado la esta clase, pues no solo se tiene la pregunta sino también están la respuesta correcta y las posibles respuestas.

Variables:

Así, las variables declaradas para guardar la información necesaria serán privadas (*private*) y de tipo *String*, y son:

- *numTema*: guardará el número de tema al que corresponde la pregunta a contestar. Se inicializará en el constructor.
- *numPreg*: número de pregunta a la que contestar. Se inicializará en el constructor.
- *pregunta*: pregunta a contestar. Se inicializará en el constructor.
- *respCorrec*: respuesta correcta de la pregunta a contestar.
- *String[] respuestas*: *array* de *String* que contendrá las posibles respuestas que se pueden elegir para la pregunta a contestar. Se inicializará a través de un método *set*.
- *separa*: carácter de separación utilizado para separar campos en la cadena que se enviará al cliente con la información de la pregunta a contestar. Está inicializada a "*n*".

Constructor:

El constructor de esta clase hay que facilitarle una serie de argumentos para inicializar el objeto creado, siendo el constructor de la forma *public infoPregunta(String numTema, String numPreg, String Preg, String respCorr, int numresp)*. Los argumentos pasados son:

- *String numTema*: número de tema al que corresponde la pregunta a contestar. Inicializará el número de tema.
- *String numPreg*: número de pregunta a contestar. Inicializará el número de pregunta.
- *String Preg*: pregunta a la que contestar. Inicializará la pregunta.

- *String respCorr*: respuesta correcta de la pregunta a contestar. Inicializará la respuesta correcta.
- *int numresp*: número de respuestas posibles para la pregunta a contestar. Creará el *array* de respuestas posibles para albergar el número de respuestas posibles.

Métodos Get/Set:

Puesto que en el constructor se pasan los argumentos necesarios para inicializar el objeto solo habrá un método *set* que será para rellenar el *array* de respuestas posibles:

- *public void setResp(int i, String respuesta)*: guardará en la posición *i* del *array* de respuestas posibles la respuesta posible *respuesta*.

Para poder consultar el valor de las variables de este objeto se han creado métodos *get* para cada variable:

- *public String getNumTema()*
- *public String getNumPreg()*
- *public String getPreg()*
- *public String getRespCorrec()*
- *public String[] getRespuestas()*
- *public String getResp(int i)*

Métodos públicos:

Además de los métodos *get* de consulta de variables y *set* de inicialización, se ha diseñado un método para obtener la cadena con la información de la pregunta a enviar al usuario. El usuario solo necesita conocer la pregunta y las posibles respuestas, por lo que este método devuelve el *String* formado para enviar al usuario. Este método es *public String getStringInfoPreg()*.

3.4.3. Clase infoRecibida

Clase implementada para almacenar la información extraída de la cadena de respuesta a la pregunta enviada por el usuario y que permite tener la información organizada.

Variables:

Las variables que contiene esta clase son privadas y de tipo *String*, y se inicializarán en el constructor.

- *MAC*: dirección MAC del teléfono que envía la respuesta.
- *name_alumno*: nombre del alumno que ha respondido a la respuesta.
- *alumno_apell*: apellidos del alumno que ha contestado a la pregunta.
- *dni_alumno*: DNI o NIF del alumno que responde a la pregunta.
- *num_pregunta*: número de pregunta a la que contesta el alumno.

- *respuesta*: respuesta dada por parte del alumno a la pregunta formulada.

Constructor:

El constructor que crea el objeto e inicializa las variables necesarias es *public infoRecibida(String MAC, String nombre_alum, String apellidos, String dni, String num_preg, String resp)*. Cada uno de estos argumentos inicializará la variable correspondiente del objeto creado.

- *MAC*: dirección MAC del dispositivo móvil que ha enviado la respuesta a la pregunta.
- *nombre-alum*: nombre del alumno que contesta a la pregunta formulada.
- *apellidos*: apellidos del alumno que contesta a la pregunta hecha.
- *dni*: DNI o NIF del alumno que contesta a la pregunta.
- *num_preg*: número de pregunta a la que contesta el alumno.
- *resp*: respuesta que ha elegido el alumno a la pregunta expuesta.

Métodos Get:

Se han implementado métodos *get* para poder acceder al valor de las variables que contiene el objeto de la clase que nos ocupa.

- *public String getMAC()*
- *public String getNameAlumno()*
- *public String getApellAlumno()*
- *public String getDniAlumno()*
- *public String getNumPregunta()*
- *public String getRespuesta()*

3.4.4. Clase Form1

Esta es la clase principal de la aplicación, el hilo principal de ejecución. Se hereda de la clase *Form*, clase para crear ventanas estándar o cuadros de diálogo que constituye la interfaz de usuario de la aplicación.

Variables:

Se han declarado en esta clase una serie de variables para controlar la ejecución de la aplicación. Dado el número de variables, se explicarán por conjuntos, es decir, por el motivo por el cual se declararon dichas variables.

1. Variables declaradas para dar nombre a tablas y campos de tablas de la base de datos. Serán public (públicas) para que puedan ser accesible desde otras clases, y *const* (constantes) para que su valor no pueda ser modificado.
    a. *String tabla_preg*: variable de tipo *String* que da nombre a las tablas de preguntas. Está inicializada a "*tema*", de forma que las tablas de preguntas serán de la forma "tema1", "tema2", etc.

b. *String tabla_resp*: variable de tipo *String* que da nombre a las tablas de respuestas a las preguntas. Habrá una tabla por cada tema. Está inicializada a "*respuestas_tema*", ya que las tablas serán de la forma "respuestas_tema1", "respuestas_tema2", etc.
c. *String tabla_alumn*: variable de tipo *String* que da nombre a la tabla de información de los alumnos. Está inicializada a "*infoalumnos*".
d. *String campo_id*: variable de tipo *String* que da nombre al campo de identificador de usuario en la tabla de respuestas. Se ha inicializado a "*identificador*".
e. *String campo_RespAlm*: variable de tipo *String* que da nombre al campo de pregunta a la que se va a contestar. Está inicializada al valor "*Pregunta*", pues será de la forma "Pregunta0", "Pregunta1", etc.

2. Variables declaradas para controlar el funcionamiento de la aplicación; serán de tipo *boolean*.
   a. *Public static Boolean run*: variable pública para que esté accesible desde otras clases, y estática para que esté accesible desde métodos estáticos. Se utiliza para indicar si el servidor se encuentra en ejecución, es decir, si está esperando clientes. Será inicializada en el constructor.
   b. *Public static Boolean conecToBD*: variable pública y estática que indica si se está conectado a la base de datos. Está inicializada a *false*.
   c. *Private Boolean salir*: variable privada para indicar a los procesos que se quiere abandonar y cerrar la aplicación. Será inicializada en el constructor.
   d. *Private Boolean preguntaCambiada*: variable privada para señalizar si se ha cambiado la pregunta mostrada en pantalla. Se ha inicializado a *false*.
   e. *Private Boolean puedeAbortar*: variable privada para indicar que se quiere salir de la aplicación al proceso de búsqueda de dispositivos *Bluetooth*. Está inicializada a *false*.
   f. *Private Boolean pausarBusc*: variable privada para indicar desde la aplicación principal al proceso de búsqueda de dispositivos *Bluetooth* que se quiere pausar esa búsqueda. Inicializada a *false*.
   g. *Private Boolean puedePausarBusc*: variable privada para indicar desde el proceso de búsqueda de dispositivos a la aplicación principal que se puede pausar ese proceso. Se utiliza para evitar pausar el proceso mientras se están procesando dispositivos *Bluetooth* encontrados. Inicialmente se inicializa a *false*.
   h. *Private Boolean pausarProcResp*: variable privada para indicar al proceso de procesamiento de respuestas que se quiere pausar. Se indica desde la aplicación principal. Se ha inicializado a *false*.
   i. *Private Boolean puedePausarProcResp*: variable privada para indicar desde la aplicación principal al proceso de procesamiento de respuestas que se puede pausar, puesto que se habrán procesado las respuestas que se tenían. Inicializada a *false*.

3. Variables constantes para utilizar en aplicación.

   a. *int MAX*: variable entera empleada para dar tamaño a *buffers*. Inicializada a *1024*.
   b. *int MIN_DISPOSITIVOS*: inicializada a *8*, se utiliza para indicar el número mínimo de dispositivos que se tienen que encontrar en el proceso de búsqueda de dispositivos *Bluetooth* para no dormir el proceso durante un tiempo (prevenir estar haciendo continuas búsquedas).
   c. *float tamRespuestasPantalla*: variable *float* para establecer el tamaño de fuente de las respuestas que se muestran por pantalla en la aplicación principal. Está inicializada a *20*.
4. Variables enteras utilizadas para detectar el tipo de error devuelto por la base de datos *MySQL*. Serán *public* (públicas) para que puedan verlas otras clases, y *const* (constantes) para que su valor no pueda ser cambiado.
   a. *DUPLICATE_COLUMN*: columna ya existente. Código de error *1060*.
   b. *TABLE_NOT_EXIST*: la tabla no existe. Código de error *1146*.
   c. *TABLE_EXIST*: la tabla ya existe. Código de error *1050*.
   d. *UNKNOWN_COLUMN*: columna no encontrada. Código de error *1054*.
5. Variables empleadas para identificadores de servicio de *Bluetooth*, para identificar entre el servicio de envío de preguntas, y el servicio de recolección de respuestas. Son de tipo *Guid*, y privadas.
   a. *uuidResp*: identificador de servicio *Bluetooth* mediante el cual el dispositivo móvil indica que quiere enviar la respuesta a una pregunta. Se inicializa a "*12345678-1234-1234-1234-123456789012*".
   b. *uuidPreg*: identificador de servicio *Bluetooth* mediante el que el dispositivo móvil indica que está esperando una pregunta. Inicializado a "*12345678-1234-1234-1234-123456789000*".
6. Variables empleadas para guardar las respuestas recibidas por parte de los dispositivos móviles, los que han contestado y los que han solicitado una pregunta. Serán privadas, y *arrays* de tipo *List*, que es un tipo de *array* dinámico.
   a. *List<byte[]> respuestas*: *array* que albergará datos de tipo *byte[]* (*array* de *bytes*), que serán las respuestas recibidas de los dispositivos móviles. Se guardan en binario, tal y como se reciben a través de *Bluetooth*.
   b. *List<String> arrayPreguntados*: *array* de variables *String* que contendrá la dirección MAC de los dispositivos que hayan solicitado una pregunta. Se utilizará para contabilizar los clientes que han solicitado preguntas, de forma que se pueda detectar si han contestado todos los clientes y aportar información en la ventana principal.
   c. *List<String> arrayContestados*: *array* de variables *String* que contendrá la dirección MAC de los dispositivos que hayan contestado a la respuesta actual. Se empleará para contabilizar los clientes que han respondido a la pregunta actual, con la misma misión que el *array* anterior.

7. Variables enteras empleadas para indicar los retardos empleados en los procesos, retardos empleados para evitar que, en caso de que no haya clientes (o haya pocos), se esté continuamente buscando y procesando sin pausa, evitando así un uso excesivo de recursos por parte de la aplicación. Estas variables serán privadas (*private*).
    a. *retardoResp*: inicializada a *4*, se empleará para pausar el proceso que se encarga de procesar las respuestas recogidas y obtener los datos brutos. El valor inicializado representa que se pausará durante el tiempo especificado (en segundos).
    b. *retardoBusqDisp*: se utilizará para pausar el proceso de búsqueda de dispositivos *Bluetooth* (clientes). Se ha inicializado a *4*, lo cual indica que si se han encontrado un número mínimo de dispositivos (recordemos que se dispone de una variable que indica este valor mínimo llamada *MIN_DISPOSITIVOS*), se pausará este proceso durante el tiempo especificado por esta variable, en segundos.
8. Las siguientes variables son objetos que se utilizan en la ejecución de la aplicación.
    a. *BluetoothClient servidor*: objeto del tipo especificado que representa un cliente *Bluetooth*. Se utiliza para realizar el descubrimiento de dispositivos *Bluetooth* disponibles. Está definido en la librería empleada para manejar *Bluetooth* que se puede encontrar en el capítulo de *Bluetooth* (*2.4.1. Librería Bluetooth para C#*).
    b. *BluetoothRadio blue*: objeto que representa el dispositivo *Bluetooth* local. Se emplea para configurar el adaptador *Bluetooth* local, así como averiguar si es compatible. Está definido en la librería empleada para utilizar *Bluetooth* que se localiza en el capítulo de *Bluetooth* (*2.4.1. Librería Bluetooth para C#*).
    c. *public static MySqlConnection conexión*: objeto que representa un conector a la base de datos de *MySQL*. Está definido en las librerías empleadas para interactuar con bases de datos *MySQL* (*2.2.2. Librería MySQL para C#*). Será *public* (pública) para que pueda ser utilizada desde otras clases, y estática (*static*) para que también pueda ser empleada desde métodos estáticos.
    d. *public static infoBaseDatos infoDB*: objeto de la clase *infoBaseDatos* creada para guardar la información de conexión de la base de datos (*3.4.1. Clase infoBaseDatos*). Será público (*public*) para ser accesible desde otras clases, y estático (*static*) para ser empleado desde métodos estáticos.
    e. *infoPregunta infoPreg*: objeto de la clase *infoPregunta* desarrollada para almacenar la información de la pregunta a contestar (*3.4.2. Clase infoPregunta*).
    f. *Thread thBuscCli*: objeto de la clase *Thread* incluida en la librería *System.Threading* que representa un proceso. Este proceso será empleado para realizar la búsqueda de clientes *Bluetooth*, de forma que

no se bloquee la ejecución de la ventana principal; es decir, es un hilo que se ejecuta en paralelo a la aplicación.

g. *Thread thprocesResp*: objeto de la clase *Thread*. Tiene la misma misión que la variable anterior, es decir, ejecutarse en paralelo a la aplicación principal. Este proceso se encarga de procesar las respuestas recibidas por parte de los dispositivos móviles (clientes).

h. *String pregToSend*: variable de tipo *String* que contendrá la cadena con la pregunta y posibles respuestas que enviará el servidor a los clientes (dispositivos móviles) que la soliciten.

i. *private OpcionesForm OpcF*: objeto de la clase *OpcionesForm* que se ha creado para configurar el servidor, tanto datos de conexión de la base de datos como distintas opciones de visualización de datos ya recogidos en la base de datos, así como incorporación de datos en ésta. Se explicará posteriormente (*3.4.5. Clase OpcionesForm*).

Por otro lado, también se dispone de una serie de variables que se corresponden con componentes gráficos como botones, etiquetas de texto, barras de estado… Puesto que el entorno de desarrollo, tal y como se comentó en el capítulo *2.3. C# y Entorno de desarrollo*, permite crear la parte gráfica de la aplicación mediante un editor gráfico, todo el código referente a estos componentes gráficos se crea de forma automática, y está disponible en el archivo *Form1.Designer.cs*. Así, estas variables o componentes gráficos que forman parte de esta clase (*Form1*) son:

- *StatusStrip statusStrip1*: barra de estado para indicar detalles de la aplicación.
- *Button iniciar_Serv*: botón para seleccionar que se inicie/pause/continue el servidor.
- *Button salir_button*: botón para abandonar la aplicación.
- *NumericUpDown numPregnumericUpDown*: componente numérico que permite modificar el valor numérico de éste mediante pulsaciones con el *mouse* o mediante teclado, y que se utiliza para seleccionar el número de pregunta a cargar.
- *Button cargarPreguntaButton*: botón para cargar por pantalla la pregunta seleccionada del tema seleccionado.
- *Button MostrarRespbutton*: botón para resaltar o mostrar la respuesta correcta de las posibles respuestas mostradas en pantalla.
- *GroupBox RespuestasgroupBox*: grupo de elementos que contendrá las posibles respuestas de la pregunta cargada.
- *Label Preguntalabel*: etiqueta de texto que se cargará con la pregunta seleccionada.
- *Button Opcionesbutton*: botón que abrirá la ventana de opciones de la aplicación.
- *Button aumentarButton*: botón para aumentar el tamaño del texto de la pregunta y posibles respuestas mostradas por pantalla.
- *Button disminuirbutton*: botón para disminuir el tamaño del texto de la pregunta y las posibles respuestas mostradas por pantalla.
- *ToolStripStatusLabel toolStripStatusLabel1*: etiqueta de texto incrustada en la barra de estado y que se empleará para indicar si se encuentra iniciado el servidor o no.

- *ToolStripStatusLabel toolStripStatusLabel3*: etiqueta de texto incrustada en la barra de estado que se empleará para indicar si se encuentra o no conectado a la base de datos.
- *System.Windows.Forms.Timer timer*: objeto *timer* (temporizador) para si se desea activar la temporización para contestar a la pregunta formulada.
- *DateTimePicker temporizador*: componente gráfico que muestra el tiempo en formato *HH:MM:SS* utilizado para especificar el tiempo de que se dispone para contestar a la pregunta.
- *NumericUpDown numTemanumericUpDown1*: componente numérico para seleccionar el número de tema sobre el que seleccionar la pregunta.
- *Label label1*: etiqueta de texto situado junto al componente numérico de selección de número de pregunta para indicar al usuario que es de selección de pregunta (estará inicializado con la cadena "*Pregunta:*").
- *Label label2*: etiqueta de texto situado junto al componente numérico de selección de tema sobre el que se selecciona la pregunta para indicar al usuario que es de selección de tema (estará inicializado con la cadena "*Tema:*").
- *ToolStripStatusLabel infoContesttoolStripStatusLabel2*: etiqueta de texto incrustada en la barra de estado, y que se utiliza para mostrar información sobre el número de preguntas servidas y el número de respuestas recogidas.

Constructor

En el constructor de la presente clase se lleva a cabo la inicialización de variables y objetos.

En primer lugar se lleva a cabo la inicialización de todos los componentes gráficos que componen la ventana de interfaz de usuario (botones, cuadros de texto, …) mediante la función *InitializeComponent()*, función que hereda de la clase *Form*, que como se ha mencionado anteriormente, es la clase que represente una ventana o cuadro de diálogo. Esta función la implementa de forma automática el entorno de desarrollo, que como se comentó en el capítulo correspondiente (*2.3. C# y Entorno de desarrollo*) abstrae al desarrollador de implementar todo el código correspondiente con esos componentes gráficos. A continuación, puesto que se emplean variables compartidas, para evitar que la aplicación nos muestre errores al utilizar estas variables compartidas en más de un método, se pone la propiedad *CheckForIllegalCrossThreadCalls* a *false*. También se inicializa el valor de la barra de estado (indicando que el servidor está parado, y que no se encuentra conectado a la base de datos), los *arrays* (*List*) de *respuestas*, *arrayContestados* y *arrayPreguntados*, así como los objetos *infoDB* y *conexión*. Además, se indica que la aplicación no se encuentra en ejecución (*run = false*) e inicializamos la bandera de abandonar la aplicación a *false* (*salir = false*).

Métodos Implementados:

Tanto para atender a los componentes gráficos que lancen eventos cuando interactúen con el usuario como para lograr la funcionalidad deseada por la aplicación, se han desarrollado una serie de métodos. En primer lugar se explicarán de forma detallada aquellos más importantes puesto que se corresponden con los más comprometidos, y nos ayudarán a la comprensión de otros métodos. En segundo lugar se detallarán los métodos creados para realizar un código más flexible, y por último los métodos de atención a eventos lanzados por los componentes gráficos.

Método del hilo de búsqueda de clientes (public void buscaClientes()):

Como se ha mencionado con anterioridad, será necesario crear hilos de ejecución en paralelo a la ventana principal, puesto que de otra forma la ventana quedaría congelada y no respondería a las instrucciones del usuario. Uno de los hilos a crear es el de búsqueda de clientes, pues mientras se busca y atiende a los clientes que lo soliciten, el usuario tiene que tener total control sobre la aplicación por si en cualquier momento desea cancelar la búsqueda, trabajar en la ventana de opciones, o simplemente aumentar o disminuir el tamaño de la pregunta y las respuestas mostradas por pantalla.

Con anterioridad, en el capítulo *3.2. Problemas* se explicaba el problema surgido con la comunicación *Bluetooth* con los dispositivos móviles. La solución adoptada para salvar este problema es hacer que sea el servidor quien se conecte a los dispositivos móviles, de forma que el dispositivo móvil será quien espere a que se conecten a él. Por tanto, para evitar el uso excesivo de recursos y realizar este proceso de manera óptima se plantea haciendo uso de la visibilidad e identificadores de servicio que ofrece *Bluetooth*.

De esta forma, y sin entrar en este punto en detalles de la aplicación del dispositivo móvil (puesto que se abordará en el capítulo correspondiente), el adaptador *bluetooth* del dispositivo móvil solo se pondrá en modo *visible* cuando esté preparado para recibir la pregunta o enviar la respuesta dada por el usuario. Por tanto, no se descubrirán todos los dispositivos cuando no sea necesario. Puesto que el servidor puede detectar otros dispositivos que dispongan de *Bluetooth*, habrá que diferenciar entre los dispositivos partícipes del sistema, y los que no lo sean. Además, entre los dispositivos móviles que estén formando parte del sistema, habrá que diferenciar entre los que quieran recibir la pregunta a contestar, y los que quieran enviar la respuesta que el usuario ha dado a la pregunta. Para hacer esta diferenciación se utilizan los identificadores de servicio o *GUID*, estableciendo los que se quieran y sin que interfieran con los estándares. En este caso son las variables antes presentadas *uuidPreg* y *uuidResp*.

Conociendo estos detalles, el funcionamiento de este hilo es un bucle infinito que seguirá siempre la misma secuencia. Y es en bucle infinito puesto que, al tratarse de un hilo o *Thread*, se podrá pausar, continuar o interrumpir cuando se desee desde el hilo principal. La secuencia seguida en el hilo es, en primer lugar, realizar la búsqueda de dispositivos mediante el método *DiscoverDevices()* del objeto creado al iniciar el servidor, y que se aloja en la variable *servidor* (es de tipo *BluetoothClient*). El resultado devuelto, que será un *array* con los dispositivos encontrados en la búsqueda se guarda en *dispositivos* (de tipo *BluetoothDeviceInfo[]*).

Una vez se tienen los dispositivos *Bluetooth* activos, se procede a comprobar si es forma parte de la aplicación, y si quieren recibir la pregunta o enviar la respuesta. Para ello se comprueba, en primer lugar, si dispone del identificador de servicio designado para las preguntas. Esto se hace mediante la función *GetServiceRecords(uuidResp)* pasando como argumento el identificador de servicio buscado. Si se encuentra es que es un dispositivo que espera la pregunta a contestar, por lo que se crea un *BluetoothClient*, se configura la conexión, y se envía la pregunta que se encuentra contenida en la variable *pregToSend*. Una vez enviada, se espera asentimiento, y puesto se comprueba si a este dispositivo ya se le ha enviado la pregunta (para llevar una contabilización),

añadiéndose a este array (*arrayPreguntados*) en caso de no encontrarse mediante la función *Add()* con la MAC del dispositivo móvil al que se ha servidor, y actualizando el valor de la barra de estado correspondiente. Por último, se cierra la conexión *Bluetooth* con el dispositivo móvil, y se pasa al siguiente dispositivo encontrado puesto que un mismo dispositivo no puede esperar la pregunta y enviar la respuesta a la misma vez. Esto se hace mediante la palabra reservada *continue*.

En caso de no encontrarse el identificador de servicio de pregunta se comprueba si dispone del de la respuesta de la misma forma que se ha procedido en el caso anterior. En caso afirmativo, se sigue la misma secuencia solo que en este caso primero espera recibir la respuesta por parte del dispositivo móvil y después envía el asentimiento. La respuesta recibida por parte del dispositivo móvil no se procesa en este hilo, solo se guarda en el *array* de tipo *List* llamado *respuestas* en binario, en el mismo formato que lo recibe el servidor, para evitar pérdidas de tiempo de procesado de información. Será otro hilo en que se encargue de obtener la información.

Si el dispositivo *bluetooth* encontrado no ha entrado en ninguno de los dos casos es porque no forma parte del sistema. Cuando termina de procesar los dispositivos encontrados, se comprueba si se ha solicitado por parte del hilo de ejecución principal que se pause (*pausarBusc = true*), en cuyo caso se le notifica que puede pausarlo (*puedePausarBusc = true*). Cuando se decida reanudar el hilo, además de hacerlo con la función correspondiente, también se pondrá *pausarBusc* a *false*. También comprueba si se ha solicitado salir (*salir = true*), en cuyo caso también lo señala (*puedeAbortar = true*). Esto se hace de esta forma para evitar pausar o para el hilo sin que se hayan terminado de procesar todos los dispositivos encontrados. Para evitar muchas búsquedas en muy poco tiempo porque hay pocos dispositivos o ninguno, se comprueba si el número de dispositivos encontrados es menor o igual que el número mínimo de dispositivos indicado por la variable *MIN_DISPOSITIVOS*. En caso de que sea menor, se duerme este hilo durante el periodo de tiempo indicado por la variable *retardoBuscDisp*.

A continuación, y para mayor claridad, se adjunta el diagrama de flujo resumen (*Figura 5*).

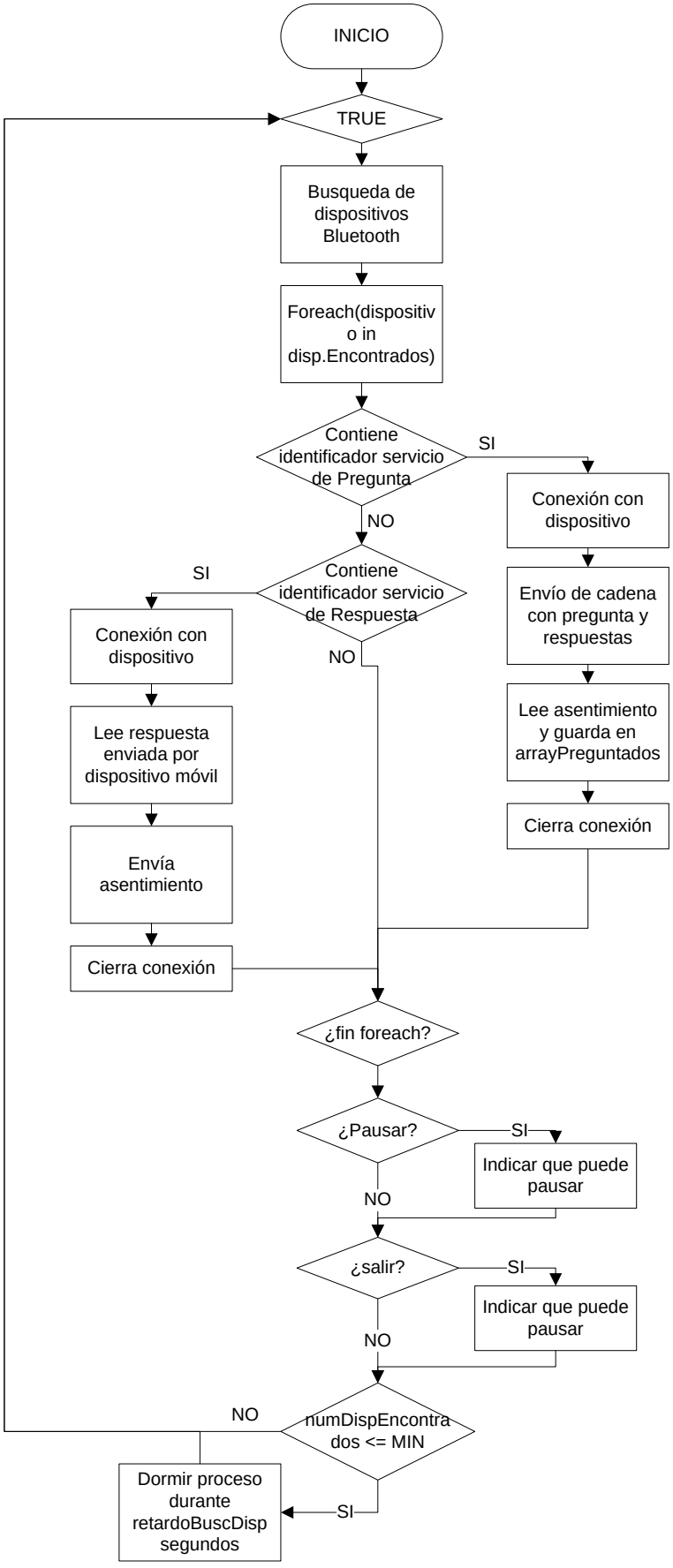

Figura 5: diagrama flujo hilo búsqueda clientes

Método del hilo procesar respuestas (public void processResp()):

El segundo y último hilo a crear es el que se encarga de procesar las respuestas recogidas en el proceso de búsqueda de clientes, pues no se hace en ese hilo para emplear el menor tiempo posible en procesar a los clientes. Por lo tanto, el presente hilo extraerá la información de la cadena de respuesta enviada por el dispositivo móvil, y la guardará como corresponda.

En este hilo, en primer lugar se reserva la variable *codif* de tipo *UTF8Encoding* para realizar conversiones de *bytes* a *String*, u otras representaciones, y un *Boolean* para detectar si los datos se han insertado satisfactoriamente en la base de datos. Una vez reservadas, se entra en un bucle infinito. En primer lugar se comprueba si el número de respuestas a procesar es nulo, en cuyo caso, si no se ha solicitado pausar el hilo o salir, se duerme el hilo durante el tiempo definido por la constante *retardoResp*; si se ha solicitado pausar el hilo desde el hilo principal de ejecución, se indica que ya puede pausarlo mediante *puedePausarProcResp* (poniéndola a *true*) y, aunque la reanudación la ejecuta otra función, también se espera a que se indique con *retardoResp = false* para que, mientras se pausa o no, siga ejecutándose. Esto se hace para evitar pausar el hilo mientras queden respuestas por procesar. En caso de que el hilo principal haya solicitado que la aplicación se va a cerrar, también se le notifica que está preparado con *puedePausarProcResp*.

Si quedaban respuestas por procesar, se crea el objeto *inforx* de la clase *infoRecibida*, y se le asigna el resultado del método *procesaXml(byte[] respuesta)* que se explicará con posterioridad. Esta función devuelve un objeto de la clase *infoRecibida* con los datos de la respuesta del dispositivo móvil procesados. Si todo ha ido bien, se inserta la información en la base de datos mediante el método *insertarenDB(infoRecibida inforx)* que se encargará de guardar los datos correspondientes en la base de datos, como se verá después. Si los datos se han insertado con éxito, se elimina esta entrada de la lista de respuestas (variable de tipo *List* llamada *respuestas*), y se actualiza la lista de dispositivos móviles que han contestado (*arrayContestados*) si es necesario (actualizando el valor de la barra de estado correspondiente que indique los clientes que han contestado). Si ha ocurrido algún error en la inserción de datos en la base de datos, se notifica al usuario, se intenta parar el servidor haciendo que salte el evento de pulsación del botón para iniciar/pausar/continuar la ejecución del servidor. Además, si se ha producido este error de inserción de datos en la base de datos, se indica que se puede pausar el proceso.

El diagrama de flujo que resume el funcionamiento de este hilo es el siguiente (*Figura 6*)

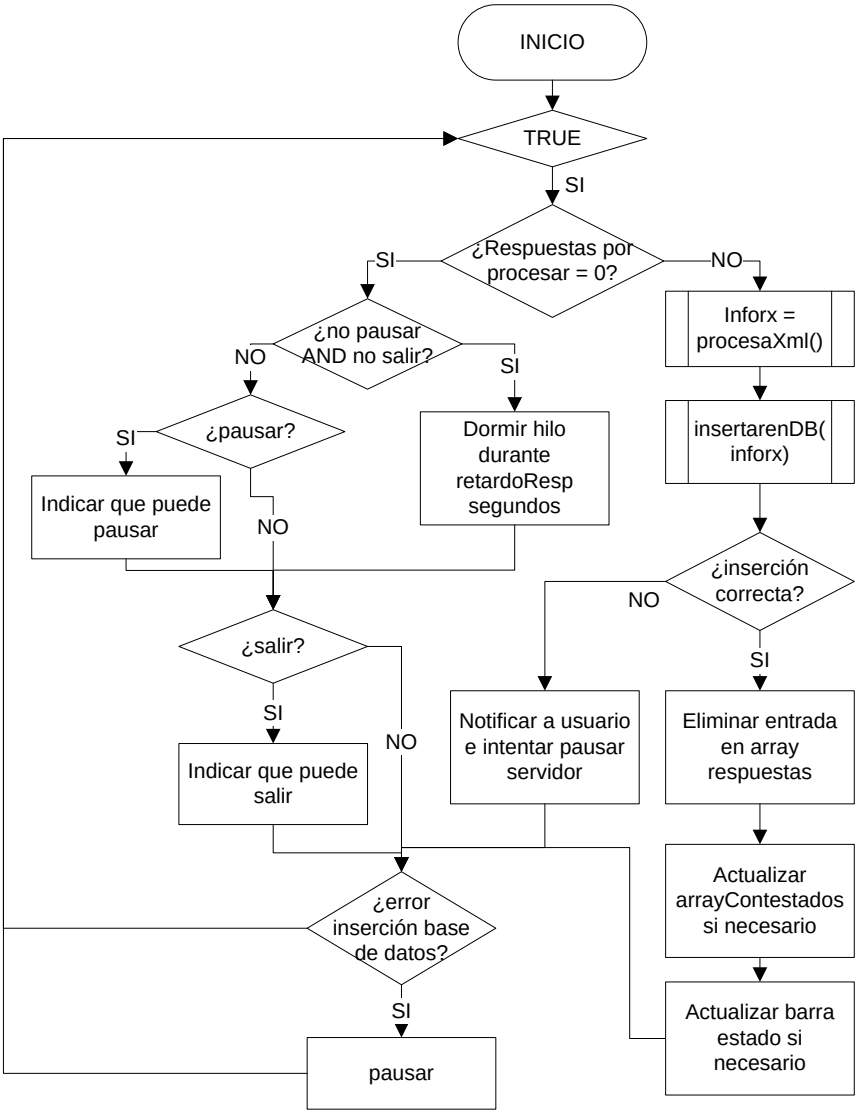

Figura 6: diagrama de flujo de hilo de procesar respuestas

Métodos:

Se han implementado métodos, como los que se explicarán a continuación, para hacer más flexible el código y emplearlas a modo de caja negra, pues si después cambia el tipo de procesamiento únicamente habrá que modificar el método. De esta forma se dispone de los siguientes métodos:

- *public infoRecibida procesaXml(String contenido)*:

  Este método procesa la cadena de respuesta enviada por el dispositivo móvil. Esta cadena viene en lenguaje de marcas XML, pues de esta forma se tienen bien identificados los datos, y *C#* es capaz de trabajar con estos archivos de manera muy eficiente. Los datos extraídos se guardan en un objeto de la clase *infoRecibida*, que es lo que devuelve el método.
  La cadena con la respuesta enviada por el usuario se tiene el siguiente formato:

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<datos>
```

```
  <info_alumno>
        <addr></addr>
        <nombre_alumn></nombre_alumn>
        <apellido_alumn></apellido_alumn>
        <dni_alumno></dni_alumno>
  </info_alumno>
  <info_resp>
        <pregunta></pregunta>
        <respuesta></respuesta>
  </info_resp>
</datos>
```

En primer lugar se crea un objeto del tipo *XmlDocument* llamado *xmlDoc*, el cual representa un documento xml. También se crea un puntero a objeto *infoRecibida*, *inforx*. A continuación se carga la cadena con la respuesta recibida del dispositivo móvil en el documento XML, y si no se produce ningún error es que está bien formado (lo cual indica que viene completo). Una vez cargado, se crea un objeto *XmlNodeList* y otro *XmlNode* para extraer la información, pues en el primero se guardarán todos los nodos que cuelguen del especificado; y en el segundo el nodo a procesar.
Una vez sacados los valores, se crea un objeto de la clase *infoRecibida* pasando en el constructor los datos obtenidos y se asigna a *inforx*, que será el objeto que devuelva el método.

- *public bool insertarenDB(infoRecibida inforx)*:

  Método para insertar en la base de datos la información de la respuesta procesada.
  En primer lugar se construye la consulta a ejecutar en la base de datos, que será la de buscar el identificador del alumno que ha enviado esa pregunta (tanto la información del alumno, como la MAC del dispositivo móvil desde el que se envía y la información de la pregunta están en el objeto *inforx*). Si el alumno se encuentra en la base de datos, se devolverá su identificador; en caso negativo, hay que insertar al alumno en la tabla de alumnos dentro de la base de datos. Para ello se construye una nueva consulta de inserción de datos en la base de datos y se ejecuta. Si se ha ejecutado con éxito, se vuelve a consultar el identificador del alumno insertado, puesto que este valor lo asigna la tabla al insertar al alumno.
  Una vez tenemos el identificador de alumno, se crea la consulta para insertar la respuesta en la tabla correspondiente. Si el usuario ya había contestado a otras preguntas de ese tema, la consulta daría un error, por lo que se hace dentro de un bucle para que, si salta el error, crear una consulta de actualización de fila, introduciendo el nuevo valor contestado.
  Si todo el proceso ha ido bien, se devolverá el valor *true*. En caso negativo, allí donde se haya producido el error se le habrá indicado al usuario, además de devolver el valor *false*.

- *private void mostrarPregunta()*:

  Método empleado para mostrar por pantalla la pregunta seleccionada y sus posibles respuestas. Todos estos datos están dentro del objeto *infoPreg*.
  En primer lugar se limpian las respuestas que ya hubiese cargadas dentro del *GroupBox RespuestasgroupBox*, llamando al método *Clear()* dentro de la

propiedad *Controls* (*RespuestasgroupBox.Controls.Clear()*). Puede haber dos tipos de preguntas: se hace una pregunta y hay que seleccionar la respuesta entre las posibles que ofrece; la segunda opción es que haya que introducir manualmente la respuesta, es decir, que se pida completar una frase, por ejemplo. Para saber de qué tipo se trata, se declara la variable *k* de tipo *int*, inicializada a 0, y que se incrementará en caso de que la respuesta tenga texto. Se carga la pregunta en el *Label* correspondiente (*Preguntalabel*), y se hace visible. Las respuestas se escribirán en controles *RadioButton*. Por tanto, se va recorriendo cada respuesta posible, y si contiene texto, se crea el objeto *RadioButton*, se configura la fuente y tamaño de letra, se le asigna como texto la respuesta posible, y se añade al *GroupBox*.
Procesadas todas las respuestas posibles, se comprueba si las respuestas posibles no contenían texto, en cuyo caso se trata de una pregunta en la que hay que introducir por teclado la respuesta. En ese caso, se crea un objeto *RadioButton*, y al igual que antes, se le configura la fuente y tamaño de letra, y se le asigna al *GroupBox*.

- *public static Boolean conectarConBD()*:

  Método estático para conectar con la base de datos. Devuelve un valor *Boolean* que indica si se ha llevado a cabo la conexión con la base de datos o no.
  En primer lugar, se elimina el conector *conexion* declarado en la clase *Form1*. A continuación se crea la cadena de conexión, y se intenta conectar; en caso favorable se pone la variable global *conecToBD* a *true*, se suscribe al evento de cambios de estado en la base de datos, y se devuelve *true*. En caso de que suceda algún problema en la conexión, tratamos la excepción, indicando al usuario el motivo del error, y devolviendo *false*.

- *public Boolean addColumna()*:

  Método para añadir una nueva columna a la tabla de respuestas de la base de datos. Devuelve un valor *Boolean* que indica si se ha añadido la columna con éxito o no.
  Se comienza construyendo la consulta de añadir columna la tabla de respuestas, y se ejecuta la consulta. Si ocurre algún error se trata, notificando al usuario en caso de que esa columna ya exista, pues eso significará que se ha contestado con anterioridad y se pueden machacar datos; se le da a elegir entre si quiere continuar y machacar datos, o por el contrario no.
  Si se ejecuta con éxito, o si el usuario decide continuar, se liberan recursos y se indica que la columna se ha añadido con éxito devolviendo *true*.

Métodos de atención a eventos:

Al tratarse de una aplicación de escritorio con interfaz de usuario donde éste interactúa con componentes gráficos, éstos lanzan eventos, y es en estos eventos donde se incrusta el código que se quiere que se ejecute, pues al suscribirse al evento se le indica el método que se quiere ejecutar cuando se lance. A continuación se explica el

código realizado para cada uno de los botones que componen la interfaz gráfica de usuario, así como otros componentes que generan eventos.

- *private void timer_Tick(object sender, EventArgs e)*:

  Cuando está iniciado el temporizador para contestar una pregunta, cada vez que transcurre un segundo salta este evento, pues está así configurado para decrementar el *DateTimePicker*, además de detectar cuándo el contador ha llegado a cero.
  En primer lugar se resta un segundo al tiempo restante en *temporizador* (variable de tipo *DateTimePicker*). En caso de que haya alcanzado el valor *0*, lo cual quiere decir que se ha cumplido el tiempo establecido para contestar la pregunta, se para la temporización y se comprueba si han contestado todos (comparando el tamaño de *arrayContestados* y *arrayPreguntados* donde se han ido guardando los que han solicitado pregunta, y los que han respondido). En caso de que aún no hayan contestado todos, se le notifica al usuario, cuestionándole si se debe parar el servidor o no. Si responde "*no*", se sale de este método; en caso afirmativo o que hayan contestado todos los que han solicitado pregunta, hace saltar el evento de pulsación de botón de iniciar/pausar/continuar servidor para pausar el servidor, y se le notifica al usuario que el tiempo ha expirado. Además se le da la opción de mostrar la respuesta correcta, procediendo a mostrarla en caso afirmtivo.

- *private void iniciar_Serv_Click(object sender, EventArgs e)*:

  Evento que se produce cuando el usuario pulsa sobre el botón *iniciar_Serv* con texto "Iniciar Servidor", "Continuar" o "Pausar", dependiendo del texto que muestre el botón por el estado de ejecución de la aplicación. Para evitar tener tres botones en los que, cuando uno activa una función no se puede pulsar de nuevo el mismo botón, se simplifica utilizando un solo botón, y haciendo que cambie el texto que muestra dependiendo del estado en el que se encuentre.
  Este botón controla la ejecución del servidor, es decir, la búsqueda de dispositivos y procesamiento de respuestas. Si el texto mostrado es "Iniciar Servidor" significa que aún no se ha ejecutado ninguna vez, por lo que se deberán de crear e inicializar los objetos oportunos; en caso de que el servidor se encuentre en ejecución mostrará el texto "Pausar", que pausará al servidor; si se encuentra pausado, mostrará el texto "Continuar", a lo que responderá arrancando de nuevo el servidor.
  Al pulsar sobre este botón *iniciar_Serv*, en primer lugar se comprueba la opción a ejecutar a través de la comparación del texto que muestra el botón:

  - "*Iniciar Servidor*": se comprueba si se ha conectado a la base de datos, y si se ha cargado la pregunta por pantalla. En caso negativo en cualquiera de los dos casos se notifica al usuario y se sale del método. A continuación se comprueba si se encuentra conectado algún adaptador *Bluetooth* o si es compatible mediante la propiedad *BluetoothRadio.IsSupported* (en caso negativo se notifica al usuario y se sale de este método). Se comprueba si se ha cambiado la

pregunta mostrada por pantalla mediante el flag *preguntaCambiada*, en cuyo caso se procede a añadir la nueva columna en la tabla de respuestas de la base de datos. Si se ha añadido con éxito, se configura el adaptador *bluetooth* local en modo descubrible, y en la barra de título de la ventana principal aparece el nombre y la MAC del adaptador *bluetooth*.
Se configura el objeto *servidor* de tipo *BluetoothClient*, que será el que realizará búsquedas de dispositivos en el hilo correspondiente, y se crean los hilos de ejecución paralelos de búsqueda de dispositivos (*thBuscCli*) y (*thprocesResp*), y se inician mediante el método *Start()*. También se cambian los valores de la barra de estado correspondiente (indica que el servidor está iniciado, las preguntas que se han servido y las respuestas recogidas), si señala que la aplicación está corriendo (*run = true*), se cambia el nombre del botón que nos ocupa a "Pausar", y se comprueba si se ha especificado temporización, en cuyo caso también se activa la temporización con el *Timer timer* (llamando al método *Start()*).

- "Continuar": se comprueba si se hay alguna pregunta cargada en pantalla. En caso positivo se ve si se ha cambiado la pregunta, indicado por el flag *preguntaCambiada* (en caso positivo se procede a añadir una nueva columna la tabla de respuestas correspondiente mediante el método *addColumna()*, y se limpian arrays (*arrayContestados* y *arrayPreguntados*, mediante el método *Clear()*). A continuación, si no se está conectado a la base de datos, se intenta reconectar. En caso de éxito, se configura el adaptador *Bluetooth* como descubrible, se ponen los flags de pausar procesos (*pausarBusc* y *pausarProcResp*) a *false*, se continúa con la ejecución de los hilos (llamando a la función *Resume()*), se indica que la aplicación está en ejecución (*run = true*), se cambia el nombre del botón actual a "Pausar", se actualizan valores de la barra de estado, y se comprueba de nuevo la temporización para actuar en consecuencia.

- "Pausar": En primer lugar se comprueba si han contestado todos los que han solicitado una pregunta, mediante la comparación del tamaño de los *arrays arrayContestados* y *arrayPreguntados*. En caso negativo, se informa al usuario y se le pide que decida entre continuar pausando o no. Si han contestado todos, o si se decide continuar pausando el servidor, se indica a los hilos mediante los flags *pausarBusc* y *pausarProcResp* que se quieren pausar, para que de esta forma terminen de procesar lo que les quede y se preparen para ser pausados, y se crea una ventana para indicar al usuario que se está terminando de procesar clientes. Cuando cada hilo notifica que está preparado para ser pausado (mediante *puedePausarBusc* y *puedePausarProcResp*), se pausan los hilos con el método *Suspend()*, se cierra la ventana de información y se apaga el adaptador *bluetooth*. Además, se indica que la aplicación no está corriendo (*run = false*), se cierra la conexión con la base de datos, se para la temporización, se cambia el nombre del botón a "Continuar"

y se cambia el texto de la barra de estado para notificar el estado del servidor.

- *private void salir_button_Click(object sender, EventArgs e)*:

  Evento que se produce cuando el usuario pulsa sobre el botón *salir_button*, que indica que el usuario quiere abandonar la aplicación, en cuyo caso hay que liberar recursos y salir.
  Para prevenir que el usuario haya pulsado sin quererlo, se le pregunta si quiere abandonar la aplicación realmente. Si es así, se crea la ventana para indicar al usuario que se está terminando de procesar clientes, se pone el flag *salir* a *true* para que los hilos se preparen para abandonar la aplicación (evitar dejar procesamiento de clientes a la mitad). Para salir, se sigue el mismo proceso que el método anterior cuando se pulsa sobre "Pausar", es decir, se le notifica a los hilos que se quiere abandonar la aplicación para que terminen de procesar lo que tienen y no sigan, y cuando los hilos están preparados se lo notifican al hilo principal de ejecución, que procede a la interrupción de éstos. En este caso, como se abandona la aplicación, una vez los hilos han mostrado su preparación, se llama al método *Abort()* que "mata" a los hilos.
  Por último, cierra la ventana de información al usuario, cierra la conexión con la base de datos si está activa, libera recursos y cierra la ventana.

- *private void cargarPreguntaButton_Click(object sender, EventArgs e)*:

  Evento que se produce cuando el usuario pulsa sobre el botón *cargarPreguntaButton*. Esto indica que cargue en pantalla la pregunta designada por el número de tema, y número de pregunta dentro de ese tema, mediante los componentes *numTemanumericUpDown* y *numPregnumericUpDown*.
  Comienza comprobando si el servidor se encuentra iniciado, en cuyo caso se solicita que sea detenido. En caso negativo, se comprueba si, de la pregunta que ya había cargada, han contestado todos, avisando al usuario de no ser así. A continuación, si no se está conectado a la base de datos, se intenta conectar. Si se ha realizado la conexión con éxito, se procede a cargar la pregunta en pantalla. El primer paso es obtener la pregunta de la base de datos, por lo que se construye la consulta correspondiente, y se ejecuta.
  Si se obtiene la información de la pregunta, se averigua el número de respuestas que tiene dicha pregunta, y se inicializa el objeto *infoPreg* con los datos de la pregunta mediante el constructor. Para inicializar las posibles respuestas de la pregunta, se llama al método set en un bucle hasta haber almacenado todas las respuestas. Guardadas las respuestas, se llama al método *mostrarPregunta()* que mostrará por pantalla la pregunta deseada, se obtiene la cadena a enviar a los clientes que soliciten pregunta en la variable *pregToSeng*, se inicializan los *arrays arrayContestados* y *arrayPreguntados* mediante *Clear()* y se indica que se ha cambiado de pregunta (*preguntaCambiada = true*). Para terminar, se liberan recursos.
  Si se produce algún error o excepción, se trata informando al usuario del error sucedido.

- *private void MostrarRespbutton_Click(object sender, EventArgs e)*:

  Evento que se produce al pulsar sobre el botón *MostrarRespbutton*, que el usuario pulsará cuando quiere visualizar por pantalla la respuesta correcta.
  Comienza asegurándose de que está conectado a la base de datos y que hay una pregunta cargada en pantalla. A continuación comprueba si han contestado todos los usuarios que han solicitado una pregunta, informando al usuario en caso contrario. En caso de que el servidor se encuentre en ejecución (el nombre del botón *iniciar_Serv* debe ser "Pausar"), se genera el evento para que se pause llamando al método de tratamiento del evento (*iniciar_Serv_Click(null, null)*).
  Obtiene la respuesta correcta del objeto *infoPreg*, que contiene toda la información de la pregunta cargada en pantalla, mediante el método *getRespCorrec()*. Comprobamos el número de respuestas mostradas en pantalla: si solo hay una, es una pregunta para introducir por teclado la respuesta, por lo que se le asigna la respuesta correcta; si por el contrario hay más de una, se comprueba una a una las respuestas cargadas en pantalla para encontrar la correcta, asignándola en ese caso a un objeto *RadioButton* llamado *rb*, que posteriormente se situará como "checkeado", se incrementará su tamaño de letra y se pondrá en cursiva para que resalte entre las demás.

- *private void Opcionesbutton_Click(object sender, EventArgs e)*:

  Evento que se produce cuando se pulsa sobre el botón *Opcionesbutton*. Este botón tiene como misión abrir la ventana de opciones de la aplicación, donde introducir los datos de conexión de la base de datos, insertar datos en la base de datos, consultar valores, etc.
  Se comprueba si la ventana ya está abierta viendo si está inicializado. Si no lo está, se crea el objeto, se suscribe al evento de detectar el cierre de esta nueva ventana y se indica que se muestre (mediante *Show()*); si está inicializado, se pone la ventana sobre las demás, consiguiéndolo con el método *Focus()*.

- *void OpcF_FormClosed(object sender, FormClosedEventArgs e)*:

  Evento que se produce cuando la ventana de opciones *OpcF* se cierra. Se quiere detectar el cierre para actualizar datos de la barra de estado.
  Solamente comprueba si se está conectado o no a la base de datos, se des-suscribe al objeto *OpcF* del evento de cierre de ventana, y se libera.

- *private void aumentarButton_Click(object sender, EventArgs e)*:

  Evento que se produce cuando se pulsa sobre el botón *aumentarButton* para aumentar el tamaño de la fuente de la pregunta y las respuestas mostradas por pantalla. Este botón se encuentra en la parte izquierda del cuadro donde se cargarán las posibles respuestas.
  Puesto que se aumentará la fuente tanto de la pregunta como de las respuestas, si las respuestas no cambian de ubicación, unas taparán a otras, por lo que, para evitar esto, se emplea la variable entera *aumento*,

inicializada a *0*. En primer lugar se aumenta el tamaño de fuente de la pregunta. Después, si hay respuestas en *RespuestasgroupBox*, aumenta el tamaño de cada una, e incrementa la variable *aumentar* para ir cambiando la ubicación de las respuestas, evitando así que se machaque entre ellas.

- *private void disminuirbutton_Click(object sender, EventArgs e)*:

  Este evento se produce al pulsar sobre el botón de disminución de fuente, *disminuirbutton*, situado en la parte izquierda del cuadro donde se cargarán las respuestas. Tiene la misma misión que el método anterior, funcionando de igual manera pero disminuyendo el tamaño de la fuente.

- *private void Form1_FormClosing(object sender, FormClosingEventArgs e)*:

  Evento que se produce cuando se pulsa sobre la "X" de cierre de aplicación, en la parte superior derecha de la ventana principal, y que provoca el cierre de la aplicación. Puesto que se utilizan numerosas variables, y para evitar que, por ejemplo en el caso de los hilos, se queden en ejecución, es necesario detectar este cierre.
  Lo único que se hace es lanzar el evento del botón *salir_button*, para que sea éste quien libere recursos y cierre la aplicación.

3.4.5. Clase OpcionesForm

Clase implementada para dar más opciones al usuario, y no sobrecargar la ventana principal. Al igual que la clase anterior, es una clase de interfaz gráfica de usuario que permite a éste explotar las funcionalidades de la ventana a través de componentes gráficos que facilitan su navegación por la aplicación.

En esta clase, además de proporcionar la interfaz de conexión con la base de datos, también se pretende abstraer al usuario de tener conocimientos sobre bases de datos, pues se le facilitan, mediante una sencilla interfaz de usuario, todas las funcionalidades que necesitará, como crear una base de datos, crear tablas de preguntas, introducir preguntas, modificar el número de respuestas posibles de una tabla de preguntas, ver alumnos contenidos en la base de datos, ver preguntas, ver respuestas (estas tres últimas con opciones de búsqueda), … Además, también se da la opción de exportar los datos de la base de datos a un documento, de forma que se puedan tratar, por ejemplo, con *Microsoft Excel.*

Variables:

Solo se dispondrán de unas pocas variables puesto que esta clase no se encarga de controlar ninguna ejecución más allá de la suya misma. Todas las variables mencionadas a continuación serán privadas.

- *infoBaseDatos auxInfBD*: objeto de la clase especificada desarrollada para guardar los datos de conexión a la base de datos.

- *FormAlumnos alumnwindow*: objeto de la clase *FormAlumnos*, clase creada para visualizar la información referente a alumnos guardados en la base de datos. Es una interfaz gráfica. Se inicializa a valor nulo.
- *FormVerPreg fvp*: objeto de la clase *FormVerPreg*, desarrollada para mostrar al usuario la información guardada en la base de datos sobre las preguntas guardadas. Es una interfaz gráfica. Inicializada a valor nulo.
- *FormRespAlumn fra*: objeto de la clase mencionada, implementada para mostrar al usuario las respuestas dadas a las preguntas. Es una interfaz de usuario. Inicializada a null.
- *FormExportarDatos exportar*: objeto de la clase especificada, escrita para exportar todos los datos de la base de datos a un documento que pueda ser tratado mediante otros programas. Es una interfaz gráfica. Inicializada a valor nulo.

Al tratarse de una clase con interfaz gráfica, también se dispone de una serie de elementos o componentes gráficos, cuyo código elabora de manera automática el entorno de desarrollo al crear la apariencia visual mediante el editor gráfico. Dichos elementos son los que siguen:

- *Button infoBDbutton*: botón para mostrar en pantalla el formulario para especificar los datos de conexión con la base de datos.
- *Button InsrtPregbutton*: botón para mostrar por pantalla el formulario para insertar una pregunta en la base de datos.
- *Button Cerrarbutton*: botón para cerrar la presente ventana o interfaz gráfica.
- *Label servAddrLabel*: etiqueta de texto situada junto a otro elemento para dar descripción de utilidad de dicho elemento. Contenido en *groupBox1*.
- *Label BDlabel*: etiqueta de texto situada junto a otro elemento para dar descripción de utilidad de dicho elemento. Contenido en *groupBox1*.
- *Label Userlabel*: etiqueta de texto situada junto a otro elemento para dar descripción de utilidad de dicho elemento. Contenido en *groupBox1*.
- *Label passlabel*: etiqueta de texto situada junto a otro elemento para dar descripción de utilidad de dicho elemento. Contenido en *groupBox1*.
- *Label puertoLabel*: etiqueta de texto situada junto a otro elemento para dar descripción sobre utilidad de dicho elemento. Contenido en *groupBox1*.
- *TextBox servAddrTextBox*: control de cuadro de texto para introducir la dirección IP de conexión con la base de datos. Se encuentra en *groupBox1*.
- *TextBox BDtextBox*: control de cuadro de texto para introducir el nombre de la base de datos a la que conectarse. Se encuentra en *groupBox1*.
- *TextBox UsertextBox*: control de cuadro de texto para introducir el usuario para autenticación con la base de datos. Se encuentra en *groupBox1*.
- *TextBox PasswtextBox*: control de cuadro de texto para introducir la contraseña de usuario para autenticación con la base de datos. Se encuentra en *groupBox1*.
- *TextBox PuertotextBox*: control de cuadro de texto para introducir el puerto de conexión con la base de datos. Contenido en *groupBox1*.

- *Button Aplicarbutton*: botón para conectar/crear base de datos. Está contenido en *GroupBox1*.
- *GroupBox groupBox1*: agrupador de controles que contiene los componentes necesarios para introducir los datos de conexión a la base de datos, así como su creación.
- *GroupBox groupBoxInsrtPreg*: agrupador de controles que agrupa los controles de utilidad para insertar una pregunta (junto con sus posibles respuestas) en la base de datos.
- *Button Ocultarbutton*: botón para ocultar/mostrar la cadena escrita como *password* en el formulario de conexión/creación de base de datos. Se encuentra en el grupo de controles *groupBox1*.
- *NuemricUpDown TemaNumericUpDown3*: componente numérico para seleccionar el número de tema sobre el que insertar la pregunta en la base de datos. Está contenido en *groupBoxInsrtPreg*.
- *Button Insertarbutton*: botón para insertar una pregunta en la base de datos. Contenido en *groupBoxInsrtPreg*.
- *Label TemaPregLabel*: etiqueta de texto añadida junto al elemento *TemaNumericUpDown3* para proporcionar una breve descripción del mismo. Contenido en *groupBoxInsrtPreg*.
- *Label PregLabel*: etiqueta de texto añadido junto al componente *pregTextBox* para proporcionar una pequeña descripción del mismo. Contenido en *groupBoxInsrtPreg*.
- *TextBox RespCorrectextBox*: control de cuadro de texto para introducir la respuesta correcta de la pregunta indicada. Contendio en *groupBoxInsrtPreg*.
- *Label RespCorreclabel*: etiqueta de texto situada junto a *RespCorrectextBox* para aportar una breve indicación de su función. Contenido en *groupBoxInsrtPreg*.
- *TextBox PregtextBox*: control de cuadro de texto para introducir el enunciado de la pregunta a introducir en la base de datos. Contenido en *groupBoxInsrtPreg*.
- *CheckBox CrearBD_checkBox*: checkbox para indicar en el formulario de conexión a la base de datos que se quiere crear la base de datos especificada. Contenido en *groupBox1*.
- *NumericUpDown NumPregnumericUpDown*: componente numérico para seleccionar el número de pregunta que se quiere insertar en la base de datos. Conenido en *groupBoxInsrtPreg*.
- *Label label1*: etiqueta de texto situada junto a *NumPregnumericUpDown* para aportar una breve descripción de la misión del componente. Contenido en *groupBoxInsrtPreg*.
- *Button InsertTablabutton*: botón para insertar pregunta en la base de datos. Contenido en *groupBoxInsrtPreg*.
- *GroupBox groupBox2*: agrupador de controles que agrupa los componentes empleados para la creación y edición de tablas de preguntas.
- *Button ElimiTablabutton*: botón para eliminar tabla de preguntas de la base de datos. Contenido en *groupBox2*.

- *Button InsrtTablaButton*: botón para insertar tabla/editar tabla de preguntas en la base de datos. Contenido en *groupBox2*.
- *Button CargarInsrtPregbutton*: botón para cargar tantas respuestas a introducir como contenga la tabla. Contendido en *groupBoxInsrtPreg*.
- *Button VerPregBDbutton*: botón que abre la ventana de interfaz de usuario para ver las preguntas disponibles en la base de datos.
- *Button AlumnBDbutton*: botón que abre la ventana de interfaz de usuario para visualizar los alumnos que hay en la base de datos.
- *Button RespBDbutton*: botón que abre la ventana de interfaz de usuario para ver las respuestas recopiladas a las preguntas emitidas.
- *NumericUpDown NumRespnumericUpDown2*: componente numérico para indicar el número de respuestas de las preguntas del tema especificado para crear la tabla de preguntas de dicho tema. Contenido en *groupBox2*.
- *Label NumResplabel2*: etiqueta de texto situada junto a *NumRespnumericUpDown2* para aportar una breve descripción del mismo. Contenido en *groupBox2*.
- *NumericUpDown NumTemanumericUpDown2*: componente numérico para indicar el número de tema para el que se va a insertar una tabla de preguntas. Contenido en *groupBox2*.
- *Label label3*: etiqueta de texto situada junto a *NumTemanumericUpDown2* para aportar una breve descripción del mismo. Contenido en *groupBox2*.
- *CheckBox IncrNumRespcheckBox*: checkbox para indicar si se quiere aumentar el número de respuestas para la tabla de preguntas de la base de datos especificada. Contenido en *groupBox2*.
- *Panel IntrResppanel1*: componente para agrupar controles empleado para situar tantos componentes de introducción de respuestas como haya en la tabla correspondiente de la base de datos. Contenido en *groupBoxInsrtPreg*.
- *Button ExportarDatosbutton*: botón que abre la ventana de interfaz de usuario para exportar los datos de la base de datos.

Constructor:

En el constructor se lleva a cabo la inicialización de componentes gráficos y otras variables.

En primer lugar, se realiza la inicialización de componentes gráficos mencionada mediante la función *InitializeComponent()*, función heredada de la clase *Form* donde se generan todos los componentes añadidos a la ventana diseñada. A continuación, se establece el título de *groupBox1* y de *groupBoxInsrtPreg*, y se rellenan los *TextBox* del formulario de conexión con la base de datos (*groupBox1*) con los valores del objeto *infoDB* perteneciente a la clase *Form1*, valores a los que se accede mediante los métodos *get* correspondientes.

Métodos de atención de eventos:

Como se ha mencionado en la explicación de la clase anterior (*3.4.4. Clase Form1*), al tratarse de una interfaz gráfica de usuario, los componentes, al interactuar con el usuario, lanzan eventos. Así que habrá que implementar los métodos que modelen el evento.

- *private void Aplicarbutton_Click(object sender, EventArgs e)*:

  Método del evento que se lanza cuando el usuario pulsa sobre el botón *Aplicarbutton*, es decir, el botón que aparece en el formulario de inserción de datos para conectarse a la base de datos. Depende de la selección hecha en el formulario, puede realizar la conexión con la base de datos, o crear la base de datos.
  En primer lugar se comprueban una serie de escenarios sobre los que no se puede realizar esta acción. Éstos son cuando el servidor se encuentra en ejecución y se quiere conectar a otra base de datos, y en caso de que no se haya introducido base de datos o el formato de la dirección IP escrita no sea válido. En todos estos casos, se le notifica al usuario el motivo por el que no se puede seguir adelante.
  Se diferencia entre crear base de datos o realizar conexión a ella. Así, si se quiere crear la base de datos, se crea un nuevo conector, *MSC*, se crea la cadena de conexión, se le asigna a ese conector, y se intenta conectar a la base de datos especificada. Si no se puede conectar es que la información introducida no es correcta. En caso afirmativo, se crea la consulta para crear la base de datos, y se ejecuta. Si se produce algún error, se notifica al usuario y finaliza la ejecución. En caso contrario, se crea automáticamente la tabla de alumnos, por lo que se construye la consulta y se ejecuta. Si capturamos una excepción, es decir, si se produce un error, eliminamos la base de datos creada puesto que no se ha completado el ciclo completo de creación de la base de datos. Si todo se ejecuta según lo previsto, si no se está conectado a ninguna base de datos, se le pregunta al usuario si quiere realizar la conexión con la base de datos, procediendo a la conexión si así lo indica desactivando el checkbox de crear base de datos, y provocando de nuevo el lanzamiento del evento de pulsación del botón *Aplicarbutton*.
  Si por el contrario lo que se quiere es conectarse a la base de datos, se comprueba que los datos son válidos realizando un intento de conexión. Si resulta exitoso, se cierra esta conexión, se actualiza el objeto *infoDB* de la clase *Form1*, y se conecta a la base de datos llamando al método estático *conectarConBD()* de la clase *Form1*. Si se produce error en la validación de la información de conexión a la base de datos, se identifica el tipo de error y se le notifica al usuario.

- *private void infoBDbutton_Click(object sender, EventArgs e), private void InsertTablabutton_Click(object sender, EventArgs e), private void InsrtPregbutton_Click(object sender, EventArgs e)*:

Métodos ejecutados cuando el usuario el evento de pulsación del botón *infoDBbutton* (texto: "Información de la base de datos"), *InsertTablabutton* (texto: "Insertar o Borrar Tabla") y *InsrtPregbutton* (texto: "Insertar pregunta en BD"), respectivamente. La misión de estos tres botones es la misma cambiar entre los formularios correspondientes. Es decir, al pulsar sobre el primer botón mencionado se visualiza el formulario para introducir los datos de conexión o creación de base de datos; al pulsar el segundo de ellos aparece el formulario para insertar o borrar una tabla de preguntas de la base de datos; y el tercero activa el formulario para insertar preguntas en la base de datos. Esto se consigue gracias a la propiedad *Visible* de todos los componentes gráficos, y ya que se tienen agrupados en *GroupBox*, solo hay que jugar con dicha propiedad, dejando visible el *GroupBox* deseado.

- *private void Ocultarbutton_Click(object sender, EventArgs e)*:

  Método ejecutado al lanzarse el evento de pulsación del botón de ocultar/mostrar texto de la respuesta correcta al introducir los datos sobre la pregunta a introducir en la base de datos, en el formulario para tal fin, dando privacidad a este dato. Para conseguir este objetivo, se cambia el nombre del botón cada vez que se pulsa, cambiando también entre la ocultación o no del texto.

- *private void Insertarbutton_Click(object sender, EventArgs e)*:

  Método ejecutado al lanzarse el evento de pulsación del botón de introducir pregunta en la base de datos, en el formulario que se corresponde con dicha acción.
  Las condiciones que se comprueban para saber si es posible realizar esta acción son comprobar que se encuentra conectado a la base de datos, y que tanto la pregunta como la respuesta correcta no se encuentran vacías. Pasadas estas condiciones, también se comprueba que la respuesta correcta se encuentra entre las posibles respuestas especificadas. Para ello se obtienen los controles de introducción de respuestas contenidos en *IntrResppanel1*, y se comprueba si alguno de ellos coincide con la respuesta correcta. También se comprueba si son todas las respuestas posibles vacías, pues esto indica que se trata de una pregunta en la que no hay que elegir respuesta, sino escribirla.
  Si se trata de una pregunta del último tipo mencionado, o la respuesta correcta se encuentra entre las posibles respuestas, se crea la consulta a realizar a la base de datos para introducir la pregunta. Para componer esta consulta se obtienen todos los controles de tipo *TextBox* que son de respuestas. Esto se logra con el método *groupBoxInsrtPreg.Controls.Find()*.

Una vez formada la consulta, se ejecuta, y se llama al método *limpiarCampos()* para limpiar el texto introducido en el formulario para crear la pregunta. Si se produce algún error, se le notifica al usuario.

- *private void Cerrarbutton_Click_1(object sender, EventArgs e)*:

  Método ejecutado al saltar el evento de pulsación del botón de cerrar ventana (*Cerrarbutton*). Simplemente cierra el formulario, puesto que no hay recursos que liberar.

- *private void InsrtTablabutton_Click(object sender, EventArgs e)*:

  Método ejecutado al pulsar el botón de "Insertar tabla" o "Incrementa respuestas", en el formulario de insertar/editar/eliminar tablas. Dependiendo de si está activado o no el checkbox *IncrNumRespcheckBox* (con el texto "Incrementar el número de respuestas") incrementará el número de respuestas para la tabla de preguntas del tema especificado, o creará una tabla de preguntas para el tema indicado.
  Se comprueba en primer lugar que se encuentre conectado a la base de datos y que el número de respuestas a incrementar no sea *0* si se ha activado el checkbox correspondiente. Si está activado el checkbox de incrementar el número de respuestas, se pide confirmación al usuario. Se copia el conector *conexion* de la clase *Form1*, y se construye la consulta a ejecutar en la base de datos de forma adecuada, dependiendo de la acción que se quiera realizar.
  Si se opta por crear una nueva tabla de preguntas para un nuevo tema, además de crear esta tabla se creará automáticamente la tabla de respuestas correspondiente. Si se crean con éxito, se resetean los valores de los componentes gráficos de dicho formulario mediante el método *limpiarCampos()*.
  Por el contrario, si se opta por incrementar el número de respuestas para la tabla de preguntas del tema especificado, en primer lugar se obtiene el número de repuestas que ya hay, así que se crea la consulta, se ejecuta, y se calcula por el número de columnas que devuelve. Es decir, se sabe que de las columnas que devuelva, una es el identificador de la pregunta, otra la pregunta en sí, y el tercero la respuesta correcta; el resto de campos serán las respuestas posibles. Por lo tanto, averiguados el número de respuestas, se construye la consulta para aumentar el número de respuestas, que no es más que aumentar el número de columnas, y se ejecuta.

- *private void ElimiTablabutton_Click(object sender, EventArgs e)*:

  Método ejecutado por el evento provocado al pulsar sobre el botón *ElimiTablabutton*, del formulario de crear/editar/eliminar tablas de preguntas en la base de datos. Este método elimina la tabla indicada.

En primer lugar comprueba si se encuentra ejecutándose el servidor (en ese caso no se puede borrar ninguna tabla), y si se está conectado a la base de datos. Se solicita confirmación del usuario para llevar a cabo la eliminación de la tabla. Si decide continuar, se construye la consulta de eliminación de tabla en la base de datos y se ejecuta, informando de los resultados al usuario.

- *private void CrearBD_checkBox_CheckedChanged(object sender, EventArgs e)*:

  Método que se ejecuta cuando se lanza el evento de cambio en la activación o no de checkbox. Este checkbox se corresponde con el del formulario de conectar o crear base de datos.
  Si está activado, el texto del botón *AplicarButton* se inicializa a "Crear Base de Datos"; si no lo está, se iguala a "Conectar a Base de Datos".

- *private void CargarInsrtPregbutton_Click(object sender, EventArgs e)*:

  Método que se ejecuta cuando se lanza el evento al pulsar sobre el botón *CargarInsrtPregbutton*, en el formulario de insertar pregunta en la base de datos, para cargar los componentes necesarios sobre las que introducir las posibles respuestas, y se cargarán tantos como respuestas a introducir haya en la tabla.
  Se quiere conocer el número de respuestas posibles para las preguntas del tema indicado, por lo que se crea una consulta para obtener una única fila y determinar el número de respuestas tal y como se ha hecho en métodos anteriores. Obtenidos el número de posibles respuestas a introducir, se liberan los recursos de la colección, y se limpian los controles del panel por si ya había respuestas. Por cada respuesta a introducir se crea un par *Label TextBox*, y se añaden al panel.
  Añadidas todos los controles al panel, se ponen visibles el resto de componentes del formulario.

- *private void AlumnBDbutton_Click(object sender, EventArgs e)*:

  Método que se ejecuta al lanzarse el evento de pulsación de botón de ver alumnos en la base de datos, es decir, *AlumnBDbutton*. Este botón abrirá una ventana donde visualizar los datos de la base de datos referente a los alumnos presentes en ésta si se encuentra conectado a la base de datos. Si la ventana ya se encuentra abierta, se le da el foco para que se visualice por encima de las demás.

- *void alumnwindow_FormClosing(object sender, FormClosingEventArgs e)*:

  Método que se ejecuta cuando se cierra la ventana de visualización de alumnos presentes en la base de datos. Libera recursos.

- *private void VerPregBDbutton_Click(object sender, EventArgs e)*:

  Método que se ejecuta cuando se lanza el evento correspondiente al pulsar el botón de ver preguntas guardadas en la base de datos.
  Si ya se encuentra abierta le da el foco para que se muestre por encima de las demás; si se encuentra conectado a la base de datos y la ventana no está abierta, crea una nueva ventana donde consultar las preguntas presentes en la base de datos.

- *void fvp_FormClosing(object sender, FormClosingEventArgs e)*:

  Método que se ejecuta cuando se cierra la ventana para ver las preguntas introducidas en la base de datos. Libera recursos.

- *private void RespBDbutton_Click(object sender, EventArgs e)*:

  Método que se ejecuta al lanzarse el evento de pulsación de botón de visualizar respuestas de la base de datos. Se comporta de igual modo que los anteriores, solo que abriendo en este caso la ventana de consulta de respuestas dadas a preguntas en la base de datos.

- *void fra_FormClosing(object sender, FormClosingEventArgs e)*:

  Método que se ejecuta cuando se cierra la ventana de visualización de respuestas dadas a las preguntas formuladas. Libera recursos.

- *private void IncrNumRespcheckBox_CheckedChanged(object sender, EventArgs e)*:

  Método que se ejecuta cuando se lanza el evento de cambio de estado en el checkbox de incrementar número de respuestas, en el formulario de crear o eliminar tabla de preguntas.
  Si se activa, cambia la localización del *NumRespnumericUpDown2* y se cambia el texto del *NumResplabel2* a "Número de Respuestas a incrementar:". Además, se quita la visibilidad el botón de eliminar tabla, y se redimensiona el botón de inserta tabla, sustituyendo su nombre por "Incrementa Respuestas".
  Cuando se desactiva el checkbox ocurre el proceso inverso.

- *private void ExportarDatosbutton_Click(object sender, EventArgs e)*:

  Método que se ejecuta al lanzarse el evento de pulsación de botón de exportar datos. Su funcionamiento es similar al de los botones de apertura de nuevas

ventanas explicados varios párrafos atrás, pero abriendo una ventana donde se solicita al usuario que indique los datos que desea exportar.

- *void exportar_FormClosing(object sender, FormClosingEventArgs e)*:

  Método que se ejecuta cuando se cierra la ventana de exportación de datos. Libera recursos.

Otros Métodos:

- *public void limpiarCampos(GroupBox gb)*:

  Método empleado para resetear los componentes en los que se introduce el texto requerido para realizar la acción deseada. Por ejemplo, cuando hemos introducido los datos para insertar una pregunta en la base de datos y se inserta de forma satisfactoria, automáticamente se reinician todos esos campos para no tener que ser borrados de forma manual.

3.4.6. Clase FormVerPreg

Clase desarrollada para dar al usuario una interfaz gráfica donde visualizar los datos de la base de datos referentes a las preguntas de que se dispone, con la idea de abstraerlo de tener que disponer de conocimientos sobre bases de datos o *MySQL*. Permite varias opciones para la visualización de las preguntas de la base de datos. Pueden cargarse todas las preguntas del número de tema especificado, o el número de pregunta del número de tema especificado.

Variables:

Para el desempeño de su labor no se ha necesitado ninguna variable global. Como variables u objetos que representan componentes gráficos, tenemos los siguientes:

- *Label label1*: etiqueta de texto que se sitúa cerca del control *PregBDnumericUpDown* para proporcionar una breve descripción del cometido de ese control.
- *NumericUpDown PregBDnumericUpDown*: componente numérico para seleccionar el número de tema del que se quiere cargar la pregunta de la base de datos.
- *Button VerPregBDbutton*: botón para cargar la/s pregunta/s indicadas según las preferencias seleccionadas en la ventana.
- *Label BDlabel*: etiqueta de texto situada a modo de título para indicar la base de datos sobre la que se están realizando las consultas.
- *DataGridView dataGridViewPreg*: tabla personalizable donde se representan los datos obtenidos de la consulta a la base de datos.
- *NumericUpDown NumPregnumericUpDown*: componente numérico para seleccionar el número de pregunta que se desea visualizar.

- *Label label2*: etiqueta de texto que se sitúa antes del componente *NumPregnumericUpDown* a modo de breve descripción sobre la labor del mismo.
- *CheckBox CargarAllPregcheckBox*: checkbox para seleccionar si se quieren cargar todas las preguntas del tema o solo la indicada.

Constructor:

En el constructor, como se ha explicado en las clases anteriores que proporcionan una interfaz gráfica de usuario, se realiza la instanciación de todos los componentes gráficos que lo forman. Esta función de inicialización de componentes es *InitializeComponent()*.

Métodos:

Los siguientes métodos a describir representan los métodos ejecutados al lanzarse los correspondientes eventos.

- *private void VerPregBDbutton_Click(object sender, EventArgs e)*:

  Método ejecutado al lanzarse el evento correspondiente con el botón de cargar preguntas.
  En primer lugar se copia el conector de la base de datos *conexion* perteneciente a la clase *Form1* a un conector creado localmente: *msc*. Se intenta la conexión a la base de datos. Una vez conectado, se crea la consulta a ejecutar. Si se el checkbox no está activado, se cargará la pregunta indicada del tema especificado; en caso contrario, se solicitarán todas las preguntas del tema seleccionado por *PregBDnumericUpDown*. Construida la consulta, se ejecuta, y se completa un objeto *DataSet, preguntas,* con los datos devueltos. Este objeto representa un conjunto completo de datos, incluyendo las tablas que lo contienen, orden, ... A continuación, se le asigna como datos de origen a *dataGridViewPreg* la tabla del objeto *preguntas* (que contiene los datos de la consulta). Si no se han producido errores (los cuales se le habrían notificado al usuario y habrían provocado el fin de la ejecución del presente método), se actualiza el *label* que indica la base de datos sobre la que se ha realizado la consulta, se cierra la conexión, y si los datos cargados son vacíos debido a que aún no hay entradas, se le indica al usuario.

- *private void CargarAllPregcheckBox_CheckedChanged(object sender, EventArgs e)*:

  Método que se ejecuta cuando el evento de cambio de estado del checkbox es lanzado. Se utiliza para desactivar el componente numérico para seleccionar el número de pregunta a cargar cuando se quiere cargar la tabla de preguntas del tema especificado completa.

3.4.7. Clase FormAlumnos

Clase implementada para ofrecer al usuario una interfaz gráfica donde poder ver los datos de la base de datos referentes a los alumnos que han participado en la aplicación, con la idea de abstraerlo de tener que conocer bases de datos o MySQL. Se ofrece la posibilidad de cargar todos los alumnos con sus datos, o de realizar la búsqueda por cualquiera de sus datos.

Variables:

No se han necesitado variables para la ejecución de esta clase, salvo las de los componentes gráficos que componen la ventana, y que son:

- *DataGridView dataGridViewAlumn*: tabla personalizable donde se representan los datos obtenidos de la consulta a la base de datos.
- *Button Recargarbutton*: botón para recargar la lista de alumnos basándose en los criterios seleccionados.
- *DataGridView dataGridViewAlumn2*: segunda tabla personalizable donde se representarán los datos en caso de que se disponga de muchos, es decir, en caso de que sean muchos datos se representan en dos tablas, para hacerlo más legible.
- *ComboBox BusquedacomboBox*: *combobox* para seleccionar la modalidad de la búsqueda.
- *Label Iguallabel*: etiqueta de texto inicializada con el valor "=", y que se sitúa entre *BusquedacomboBox* y *VariableBuscartextBox* en modo de dar sentido y significado de forma gráfica.
- *TextBox VariableBuscartextBox*: control de cuadro de texto para introducir la condición de búsqueda de alumno en la base de datos.

Constructor:

En el constructor se lleva a cabo la instanciación de todos los componentes gráficos que forman la ventana gráfica de la presente clase mediante la función heredada *InitializeComponent()*. Además, también se realiza una primera carga de datos al abrir la aplicación, lanzando el evento del botón encargado de ello por código.

Métodos:

Los métodos implementados son los que se ejecutan al producirse eventos por parte de los componentes gráficos con los que interactúa el usuario.

- *private void Recargarbutton_Click(object sender, EventArgs e)*:

  Método ejecutado cuando se lanza el evento correspondiente con el botón recargar alumnos.
  En primer lugar se copia el conector a la base de datos disponible en la clase *Form1* (*conexion*) y se intenta conectar a la base de datos. En caso de éxito, se

construye la consulta a ejecutar. Si en el *comboBox* de selección de tipo de búsqueda se ha seleccionado algo, se estarán buscando alumnos que contengan el dato especificado en el criterio de búsqueda. Por tanto se crea la consulta pertinente; en caso de que se estén buscando alumnos por DNI, se busca sin letra y con letra, por si acaso no se ha especificado en el criterio de búsqueda.
Al igual que en la clase anterior, se ejecuta la consulta y se llena un objeto *DataSet* con los datos devueltos de la consulta, y posteriormente, en la tabla de mostrar datos por pantalla (*dataGridViewAlumn*), se le asigna como origen de datos la tabla que contiene el objeto *DataSet*. Por último, se cierra la conexión con la base de datos, se oculta la segunda tabla *DataGridView*, y se fija el tamaño de la ventana.
Si se indica que la consulta sea sobre todos los alumnos, en primer lugar se obtiene el número de los mismos para comprobar si hay que habilitar las dos tablas de datos, pues demasiados datos en la misma tabla puede generar confusión. De esta forma, se obtiene el número de filas que ocupará el resultado de la consulta, y si hay más de 35 filas (número escogido después de realizar pruebas sobre la representación de los datos), se indica que los datos se representarán en dos tablas mediante el *Boolean doblePantalla* situado a *true*. Se construye la consulta para obtener los datos. Si se va a representar a doble pantalla, en la primera consulta se obtendrán la mitad de los datos, y se realizará una segunda consulta para obtener el resto, representando cada parte en un *DataGridView*. En caso de no emplear las dos tablas, solo se ejecuta la primera consulta, en la que se cargan todos los datos.
Por último, se pone título a la ventana con el nombre de la base de datos de la que se han obtenido los datos, y se cierra la conexión.

- *private void BusquedacomboBox_SelectedIndexChanged(object sender, EventArgs e)*:

  Método que se ejecuta cuando el evento de cambio elemento seleccionado en el *comboBox* es lanzado. Se utiliza para mostrar el *label* con el signo "=" y el *TextBox VariableBuscartextBox* solo cuando sea necesario, es decir, cuando en el *comboBox* se seleccione un criterio de búsqueda.

3.4.8. Clase FormRespAlumn

Clase desarrollada para brindar al usuario de una interfaz gráfica donde poder visualizar los datos de la base de datos referentes a las preguntas que se han contestado hasta el momento, con la idea de abstraerlo de tener que conocer bases de datos o MySQL. Además de cargar las preguntas para una pregunta de un tema especificado, permite también ver qué alumnos han contestado con la respuesta indicada en el *TextBox* a la pregunta del tema especificada.

Variables:

Las variables empleadas son las de los objetos de los componentes gráficos, que son:

- *NumericUpDown NumTemanumericUpDown*: componente numérico para seleccionar el número de tema que contendrá la pregunta sobre la que se desea averiguar las respuestas.
- *Label Temalabel*: etiqueta de texto situada a la izquierda de *NumTemanumericUpDown* para aportar una breve descripción del control.
- *NumericUpDown NumPregnumericUpDown*: componente numérico para seleccionar el número de pregunta sobre el que consultar las respuestas.
- *Label Preglabel*: etiqueta de texto situada a la izquierda de *NumPregnumericUpDown* que aporta una breve descripción del componente.
- *TextBox ResptextBox*: control de cuadro de texto para introducir la respuesta a buscar en las respuestas guardadas para la pregunta especificada del tema indicado.
- *Label Resplabel*: etiqueta de texto situada a la izquierda de *ResptextBox* que aporta una breve descripción del control.
- *Button CargarRespbutton*: botón para cargar las respuestas con los criterios de búsqueda seleccionados.
- *GroupBox infoPregRespBDgroupBox*: contendedor que alberga todos los componentes gráficos descritos.
- *DataGridView dataGridViewRespAlumn*: tabla personalizable donde se representan los datos obtenidos de la consulta a la base de datos.
- *DataGridView dataGridViewRespAlumnAlumn2*: tabla personalizable donde se representan los datos obtenidos de la consulta a la base de datos.

Constructor:

En el constructor se realiza la instanciación de componentes gráficos mediante la función heredada de la clase *Form InitializeComponent()*.

Métodos:

El único método implementado para esta clase es que responde al evento de pulsación de botón de obtener respuestas (botón *CargarRespbutton*).

- *private void CargarRespbutton_Click(object sender, EventArgs e)*:

  Método que se ejecuta cuando se lanza el evento de pulsación del botón de cargar respuestas.
  En primer lugar se clona el conector a la base de datos de la clase principal *Form1* mediante la función *Clone()*, que devuelve un objeto similar. Con este conector, *msc*, se abre la conexión con la base de datos. En caso de no producirse errores, se crea la consulta a ejecutar. Si se ha indicado una respuesta en el *textBox* correspondiente, se añade a la consulta que las respuestas sean como la introducida, pudiendo ver de esta forma los alumnos

que han contestado a una determinada respuesta. Creada la consulta, y para representar mejor la información que se va a obtener de la base de datos, se consulta primero el número de elementos con esas condiciones, y se vuelve a valorar, de la misma forma que se hacía en la clase anterior, si se representan los datos en dos *dataGridView*. Se crea la consulta para obtener los datos de la base de datos, se cargan en un *DataSet*, y se asigna la tabla que contiene como origen de datos del *DataGridView dataGridViewRespAlumn*. Si se ha optado por emplear dos tablas debido a la cantidad de información devuelta por la consulta, se crea una segunda consulta que cargará los datos que le quedan por obtener de la base de datos, los asignará de nuevo a un *DataSet*, y la tabla que contiene se asigna como origen de datos del *DataGridView dataGridViewRespAlumnAlumn2*, y se modifica el tamaño de la ventana. Si solo se empleará una *DataGridView*, se fija el *dataGridViewRespAlumnAlumn2* como no visible, y se pone como tamaño de pantalla el reducido.

Por último, actualizamos el título de la ventana mostrando información de la base de datos de la que se ha obtenido la información, y cerramos la conexión con la base de datos.

3.4.9. Clase FormExportarDatos

Clase implementada para brindar al usuario de una interfaz gráfica mediante la cual poder exportar la información contenida en la base de datos a un archivo procesable por programas como *Microsoft Excel*.

Variables:

Se han empleado una serie de variables para controlar la ejecución de la aplicación, todas ellas privadas.

- *Thread exportar*: representa un objeto proceso o hilo empleado para realizar la exportación de datos sin quitar el control de la aplicación a la ventana que lo controla.
- *StreamWriter sw*: *stream* de escritura para escribir en los archivos especificados.
- *String rutaAlumn*: cadena que contiene la ruta donde se guardará el archivo con los datos de los alumnos y sus respuestas.
- *String rutaPreg*: cadena que contiene la ruta donde se guardará el archivo con las respuestas para cada tema.
- *int progresionTablas*: progresión que se lleva al procesar las tablas.
- *int pregresionFilas*: progresión que se lleva al procesar filas.
- *int total*: progresión total.

Además de estas variables, también se pueden encontrar las de los componentes gráficos creados para la ventana, que son:

- *SaveFileDialog saveFileDialog*: representa un cuadro de diálogo para que el usuario especifique dónde quiere guardar el archivo, y el nombre que le pondrá.
- *Button CancelarExpButton*: botón para cancelar la exportación de datos en curso.
- *ProgressBar ExportarprogressBar*: barra de progreso para mostrar de forma visual el porcentaje del proceso de exportación de datos que se ha completado.
- *Label Estadolabel*: etiqueta de texto que muestra el estado actualizado del proceso de exportación de datos.
- *CheckBox checkBoxExcelalumn*: *checkBox* para que el usuario active si quiere exportar los datos de alumnos con las respuestas dadas.
- *CheckBox checkBoxExcelPreg*: *checkBox* para que el usuario active si quiere exportar los datos de las preguntas.

Constructor:

En el constructor se inicializan los componentes gráficos añadidos a la ventana (mediante la función *InitializeComponent()*), y se inicia el valor de la barra de progreso a *0*.

Métodos:

Se implementan los siguientes métodos:

- *private void checkBoxExcelAlum_CheckedChanged(object sender, EventArgs e)*:

  Método que se ejecuta cuando se cambia el estado de activación del control *checkBox* de exportación de datos de alumnos.
  Si el *checkBox* está activado, configura el cuadro de diálogo para seleccionar ruta donde guardar archivo (*saveFileDialog*), y se guarda la ruta que ha seleccionado el usuario para guardar el archivo con la información de los alumnos y sus respuestas.
  Si alguno de los dos *checkBox* han sido activados, se cambia el nombre del botón *CancelarExpbutton* a "Exportar"; si no está ninguno activado, se establece a "Cerrar".

- *private void checkBoxExcelPreg_CheckedChanged(object sender, EventArgs e)*:

  Método que se ejecuta cuando se lanza el evento por cambio de estado de activación del *checkBox* de exportar preguntas.
  Tiene el mismo funcionamiento que el método anterior, solo que almacena la ruta donde guardar el archivo con toda la información de todas las preguntas.

- *private void CancelarExpbutton_Click(object sender, EventArgs e)*:

  Método ejecutado al lanzar el evento el botón *CancelarExpbutton*.

En primer lugar se comprueba el texto del control, puesto que dependiendo de éste se ejecutará una acción u otra:

- "Cerrar": se cierra la ventana y se liberan recursos.
- "Cancelar"

  Si la barra de progreso ha alcanzado el 100%, no se hace nada. Si el proceso se encuentra en ejecución, se suspende y se pide confirmación al usuario para eliminar el hilo encargado de exportar los datos. Si confirma, se muestra en *Estadolabel* que se está cancelando, y se cierra el *stream* de escritura en archivo, el hilo, la conexión con la base de datos, y se eliminan los archivos que se estaban creando con los datos de la base de datos. Por último, se pone el nombre del botón a "Cerrar", y se lanza el evento por código.

  Si el usuario decide que no quiere abortar la exportación de datos, se dice al proceso que continue.
- "Exportar"

  Se quita la visibilidad a ambos *checkBox*, se pone la barra de progreso visible, se vuelve a obtener el conector de la base de datos a partir del conector de la clase principal *Form1*, se crea el hilo de ejecución y se le da comienzo.

- *public void exportarACsv()*:

  Método que ejecuta el hilo que se encarga de exportar los datos.
  Se comienza conectándose a la base de datos, y se obtienen las tablas del tipo "temaN" (donde N representa un número entero), guardando el resultado en un *DataTable*. Se calcula el porcentaje del total (100%), dependiendo de si se van a obtener solo los datos de alumnos, o también de respuestas.
  Si se quiere obtener la información de alumnos y sus respuestas, en primer lugar se crea el *stream* de escritura en archivo con la ruta del archivo seleccionada. Se escribe en el fichero de texto el nombre de la base de datos, y se calculan el progreso para incrementar el valor de la barra de progreso. Para los resultados obtenidos de la base de datos y guardados en un *DataTable*, que contiene las tablas de la forma "temaN", para cada fila se ejecuta una consulta a la base de datos obteniendo la información de los alumnos que contestaron a las preguntas de ese tema. A continuación se obtienen las respuestas para el tema tratado. Primero se escriben los títulos de los campos de los alumnos (nombre, apellidos, MAC, DNI, …), y después los títulos de las preguntas del tema en el que se encuentra el bucle. Por último, se introduce la información del alumno junto con sus respectivas respuestas mediante un bucle que recorre todas las filas de los alumnos obtenidos en la consulta.
  Todo este proceso se repite para cada fila de la tabla que contiene los temas que existen en esa base de datos.
  Cuando finaliza el bucle, libera recursos (el *stream* de escritura, tablas *DataTable* empleadas, …).

Si está activado el *checkBox* de exportar información de todas las preguntas, se crea el *stream* de escritura en archivo con la ruta donde guardar esta información (*rutaPreg*). Se realiza un bucle en el que, para cada tema, se obtienen sus preguntas y respuestas, y se escribe en el archivo.
Terminados de exportar los datos indicados, se cierra la conexión con la base de datos y se indica al usuario.

3.4.10. Clase FormInfoProcesos

Clase creada para mostrar al usuario una interfaz gráfica indicándole que se están terminando de procesar los datos. Esta ventana aparecerá cuando se quiera pausar el servidor, mientras se procesan los últimos clientes *bluetooth* y las últimas respuestas. Una vez procesados, se cierra.

No dispone de variables, salvo un componente gráfico, un *Label*, que indica el motivo por el que no se cierra la aplicación

.

3.5. Desarrollo de Aplicación Cliente

Por aplicación cliente se refiere a aplicación que se ejecuta en el lado del dispositivo móvil. La misión de esta aplicación es obtener la pregunta a contestar del servidor, mostrársela al usuario, recoger la respuesta que éste seleccione, y enviarla de vuelta a la aplicación servidor.

Como ya se mencionó en el capítulo *3.2. Problemas*, la conexión del dispositivo móvil con el servidor a través de *Bluetooth* no se ha podido realizar con el dispositivo móvil actuando como cliente y el servidor esperando conexiones, sino que se ha tenido que adoptar la solución de hacerlo al contrario; es decir, el cliente espera recibir conexiones, y el servidor es quien inicia la conexión. En la parte del servidor ya se ha explicado la solución adoptada para hacer ese proceso lo menos costoso y más rápido posible. Ahora se va a detallar las medidas llevadas a cabo en la parte del dispositivo móvil para colaborar en la solución adoptada.

De esta forma, el dispositivo móvil hace uso de los identificadores de servicio para indicar si quiere recibir una pregunta o quiere enviar la respuesta, de forma que el servidor sabrá qué hacer cuando se conecte al dispositivo móvil. Además, para evitar que el servidor esté encontrando continuamente estos dispositivos, solo se pondrán en modo visible cuando quieran interactuar con el servidor. Puesto que el dispositivo móvil es el que se queda a la espera de recibir conexiones, en principio cualquier dispositivo podría conectarse a él. Para evitarlo, cuando el dispositivo móvil recibe una conexión, comprueba si es el servidor, y si no lo es, cierra la conexión y se pone a la espera de otra. Pero ahora surge el problema de que puede que el servidor no se encuentre dentro del alcance, y el dispositivo móvil esté esperando de forma continuada. Esto se evita colocando un *timeout* que, en caso de que durante un tiempo no se conecte el servidor, se le notifica al usuario.

Desde un principio se incorporó una forma de iniciar la aplicación solo si el servidor se encontraba dentro del alcance, consistente en la realización de una búsqueda de dispositivos cuando se pretendía iniciar la aplicación: si se encontraba el servidor se permitía continuar; en caso contrario, no. Después las pruebas realizadas, y aunque las funciones siguen implementadas, se ha invalidado esta ejecución, pues ralentizaba mucho la comunicación ya que mientras el dispositivo móvil realizaba la búsqueda, el servidor no podía encontrarlo y por lo tanto tampoco atenderlo, teniendo que esperar una nueva búsqueda.

Para evitar la incomodidad del usuario de tener que estar introduciendo para cada pregunta sus datos, y después realizar una búsqueda de dispositivos para seleccionar el dispositivo que va a permitir que se conecte (el servidor), se ha creado una ventana de configuración donde se proporciona lo necesario para introducir estos datos de forma cómoda.

Como se mencionó en el capítulo del presente proyecto dedicado a Android (*2.1.4. Entorno de desarrollo*), a la hora de desarrollar la aplicación, se realiza la interfaz gráfica de usuario y las variables que utiliza por un lado, mediante lenguaje *xml*, y se le da la funcionalidad mediante código. Además, no hay que olvidar el manifiesto para dar permisos a la aplicación. La versión mínima para la que se puede emplear *Bluetooth* es la versión de software 2.0.

3.5.1. Values

En *values* se han creado dos archivos: *strings.xml* y *colores.xml*. El primero de ellos contendrá las cadenas con los títulos de cada una de las ventanas, texto de botones, de etiquetas de texto, … El archivo *colores.xml* contiene colores empleados en la aplicación para el fondo, texto, botones…

strings.xml:

Este archive se presenta tal cual, pues los datos están bien ordenados y se puede ver el valor de forma sencilla sin entender sobre este tipo de documentos.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">RESPONDE</string>
   <string name="atras">Atrás</string>
            <!-- ============================== -->
            <!--      Contestar Pregunta        -->
            <!-- ============================== -->
   <string name="responder_label">Contestar Pregunta</string>
   <string name="responder_titulo">Contesta Pregunta</string>
   <string name="respuesta">Respuesta:</string>
   <string name="enviar_label">Enviar</string>
   <string name="pregunta_inicio">Cargando Pregunta...</string>
   <string name="reintentar_button">Reintentar</string>
            <!-- ============================== -->
            <!--          Configuración         -->
            <!-- ============================== -->
   <string name="configuracion_label">Configuración</string>
   <string name="configuracion_titulo">Configuración de
RESPONDE</string>
   <string name="conf_shortcut">s</string>
   <string name="conf_serv_name_titulo">Información del
servidor</string>
   <string name="conf_serv_name">Nombre del servidor</string>
   <string name="conf_serv_addr">MAC del servidor</string>
   <string name="conf_alumno_titulo">Información del alumno</string>
   <string name="conf_alumno_name">Nombre del alumno</string>
   <string name="conf_alumno_apellido">Apellidos del alumno</string>
   <string name="conf_alumno_dni">DNI del alumno</string>
            <!-- ============================== -->
            <!--             keys               -->
            <!-- ============================== -->
   <string name="key_serv_name">nombre del servidor</string>
   <string name="key_serv_addr">direccion del servidor</string>
   <string name="key_alumn_nombr">nombre_alumno</string>
   <string name="key_alumn_apellido">apellido_alumno</string>
   <string name="key_alumno_DNI">dni_alumno</string>
            <!-- ============================== -->
```

```
            <!--     Lista de Dispositivos     -->
            <!-- ============================== -->
   <string name="disp_enc_titulo">Dispositivos Encontrados:</string>
   <string name="disp_enc_boton_busc">Buscar dispositivos</string>
   <string name="disp_enc_label">Búsqueda del servidor</string>
            <!-- ============================== -->
            <!--            About               -->
            <!-- ============================== -->
   <string name="about_label">About</string>
   <string name="about_titulo">About RESPONDE</string>
   <string name="about_text">Esta es una aplicación creada por Julio
Alberto...</string>
            <!-- ============================== -->
            <!--             Salir              -->
            <!-- ============================== -->
   <string name="salir_label">Salir</string>
   <string name="salir_titulo">¿Salir de RESPONDE?</string>
   <string name="salir_texto">¿Está seguro de que desea abandonar la
aplicación RESPONDE?</string>
            <!-- ============================== -->
            <!-- Mensajes que muestra el programa -->
            <!-- ============================== -->
   <string name="busq_disp_end">La búsqueda de dispositivos ha
finalizado. Seleccione uno de la lista</string>
   <string name="blueNoSoport">Bluetooth no soportado</string>
   <string name="datosNoRellenados">Rellene primero la información
requerida en la configuración.</string>
   <string name="BlueNoActivo">Bluetooth no activado</string>
   <string name="noRespuesta">No ha introducido ninguna
respuesta</string>
   <string name="errorConecServ">Error al conectar con el
servidor</string>
   <string name="errorXML">Error en xml</string>
   <string name="msgEnviadoExito">RESPUESTA ENVIADA CON
ÉXITO!!</string>
    </resources>
```

colores.xml:

Los colores están expresados en hexadecimal, y extraídos realizando pruebas.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="titulo">#4698E6</color>
   <color name="fondo">#000000</color>
   <color name="texto">#4698E6</color>
   <color name="blanco">#FFFFFF</color>
   <color name="fondo_boton">#3CC29F</color>
   <color name="letra">#000FC7</color>
</resources>
```

3.5.2. Clase responde

Es la clase principal, la que se ejecuta al iniciar la aplicación. Proporciona al usuario un menú sobre el que puede seleccionar cualquiera de las opciones que ofrece. Las opciones que se ofrecen en el menú son las mostradas en la *Figura 25*. Esta clase se hereda de la clase *Activity* (su característica principal es que permite interactuar con el usuario), e implementa la interfaz *OnClickListener* (detectar pulsaciones en un *view*).

Variables:

Las variables declaradas para la ejecución de dicha clase son:

- *private BluetoothAdapter local_blue*: objeto que se corresponde con el adaptador *bluetooth* local. Es privada.
- *private SharedPreferences conf*: clase privada que permite guardar y obtener parejas de *key*-valor, y que permanecen con el valor guardado incluso cuando se cierra la aplicación. Utilizada para las opciones de configuración.
- *private static final int ENABLE_BT*: *constant* de tipo privado inicializado a *1*, y empleado como código de solicitud por si es necesario activar *Bluetooth*.

Métodos:

Se han implementado los métodos que se detallan a continuación para efectuar la ejecución de la aplicación de forma adecuada.

- *@Override public void onCreate(Bundle savedInstanceState)*:

  Método ejecutado cuando se crea la clase. Este método sobrescribe al del padre. Comienza llamando al método *onCreate()* de la clase padre, para cargar el estado anterior de la actividad si se había guardado. A continuación se cargan los gráficos llamando al método *setContentView()* especificando el gráfico que se cargará (en este caso *main.xml*, por lo que se le pasa la ruta donde reside; es decir, *R.layout.main*).
  Cuando se ha creado la parte gráfica, a cada botón se le ha asignado un identificador de forma que después, desde el código, se puedan obtener estos componentes. Por lo tanto, se obtienen los botones mediante *findViewById()* pasándole el identificador de cada uno, y se suscriben al evento de pulsación por parte del usuario mediante el método *setOnClickListener()* sobre el objeto que se quiere suscribir, es decir, objeto.*setOnClickListener(this)*.
  Suscritos los botones a eventos de pulsación, se obtiene el adaptador *bluetooth* local mediante *BluetoothAdapter.getDefaultAdapter()*. Si es nulo significa que no se ha podido obtener, por lo que se le indica al usuario.
  Por último, se obtiene la configuración mediante *PreferenceManager.getDefaultSharedPreferences(this)*, guardando el resultado en la variable *config*.

- *public void onClick(View v)*:

  Cuando el usuario pulse sobre alguno de los controles que se han suscrito a este "escuchador", se ejecuta este método, y que tiene como parámetro de entrada el *View* sobre el que se ha realizado el clic.
  En primer lugar se comprueba a qué *View* se corresponde, comparando el identificador del *View* que se ha pulsado con el de los diferentes botones suscritos al evento de pulsación.

  - Botón de "Contestar pregunta": se obtienen la información guardada en la configuración sobre el servidor (MAC y nombre que muestra) y sobre el alumno (nombre, apellidos y DNI). Se comprueba el valor del adaptador *bluetooth* local, que en caso de ser nulo significará que no se ha podido obtener y por lo tanto se le notifica al usuario y sale de este método; se comprueba si alguno de los valores del servidor y del alumno no están inicializados, en cuyo caso también se le indica al usuario y se sale de este método; si pasa las condiciones anteriores, se comprueba si el adaptador *Bluetooth* está activado: si es así, se llama al método *empezar()*; en caso contrario, se inicia un *intent* de solicitud al usuario para activar el adaptador *Bluetooth* de forma asíncrona con código de solicitud el especificado por la variable *ENABLE_BT*. Esto significa que cuando se atienda al *intent* creado, se ejecutará el método *onActivityResult()*.
  - Botón de "Configuración": si no hay adaptador *bluetooth* se le indica al usuario y se sale de este método; en caso afirmtivo, se inicia la clase correspondiente creando un *Intent* al que se le especifica la clase que se ejecutará, y posteriormente se inicia este *Intent* pasándolo a *startActivity()*.
  - Botón "About": se abre un diálogo que muestra información sobre la aplicación.
  - Botón "Salir": se llama al método *salir()*.
  - Botón "Si": cuando se llama al método *salir()*, se activan dos botones para pedir confirmación de abandonar la aplicación por parte del usuario. Si se pulsa sobre "Si", finaliza la aplicación mediante *finish()*.
  - Botón "No": llama al método *onCreate()* pasando como argumento *null*, de forma que se restablece la vista anterior.

- *public void empezar()*:

  Método ejecutado para comenzar a contestar preguntas. Es llamada cuando se pulsa sobre el botón "Contestar Pregunta" y el adaptador *bluetooth* está activado, o cuando se envía la solicitud de activación de adaptador *bluetooth* al usuario y responde de forma favorable.
  Crea un *Intent* pasándole la clase *Contestas*, que es la encargada de realizar este proceso, y la ejecuta mediante *startActivity()*.

- *public void onActivityResult(int requestCode, int resultCode, Intent intent)*:

  Método ejecutado cuando vuelve el resultado de la activación de adaptador *bluetooth*. Si el código de solicitud coincide con el que se especificó cuando se creó, y si el usuario ha permitido la activación del adaptador *bluetooth*, espera a que se haya activado realmente para evitar errores, y se llama al método *empezar()*; si no se ha permitido la activación, se le notifica al usuario.

- *public void salir()*:

Método ejecutado cuando se pulsa sobre el botón de “Salir” para abandonar la aplicación. Este método carga otra pantalla que no es más que un diálogo para que el usuario confirme que quiere abandonar la aplicación, y activa la detección de pulsación de los dos botones: “Si” y “No”.

3.5.3. Clase Contestar

Clase que ejecuta la interfaz de usuario donde se carga la respuesta que ofrece el servidor junto con las posibles respuestas, recoge la respuesta que indica el usuario y la prepara para enviarla de vuelta al servidor. Hereda de la clase *Activity*, e implementa la interfaz *OnClickListener* para detectar la pulsación de componentes. Esto lo hace apoyándose en una clase creada específicamente para la comunicación *bluetooth*.

Variables:

En esta clase se ha llevado a cabo la declaración de multitud de variables accesibles desde otras clases, de forma que se tengan todas las variables constantes en una misma clase. Estas variables han sido declaradas como públicas, estáticas y constantes (*final*) para tal fin. Para una mejor comprensión, se irán presentado estas variables por grupos.

1. Variables de tipo *String* que contienen la cadena con el mensaje de error correspondiente.
    a. *ERROR_XML*: cadena para detectar que se ha producido un error al crear la cadena de respuesta a enviar al servidor en formato *xml*. Inicializado a “error”.
    b. *ERROR_SERV_NO_FOUND*: mensaje de error mostrado cuando el dispositivo móvil no encuentra el servidor. Está inicializado con el valor "El servidor % no se ha encontrado. Compruebe que se encuentra dentro del alcance y que el servidor se encuentra operativo". Cuando se va a emplear, se sustituye el símbolo “%” por el servidor.
    c. *ERROR_CREAR_SOCKET*: mensaje de error que se muestra cuando se produce un error al crear el socket. Inicializado a "Error al crear el canal de escucha. Vuelva a intentarlo".
    d. *ERROR_TIMEOUT*: mensaje de error para cuando se cumple el *timeout*. Se inicializa a "Se ha pasado el timeout".
    e. *ERROR_SERV_NO_ENCON*: mensaje de error que se muestra cuando no se encuentra el servidor. Es el mismo que *ERROR_SERV_NO_FOUND* salvo que se omite el símbolo “%”.
2. Variables con los identificadores de servicio para la pregunta y la respuesta. Son de tipo *UUID*.
    a. *uuidResp*: identificador de servicio para indicar que se quiere enviar la respuesta. Inicializado al valor "12345678-1234-1234-1234-123456789012".

   b. *uuidPreg*: identificador de servicio para indicar al servidor que se quiere recibir la respuesta. Inicializado a "12345678-1234-1234-1234-123456789000".

3. Variables enteras que indican el modo en el que se quiere trabajar.
   a. *MODO_PREGUNTA*: inicializado a 3, se emplea para indicar que se está trabajando en modo pregunta, es decir, que se quiere recibir la pregunta por parte del servidor.
   b. *MODO_RESPUESTA*: inicializado a 4, se emplea para indicar que se quiere enviar la respuesta de vuelta al servidor.
4. Variables que representan los *keys* empleados en el objeto *handler*, es decir, objeto que se utiliza para enviar mensajes entre procesos. Son de tipo *String*.
   a. *H_PREGUNTA*: inicializado a "pregunta", se utiliza en el objeto *handler* para obtener el valor asociado a este *key*.
   b. *H_RESPUESTA*: inicializado a "respuesta", se emplea para obtener el valor asociado a este *key* en el objeto *handler*.
   c. *H_TOAST*: inicializado a "toast", se emplea para obtener el valor asociado a esta *key*.
5. Variables de tipo entero (*int*) que se emplea para saber quién ha enviado el mensaje entre procesos a través del *handler*.
   a. *W_ES_PREGUNTA*: inicializado a 10.
   b. *W_ES_RESPUESTA*: inicializado a 11.
   c. *W_ES_TOAST*: inicializado a 12.
6. Variables de tipo *String* empleadas como *keys* para obtener los valores de configuración.
   a. *keyServName*: inicializado a "nombre del servidor".
   b. *keyServAddr*: inicializado a "direccion del servidor".
   c. *keyAlumnoName*: inicializado a "nombre_alumno".
   d. *keyAlumnoApell*: inicializado a "apellido_alumno".
   e. *keyAlumnoDNI*: inicializado a "dni_alumno".
7. Variables de tipo entero que indican los códigos de retorno del *intent*.
   a. *DESCUBRIR*: inicializado a 2, código de solicitud empleado en el *intent* para solicitar poner dispositivo descubrible.
   b. *TiempoDiscoverable*: tiempo durante el que el dispositivo móvil estará descubrible. Inicializado a 120 (segundos).
8. Otras variables.
   a. *int TIMEOUT*: timeout que indica el tiempo a esperar cuando se está esperando la conexión del servidor.
   b. *String ACK*: cadena de asentimiento cuando se recibe la respuesta por parte del servidor.
   c. *xmlFile*: cadena en formato *xml* a enviar al servidor.
   d. *separador*: carácter de separación para obtener la información de la pregunta y posibles respuestas de la cadena enviada por el servidor.
9. Variables para crear la cadena de respuesta en formato *xml* con los datos necesario, cadena que será enviada al servidor.

```
private final String cabecera = "<?xml version=\"1.0\" encoding=\"UTF-
8\"?>";
    private final String body_in = "<datos>";
          private final String nodo1_in = "<info_alumno>";
                private final String nodo1_1_in = "<addr>";
                private final String nodo1_1_out = "</addr>";
                private final String nodo1_2_in = "<nombre_alumn>";
                private final String nodo1_2_out = "</nombre_alumn>";
                private final String nodo1_3_in = "<apellido_alumn>";
                private final String nodo1_3_out = "</apellido_alumn>";
                private final String nodo1_4_in = "<dni_alumno>";
                private final String nodo1_4_out = "</dni_alumno>";
          private final String nodo1_out = "</info_alumno>";
          private final String nodo2_in = "<info_resp>";
                private final String nodo2_1_in = "<pregunta>";
                private final String nodo2_1_out = "</pregunta>";
                private final String nodo2_2_in = "<respuesta>";
                private final String nodo2_2_out = "</respuesta>";
          private final String nodo2_out = "</info_resp>";
    private final String body_out = "</datos>";
```

Para el desarrollo de esta clase, se han definido una serie de variables para controlar la ejecución.

- *private SharedPreferences conf*: clase privada que permite guardar y obtener parejas de *key*-valor, y que permanecen con el valor guardado incluso cuando se cierra la aplicación. Utilizada para las opciones de configuración.
- *private TextView pregunta*: es el componente que muestra por pantalla el enunciado de la pregunta a contestar.
- *private EditText respEscrita*: componente de inserción de texto por si la pregunta pide introducir la respuesta.
- *private CheckBox respSelec*: cada una de las posibles respuestas se representarán un componente de este tipo.
- *private boolean serv_found*: booleano que indica si se ha encontrado el servidor.
- *private String nombre_serv*: cadena que guardará el nombre de adaptador *bluetooth* del servidor.
- *private String MAC_serv*: cadena que guardará la dirección MAC del adaptador *Bluetooth* del servidor.
- *private Boolean hayRespuesta*: booleano que indica si ya se ha seleccionado una pregunta.
- *String numPreg*: guardará el valor del número de pregunta cargada.
- *String respuesta*: guardará la respuesta seleccionada o escrita.
- *private BluetoothAdapter local_adapter*: adaptador *bluetooth* local.
- *private BluetoothDevice disp_serv*: contendrá un dispositivo *Bluetooth*.
- *private int MODO*: guardará el modo de ejecución de la aplicación.
- *Comunicacion comun*: objeto de la clase *Comunicacion* que contiene todos los métodos necesarios para realizar una comunicación *bluetooth*, y además en segundo plano.

Hay una variable que ejecuta su método en la declaración:

- *private final Handler handle = new Handler(){}*:

  Este objeto se utiliza para enviar mensajes entre procesos. Implementa el método *public void handleMessage(Message msg)*, que es mediante el que se reciben los datos de la clase *comunicacion*. Se comprueba de qué es el mensaje: si es pregunta, se obtiene indicando el *key* correspondiente, y se llama al método *cargarPregunta()*; si es de tipo respuesta, contendrá el asentimiento enviado por el servidor al mensaje de respuestas, por lo que se llama al método *Enviado()*; si es del tipo toast, se muestran los datos obtenidos en un toast.

Métodos:

Los métodos desarrollados para llevar a cabo la labor de esta clase son los que se detallan a continuación.

- *@Override protected void onCreate(Bundle b)*:

  Método ejecutado cuando se crea la clase actual. Este método sobrescribe al del padre. En primer lugar ejecuta el método *onCreate()* de la clase padre, para cargar el estado anterior de la actividad si se había guardado. A continuación se cargan los gráficos empleando el método *setContentView()* especificando el gráfico que se cargará (*R.layout.contesta1*). A continuación se obtiene el *TextView* donde se cargará el enunciado de la pregunta, y el botón para volver atrás ,suscribiendo a este último de detección de clic. Se obtienen también la configuración por defecto y el adaptador local *bluetooth*. Se indica que se está en modo pregunta (se quiere recibir la pregunta a contestar), y se obtiene el nombre y la dirección del servidor del fichero de configuración. Por último se llama al método *ponerDescubrible()*.

- *public void onClick(View v)*:

  Cuando el usuario pulsa alguno de los controles que se han suscrito a este evento, se ejecuta este método que contiene el componente o *View* que ha provocado la ejecución de este método.
  A través del identificador se averigua el componente que ha lanzado el evento.
  - Botón "atrás": botón para abandonar esta ventana y volver a la principal. Se llama al método *finalizar()*.
  - Botón de "Enviar": si no se ha seleccionado una respuesta, y no hay texto escrito en el control para introducir texto, se notifica al usuario que no se ha seleccionado la respuesta. En caso de que haya texto en el control de introducir respuesta, se obtiene guardándola en la variable *respuesta*, se muestra al usuario la respuesta que se va a enviar, se indica que ya hay

respuesta seleccionada poniendo el *boolean hayRespuesta* a *true*, y se llama al método *prepararEnvio()*.

- Si el identificador está entre 1 y 10 quiere decir que se ha seleccionado una respuesta de la lista, por lo que, si no hay respuesta seleccionada, se obtiene el *checkbox* que se ha pulsado, y de éste el texto de la respuesta; se indica que hay respuesta seleccionada (*hayRespuesta = true*), y se llama al método *prepararEnvio()*.
- Si se ha pulsado sobre el botón "reintentar", se desactiva y se pone no visible. Si se está esperando pregunta, al igual que si quiere enviar la respuesta, se llama al método *ponerDescubrible()*.

- *public void cargarPregunta(String datos)*:

  Método ejecutado para cargar la pregunta recibida del servidor por pantalla. En primer lugar se comprueba que la cadena recibida no sea *null*. Si la cadena pasada (*datos*) es igual al error de timeout, en el *View* de cargar el enunciado de la pregunta se carga el error *ERROR_SERV_NO_ENCON*, se obtiene el botón de reintentar, se activa, se pone visible y se suscribe al evento de detectar pulsación.
  Si no se han cumplido ninguna de las dos cosas anteriores quiere decir que la cadena con los datos de la pregunta a responder ha llegado bien. Se obtienen los datos de la cadena puesto que vienen separados por un carácter de separación. Así, se obtiene el número de respuestas, el número de la pregunta cargada, se carga el enunciado de la pregunta en el *TextView*, y se obtiene el elemento *LinearLayout* para representar la información de forma más adecuada.
  Si el número de respuestas es mayor a cero, se crea un *TableLayout* para cargar las respuestas una debajo de otra. Se obtiene la respuesta de la cadena, se inicializa un *CheckBox* y se le asigna el texto. Se crea un elemento *TableRow* al que se añade el presente elemento, y a continuación se añade a la *TableLayout* la *TableRow*. Se suscribe al evento de detectar pulsación, y se actualizan los punteros. Este proceso se hace mientras quedan respuestas por obtener de la cadena. Cuando se han obtenido todas, se añade el *TableLayout* al *LinearLayout*.
  En caso de que el número de respuestas sea *0*, se crea un objeto de tipo *EditText*, se inicializa y se añade al *LinearLayout*. A continuación se obtiene el botón de enviar datos, se activa, se pone visible y se suscribe al evento de detección de pulsación.

- *public void prepararEnvio()*:

  Método llamado para preparar el envío de la respuesta al servidor. Se crea la cadena con los datos de respuesta al servidor en formato *xml* mediante la función *crearXmlFile()*. Si se produce error, se realizan tres intentos. Cuando

se crea, se pone en modo pregunta para enviar la respuesta al servidor, y se llama al método *ponerDescubrible()*.

- *public String crearXmlFile(String respuesta)*:

  Método que compone la cadena de respuesta al servidor en format *xml*. Se obtiene la MAC del adaptador *bluetooth* local, se obtienen los datos del usuario de la configuración, y se forma la cadena en formato *xml* con todos estos datos.

- *public void Enviado(String datos)*:

  Método que comprueba si, después de enviar la respuesta al servidor, se recibe el asentimiento correspondiente. En caso de ser así, se notifica el éxito, y se llama al método *finalizar()* con un texto. En caso de no ser así, se notifica el error, y se activa el botón de reintentar.

- *public void finalizar(String error)*:

  Se llama a este método para liberar recursos antes de abandonar la clase en ejecución. En primer lugar se cancela la búsqueda de dispositivos, si estaba activa, se des-registra de eventos, se cierra el objeto de la clase de comunicación (mediante una llamada al métodos *cancel()*), y se finaliza la clase llamando a *finish()*.

- *public void ponerDescubrible()*:

  Método para poner el adaptador *bluetooth* del dispositivo móvil en modo descubrible. Se crea el *intent*, se añade la información del número de segundos que se quiere poner descubrible, y se lanza de manera asíncrona. Este cuadro de diálogo solicita al usuario permiso para ponerse descubrible.

- *public void onActivityResult(int requestCode, int resultCode, Intent intent)*:

  Método que atiende las respuestas a *intents* lanzados de manera asíncrona. Si el código de consulta es *DESCUBRIR*, y el código devuelto es el tiempo descubrible, se ha aceptado, por lo que se suscribe al evento de detectar cambios de visibilidad. Si está en modo pregunta, carga en el *TextView* del enunciado de la pregunta "Cargando pregunta…", se crea un nuevo objeto de la clase *comunicacion* para enviar los datos a través de *bluetooth*, y se indica que inicie (mediante *Start()*). Si no se acepta, se desmarca la respuesta seleccionada.

- *public void comprobarServidor()*:

Método para comprobar servidor. No se emplea, como se ha explicado al comienzo de este capítulo, porque se pierde demasiado tiempo. Este método obtiene los datos del servidor de la configuración, y se suscribe a los eventos de dispositivo encontrado y búsqueda finalizada. Para prevenir malos funcionamientos, si está descubriendo se cancela la búsqueda de dispositivos, y se vuelve a iniciar.

3.5.4. Clase Comunicacion

Clase implementada para realizar las funciones de conexión e intercambio de datos con el servidor a través de *bluetooth*. Para evitar que la aplicación se quede congelada, se hace mediante un hilo paralelo, por lo que la clase hereda de *Thread*.

Variables:

Las variables empleadas para la ejecución de esta clase son privadas.

- *BluetoothServerSocket socket*: esperará peticiones de conexión. Se inicializará en el constructor.
- *String MAC_serv*: almacenará la MAC del servidor para diferenciar entre los dispositivos que se conecten y solo permitir la conexión con el servidor.
- *BluetoothAdapter local_blue*: adaptador *bluetooth* local.
- *int MODO*: guardará un entero que indicará si se encuentra en modo de cargar pregunta, o en modo de enviar respuesta.
- *BluetoothSocket cliente*: cuando *socket* acepte una conexión, habrá que guardarlo en un objeto de este tipo.
- *Handler handle*: objeto para intercambiar datos entre clases.
- *int toast*: indicar el los mensajes del método *devuelve()* que se trata de un mensaje toast.
- *Boolean salir*: valor utilizado para indicar que es quiere salir de la clase.

Constructor:

En el constructor se lleva a cabo la inicialización de variables. Es de la forma *public Comunicacion(BluetoothAdapter local, String MAC, int MODO, Handler h)*.

Se guarda el adaptador local, la MAC del servidor, el MODO en el que se empleará esta clase, y el objeto *Handler* para comunicarse con otras clases. Dependiendo del modo de ejecución (si se quiere la pregunta o se quiere enviar la respuesta), se cargará un identificador u otro. Se inicializa la variable *socket* mediante *local_blue.listenUsingRfcommWithServiceRecord()*. Si se produce error, se ejecuta el método *devuelve()* con el mensaje de error indicado por *ERROR_CREAR_SOCKET*, y el modo en el que se ha producido el error (pregunta o respuesta). Se hace un bucle del que no se saldrá hasta que se encuentre el servidor o se cumpla el timeout. Para poner el dispositivo móvil a la escucha, *socket* ejecuta su método *accpet()*, pasando como valor de timeout el indicado por la variable estática correspondiente. Cuando se cumpla la

temporización saltará una excepción; en ésta se incrementa una variable. Cuando se ha cumplido el tiempo 6 veces, se sale del método.

Si se conecta alguien, se comprueba que es el servidor comparando las direcciones MAC. En caso positivo se pone la variable *esServidor* a *true*; si no es el servidor se cierra la conexión, y se vuelve a la escucha. Al salir del bucle por haber encontrado el servidor, se comprueba si *cliente* está inicializado, en cuyo caso se obtienen los *streams* de entrada y salida, y se reserva espacio para la respuesta. Si se está trabajando en modo pregunta, se espera recibir la cadena con la pregunta y las respuestas, y se envía el asentimiento. Si está en modo respuesta, se envía la respuesta al servidor y se espera recibir consentimiento. A continuación se cierra el cliente, el socket, y en el *String datos* se guarda el array de bytes recibido. Por último, llama al método *devuelve()* con los datos y el modo de ejecución.

Métodos:

Se han implementado dos métodos para controlar la conexión, y otro para poder enviar mensajes entre clases.

- *public void run()*:

  Método que se ejecuta en segundo plano, y que realiza la recepción de la pregunta o el envío de la respuesta.

  En primer lugar, inicializa una serie de variables a *null*. Comprueba el modo de ejecución, ejecutando el método *devuelve()* en cada caso. Si es una pregunta indica que se está cargando la pregunta, y si es en respuesta indica que se está enviando la respuesta. Se hace un bucle del que no se saldrá hasta que se encuentre el servidor o se cumpla el timeout. Para poner el dispositivo móvil a la escucha, *socket* ejecuta su método *accpet()*, pasando como valor de timeout el indicado por la variable estática correspondiente. Cuando se cumpla la temporización saltará una excepción; en ésta se incrementa una variable. Cuando se ha cumplido el tiempo 6 veces, se sale del método.

  Si se conecta alguien, se comprueba que es el servidor comparando las direcciones MAC. En caso positivo se pone la variable *esServidor* a *true*; si no es el servidor se cierra la conexión, y se vuelve a la escucha. Al salir del bucle por haber encontrado el servidor, se comprueba si *cliente* está inicializado, en cuyo caso se obtienen los *streams* de entrada y salida, y se reserva espacio para la respuesta. Si se está trabajando en modo pregunta, se espera recibir la cadena con la pregunta y las respuestas, y se envía el asentimiento. Si está en modo respuesta, se envía la respuesta al servidor y se espera recibir consentimiento. A continuación se cierra el cliente, el socket, y en el *String datos* se guarda el array de bytes recibido. Por último, llama al método *devuelve()* con los datos y el modo de ejecución.

- *public void cancel()*:

  Método que cancela la recepción de la pregunta o el envío de la respuesta.
  Se indica que se quiere salir (*salir = true*), y se cierran el cliente y el socket.

- *private void devuelve(String sstring, int MODO)*:

  Método para enviar mensajes entre procesos.
  Se inicializa *Message msg* a *null*, y se crea un objeto *bundle*. Si el modo pasado el modo pregunta, se indica que es un menaje de pregunta, y se guarda por el *key* declarado (*H_PREGUNTA*). Para los modos respuesta y toast es exactamente igual. Para terminar, en *msg.setData()* se sitúa el *bundle*, y se envía el mensaje mediante *handle.sendMessage(msg)*.

3.5.5. Clase Configuración

Clase para guardar las opciones de configuración tales como la información del alumno y la información del servidor. Hereda de la clase *PreferenceActivity*, e implementa *OnPreferenceChangeListener* y *OnPreferenceClickListener*.

Variables:

Las variables declaradas son:

- *public final int SELECCIONADO*: inicializado a *5*, se utiliza para indicar el código de consulta en el *intent* de seleccionar servidor.
- *Private Preference pServName, private Preference pServAddr, private Preference pAlumnoName, private Preference pAlumnoApe, private Preference pAlumnoDNI*: variables de tipo *Preference* que guardarán los valores configurados.

Métodos:

- *@Override public void onCreate(Bundle bndl)*:

  Método ejecutado cuando se crea la clase actual. Este método sobrescribe al del padre. En primer lugar ejecuta el método *onCreate()* de la clase padre, para cargar el estado anterior de la actividad si se había guardado. A continuación se cargan los gráficos empleando el método *setContentView()* especificando el gráfico que se cargará (*R.layout.configuracion*). Se obtiene la categoría donde incluir el *intent*. Se crea el *Intent* de seleccionar servidor, y se le pone orden, título y comentario, además de suscribirlo para que se detecten pulsaciones. Se añade este elemento a la categoría.

  Para el nombre y dirección del servidor, y para nombre, apellidos y dni del usuario, se obtienen el valor de la configuración, se obtiene la preferencia y se

pone ese valor como comentario. Además, para el caso de la información del usuario, se suscribe al evento valor de preferencia cargado.

- *public boolean onPreferenceChange(Preference preference, Object newValue)*:

  Método que se ejecuta al detectarse un cambio de valor en una preferencia. Se actualiza al comentario mostrado en la preferencia con el nuevo valor.

- *public boolean onPreferenceClick(Preference preference)*:

  Método que se ejecuta cuando se pulsa sobre la opción de seleccionar servidor. Se crea un *Intent*, que ejecuta una nueva clase para seleccionar servidor. Se emplea como código de consulta *SELECCIONADO*.

- *public void onActivityResult (int requestCode, int resultCode, Intent intent)*:

  Método que se ejecuta cuando devuelve datos el *Intent*. Si coincide el código de consulta, y se ha aceptado el formulario, se establece como comentario la nueva información del servidor.

3.5.6. Clase Selec_serv

Clase que se ejecuta cuando se pulsa sobre la opción de seleccionar servidor en la ventana de configuración. Hereda de la clase *Activity*, e implementa *OnClickListener*.

Variables:

Las variables declaradas son:

- *private ArrayAdapter<String> disp_encont*: array de *Strings* que mostrará por pantalla los dispositivos encontrados.
- *private static final int ENABLE_BT*: inicializado a 1, se utiliza como código de consulta para el *Intent* de buscar dispositivos.
- *private Boolean serv_select*: *Boolean* que indica si se ha seleccionado el servidor.
- *private BluetoothAdapter local_blue*: adaptador *bluetooth* local.

Métodos:

Los métodos implementados para la clase son:

- *@Override public void onCreate(Bundle bndl)*:

  Método ejecutado cuando se crea la clase actual. Este método sobrescribe al del padre. En primer lugar ejecuta el método *onCreate()* de la clase padre, para cargar el estado anterior de la actividad si se había guardado. A continuación se cargan los gráficos empleando el método *setContentView()* especificando el

gráfico que se cargará (*R.layout.lista_disp*). Se inicializa el array de dispositivos encontrados, se obtiene el adaptador *bluetooth* local, y se obtienen los botones de búsqueda y "atrás" para suscribirlos al *listener*.

- *public void onClick(View v)*:

  Cuando se pulsa alguno de los dos botones se ejecuta este método. A través de identificador se averigua el botón que se ha pulsado. Si se ha pulsado sobre buscar dispositivos, se ejecuta el método *buscDisp()*. Si se ha pulsado "atrás", llama al método *finalizar()*.

- *public void finalizar()*:

  Método al que se llama para abandonar la ventana liberando recursos.

- *public void buscarDisp()*:

  Método ejecutado para realizar la búsqueda de dispositivos. Si el adaptador *bluetooth* no se encuentra activado, se crea un *intent* para activarlo con código de consulta *ENABLE_BT*. Si está activado, se prepara el *ListView* para nuevos dispositivos, y se suscribe la lista al método de detección de pulsaciones. Se registra a los eventos de descubrir nuevos dispositivos y de finalización de búsqueda, y se inicia la búsqueda de dispositivos.

- *public void onActivityResult (int requestCode, int resultCode, Intent intent)*:

  Método que devuelve si se ha activado el *Bluetooth*. En caso positivo, se llama al método *buscDisp()*. En caso contrario, se llama al método *finalizar()*.

- *private final BroadcastReceiver eventoR = new BroadcastReceiver()*:

  Dentro de este objeto se encuentra el método *public void onReceive(Context context, Intent intent)*, que trata los eventos de dispositivo encontrado y búsqueda finalizada. En el caso de dispositivo encontrado, ese nuevo dispositivo se añade a la lista de dispositivos encontrados (si no se encuentra ya en ella). Cuando finaliza la búsqueda, se muestra al usuario un mensaje por pantalla indicándolo. Si no se han encontrado elementos, se añade el mensaje correspondiente.

- *private OnItemClickListener ClickEnDisp = new OnItemClickListener()*:

  Objeto que detecta si se hace clic en la lista de dispositivos, en cuyo caso es tratado por el método *public void onItemClick(AdapterView<?> arg0, View v, int arg2, long arg3)*. Al seleccionar un dispositivo de la lista, si estaba

descubriendo dispositivos, se cancela la búsqueda. Se obtienen el nombre y la MAC del dispositivo seleccionado. A continuación se llama a los métodos *actualizarInfoServ()* y *finalizar()*.

- *public void actualizarInfoServ(String name, String addr)*:

  Método para actualizar el valor de la información del servidor.

3.5.7. Clase About

Clase que muestra una pequeña información sobre la aplicación. Solo dispone de dos métodos:

- *@Override protected void onCreate(Bundle savedInstanceState)*:

  Método que se ejecuta al crear la clase. En primer lugar se ejecuta el método *onCreate()* del padre, y a continuación se carga la interfaz de usuario con *setContentView(R.layout.About*). Se obtiene el botón aceptar, y se suscribe al evento de pulsación.

- *public void onClick(View v)*:

  Método que se ejecuta al pulsar sobre el botón aceptar, y que provoca la finalización de la clase.

3.5.8. Manifiesto

El manifiesto de *Android* (*AndroidManifiest.xml*) es un archivo que presenta información esencial para la aplicación al sistema *Android*, información que el sistema debe tener antes de que se ejecute cualquier código de aplicación. Para que puedan ejecutarse las clases explicadas, y además se tenga acceso a *Bluetooth*, es necesario indicarlas en este archivo *xml*, por lo que habrá que añadir las líneas que se detallan a continuación.

- Se establece el nombre de la aplicación y el icono que mostrará mediante los atributos de la etiqueta *application*.

```
<application android:icon="@drawable/icono"
android:label="@string/app_name">
```

- Para permitir la ejecución de las aplicaciones habrá que incluir, entre las etiquetas de *application*, la actividad que se ejecutará junto con las opciones deseadas. Así, la aplicación principal es la clase *responde* por lo que habrá que indicarlo añadiendo:

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

```
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

- A la clase *About* queremos que se le aplique un tema de diálogo, así que se le indica en el manifiesto.

```
<activity android:name=".About"
      android:label="@string/about_titulo"
      android:theme="@android:style/Theme.Dialog" >
</activity>
```

- La clase *Contestar*, como se ha explicado, es la que se encarga de cargar la pregunta por pantalla y recoger la respuesta del usuario. Si se gira la pantalla, esta aplicación se inicia de nuevo, por lo que solo queremos que la pantalla esté en sentido vertical.

```
<activity android:name=".Contestar"
      android:label="@string/responder_titulo"
      android:screenOrientation="portrait" >
</activity>
```

- En la clase *Configuracion* no se requiere nada especial, así que se añada sin opciones.

```
<activity android:name=".Configuracion"
      android:label="@string/configuracion_titulo">
</activity>
```

- En la clase *Selec_serv*, al igual que en la clase *Contestar*, no se quiere que se gire la pantalla, así que se establece en vertical.

```
<activity android:name=".Selec_serv"
      android:label="@string/disp_enc_label"
      android:screenOrientation="portrait" >
</activity>
```

- Por último, para poder utilizar las funcionalidades de *bluetooth* del dispositivo, será necesario darle permisos, por lo que fuera de las etiquetas *application* se añade la siguiente línea.

```
<uses-permission
android:name="android.permission.BLUETOOTH_ADMIN">
```

Establecidos estos valores, la aplicación se ejecuta sin problemas de permisos.

CAPÍTULO 4. APLICACIÓN SERVIDOR

Como ya se ha mencionado con anterioridad, la aplicación servidor se ha desarrollado en *C#*, puesto que nos permite realizar aplicaciones de escritorio con una estética atractiva y con buenas prestaciones. Además, al ofrecer una interfaz gráfica al usuario, también hará que le resulte más sencillo su manejo, pues es muy cómodo poder explorar toda la funcionalidad de la aplicación simplemente con el *mouse* haciendo clic.

Esta aplicación no necesita ser instalada para ejecutarse. En la misma carpeta donde se encuentre el ejecutable deben encontrarse las librerías "*InTheHand.Net.Personal.dll*", "*mysql.data.dll*", "*mysql.data.entity.dll*" y "*mysql.visualstudio.dll*". Al ejecutar la aplicación servidor, aparece una ventana como la siguiente:

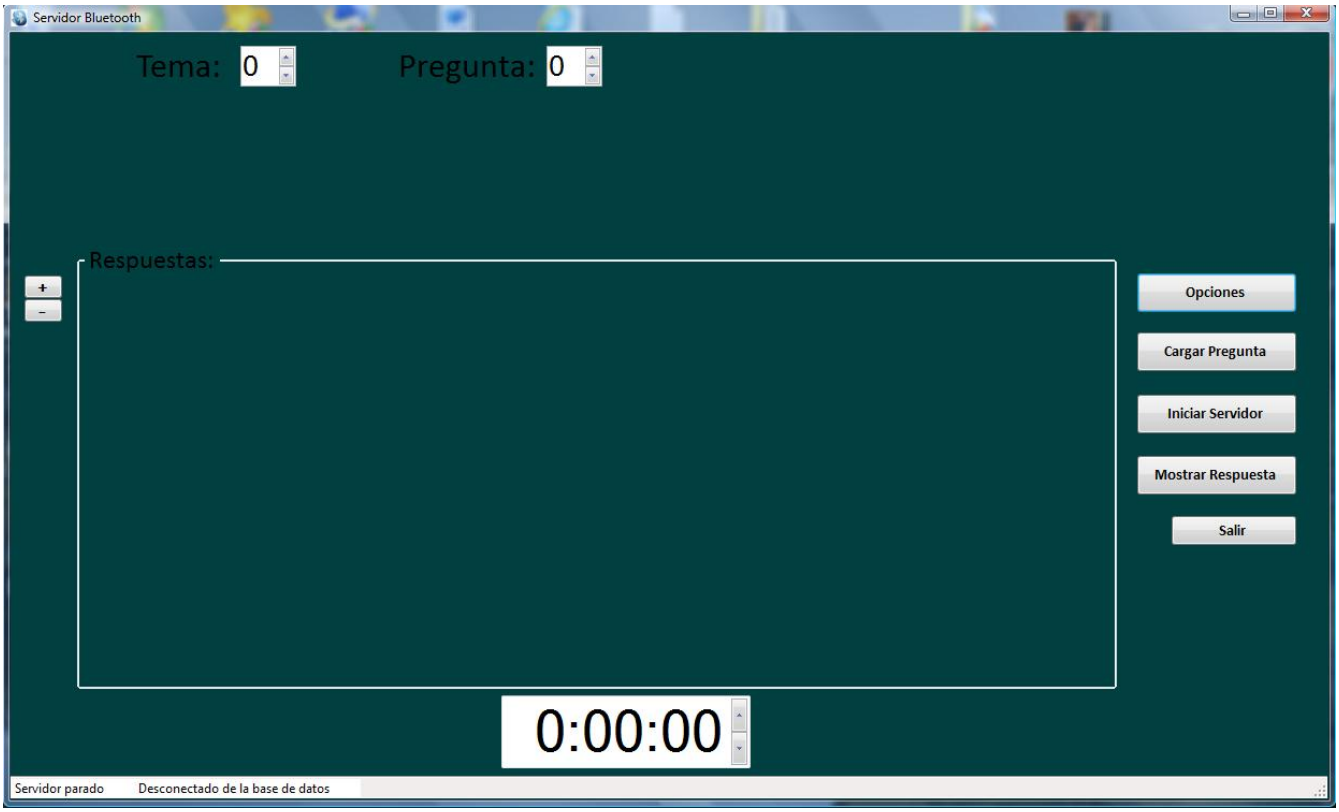

Figura 7: ventana principal aplicación servidor

Es una ventana muy amplia puesto que se ha desarrollado con el objetivo de mostrar las preguntas junto con sus respectivas respuestas por pantalla a través de un proyector, de forma que no solo se carguen en el dispositivo móvil.

El recuadro con el título *Respuestas* que ocupa parte de la aplicación será donde aparezcan las posibles respuestas a la pregunta. La pregunta se visualizará encima de este recuadro, entre los controles de selección de pregunta a través del número de tema y número de pregunta, y el cuadro de *Respuestas*.

En la barra inferior, la barra de estado, se muestra información de estado del servidor. El temporizador que aparece en la parte inferior puede ser inicializado al valor que se precie, de modo que se deje contestar la pregunta durante el periodo de tiempo deseado. La temporización comienza a correr cuando el servidor está iniciado.

En la parte izquierda de la ventana se pueden apreciar un par de pequeños botones con un símbolo más y otro menos. Éstos sirven para aumentar o disminuir el tamaño de la fuente del texto mostrado en pantalla, parar ajustarlo a las necesidades.

En el lado derecho de la ventana se tienen los botones más influyentes sobre la aplicación, son los que ofrecen mayor funcionalidad, pues es donde se inicia o para el servidor, se carga la pregunta, se especifican los detalles de conexión de la base de datos, etc.

Teniéndose localizados los controles dentro de esta ventana, se va a explicar con más detalle el funcionamiento de la aplicación. Se va a ejecutar la aplicación desde cero, es decir, sin disponer de base de datos ni tablas de preguntas ni nada.

Una vez arrancada la aplicación, se dispone de los controles para seleccionar la pregunta del tema especificado, botón de cargar pregunta, iniciar servidor, etc. Pero en realidad no se podrá llevar a cabo ninguna de estas funciones si no se está conectado a la base de datos, pues si no se dispone de conexión con ésta no se podrán obtener las preguntas, el servidor no se podrá ejecutar porque tiene que guardar los clientes procesados, y prácticamente no se podrá hacer nada. Por tanto, el primer paso que se va a realizar al arrancar la aplicación es rellenar la información de la base de datos, pulsando para ello el botón *Opciones*, que hará aparecer una ventana llena de funcionalidades.

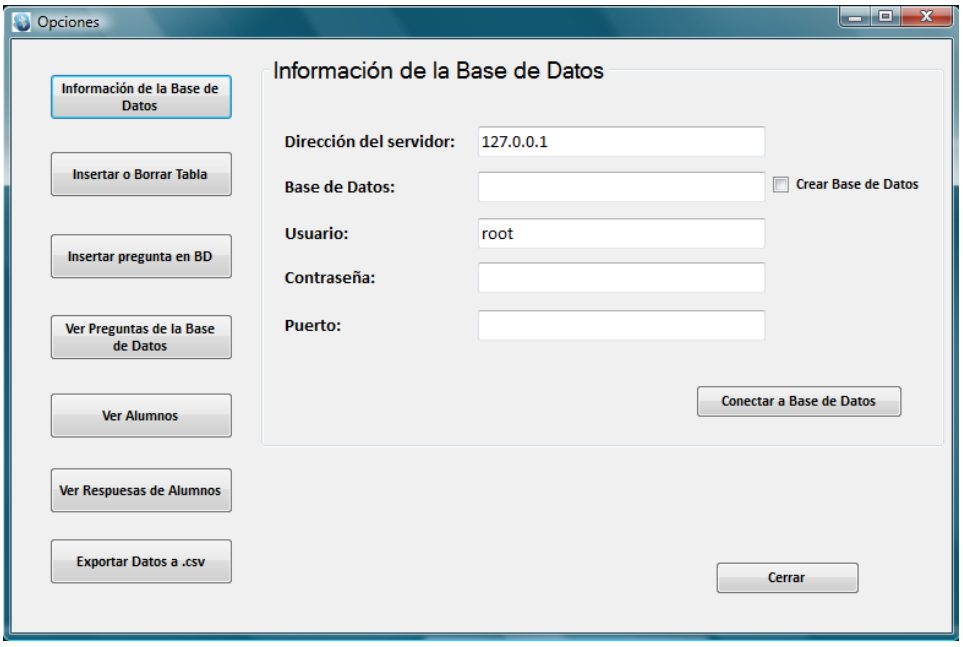

Figura 8: ventana opciones aplicación servidor, formulario conexión a BBDD

A la izquierda de la ventana aparecen una serie de botones con las utilidades necesarias para interactuar con los datos que se van a tratar y la base de datos sin necesidad de emplear otras herramientas, ni tener conocimientos sobre bases de datos. Es más, se aconseja crear las tablas necesarias desde esta interfaz, para evitar un mal funcionamiento de la aplicación.

Este primero formulario que aparece es el de conexión con la base de datos. Rellenando los campos y pulsando sobre el botón conectar se realizará la conexión con la base de datos. Si se conocen los datos del host en el que está la base de datos, pero no se dispone de una, también puede crearla desde este formulario, pues podrá contemplar que si activa el *checkbox* de crear base de datos, el botón se renombra.

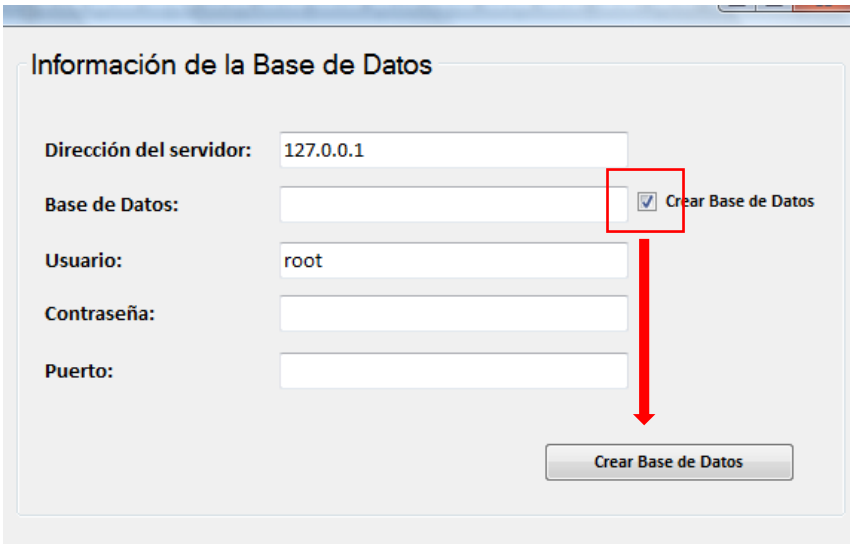

Figura 9: ventana opciones aplicación servidor, crear BBDD

Se introducen los datos correspondientes, y se crea la base de datos. Cuando se genera la base de datos, como no se estaba conectado a ninguna base de datos, la aplicación pregunta que si se quiere conectar a la que se acaba de crear, por lo que se tiene base de datos y se está conectado.

Ya se dispone de base de datos. Para poder ejecutar la aplicación se necesitan tener al menos una pregunta de un tema, por lo que habrá que introducir una tabla de preguntas. Para ello se pulsa sobre el botón correspondiente, y aparece el formulario para insertar, editar o eliminar tabla, tal como el que se muestra en la *Figura 10*.

Para crear la tabla de preguntas simplemente hay se seleccionar el número de tema sobre el que se quiere crear la tabla de preguntas, y el número de respuestas que tendrán las preguntas de la tabla de preguntas. Introducidos los datos deseados, se pulsa sobre el botón “Insertar Tabla”.

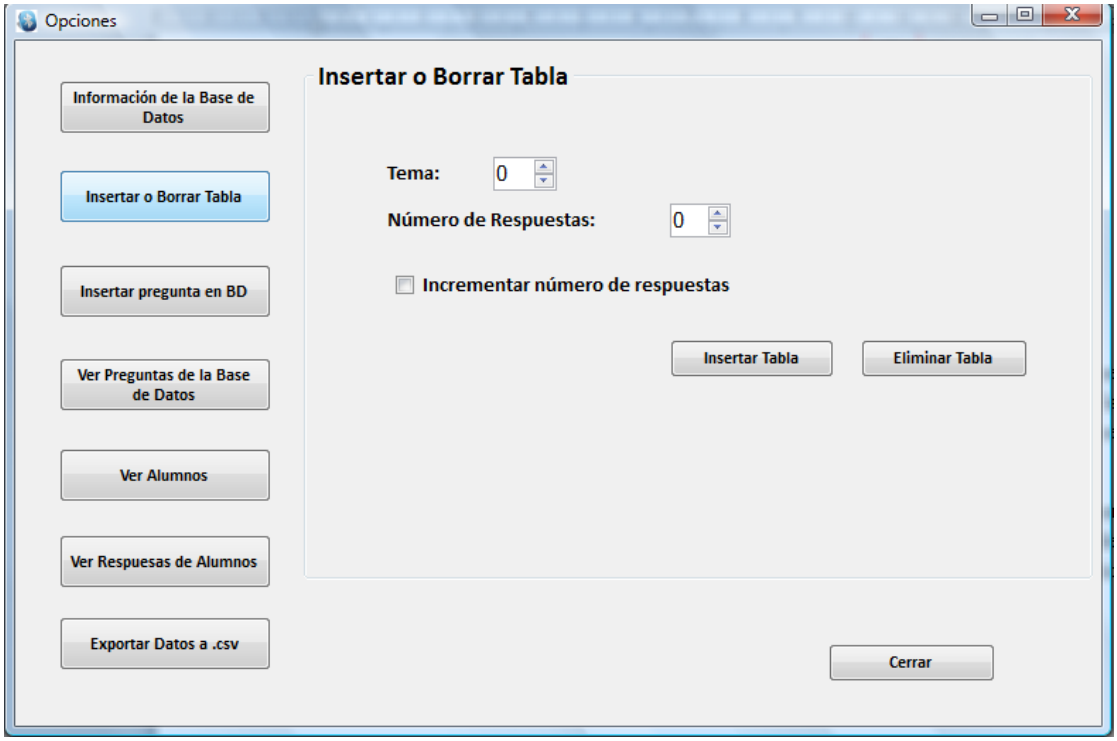

Figura 10: ventana opciones aplicación servidor, insertar tabla de preguntas

Ahora se está en disposición de introducir preguntas para el tema que se ha creado la tabla. Así que se pulsa sobre el botón "Insertar pregunta en BD", apareciendo el formulario de insertar pregunta.

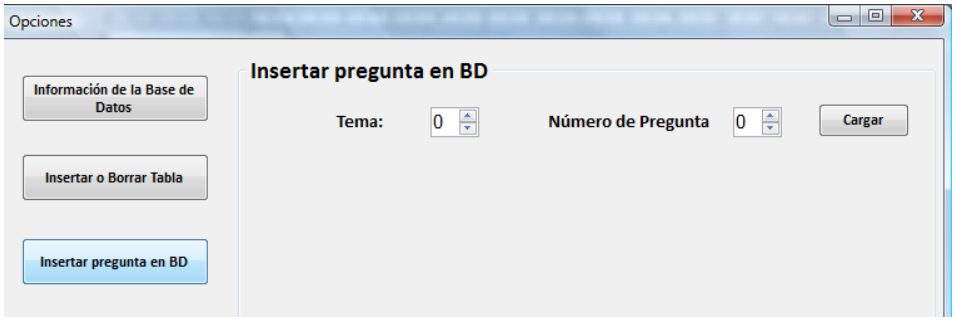

Figura 11: ventana opciones de aplicación servidor, insertar pregunta en BBDD (1)

Como se puede ver, solo aparece para seleccionar el número de tema y número de pregunta que se quiere insertar. Esto es porque, puesto que el tamaño de las tablas será distinto, no se pueden poner un número determinado de preguntas, y en ese caso se generaría confusión. Para evitarlo, el procedimiento es seleccionar la pregunta del tema que se quiere insertar, y pulsar sobre el botón "Cargar". Este botón se encargará de mostrar los controles de respuestas posibles necesarias, por lo que el formulario se rellenará quedando así (*Figura 12)*.

No hay por qué introducir todas las respuestas, solo las que se necesiten, pues se pueden tener dos tipos de preguntas: en la que se tiene que seleccionar la respuesta correcta de la lista de respuestas posibles, o en la que se introduce la respuesta correcta.

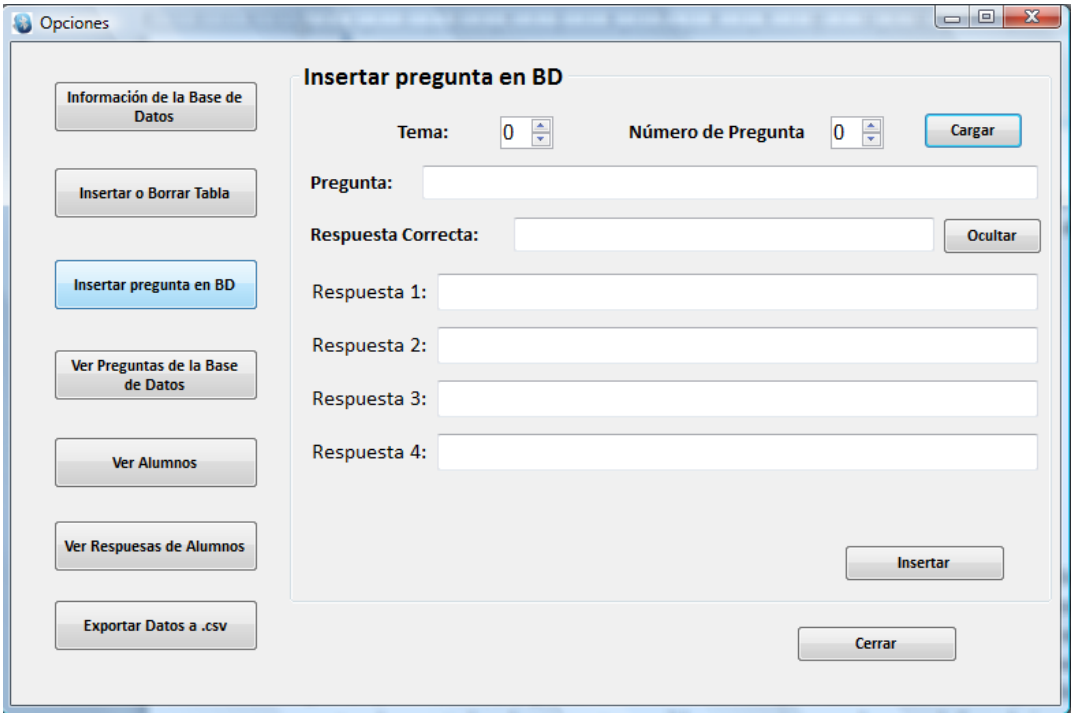

Figura 12: ventana opciones aplicación servidor, insertar pregunta en BBDD (2)

Una vez que se han introducido la pregunta, la respuesta correcta y las posibles respuestas, se pulsa sobre insertar para guardar la pregunta en la base de datos. Si el número de pregunta que se intenta introducir ya se encuentra en la base de datos, la aplicación lo indicará, teniendo que modificar el número de pregunta. Tampoco se insertará si hay alguna posible respuesta introducida y ninguna de ellas coincide con la respuesta correcta. Una funcionalidad añadida este formulario es el botón "Ocultar", que permite ocultar la respuesta correcta, por si no se quiere que el resto de personas vean la respuesta correcta. Al igual que se oculta, se puede mostrar, pues el mismo botón cambia de nombre a "Mostrar" cuando se pulsa para ocultar.

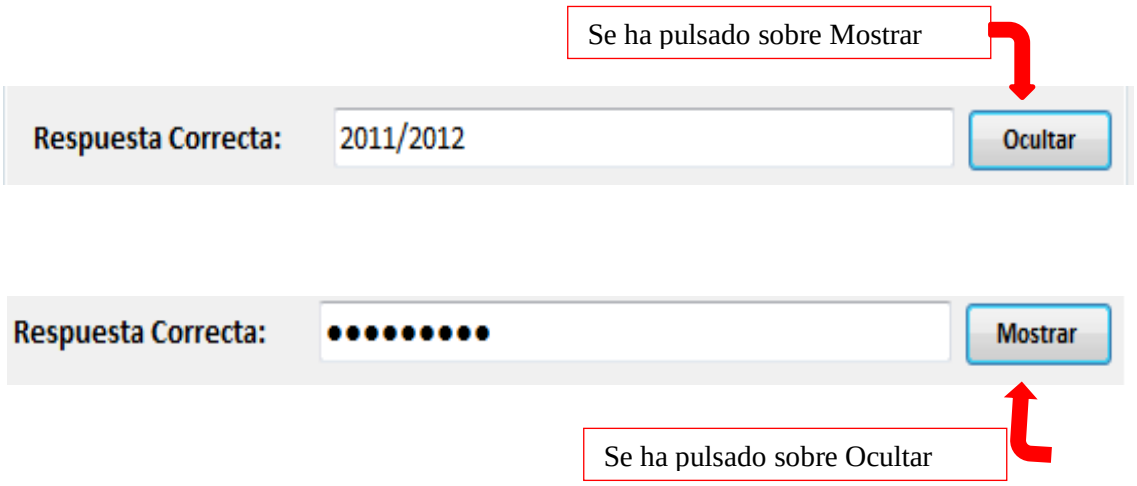

Figura 13: ventana opciones aplicación servidor, botón ocultar/mostrar

Ya se dispone de una pregunta en la base de datos. Se podría iniciar el servidor puesto que ya se dispone de una pregunta, pero se va a seguir explorando esta ventana.

El siguiente botón que aparece es “Ver Preguntas de la Base de Datos”. Y como su propio nombre indica, es para ver las preguntas que tenemos dentro de la base de datos. Al pulsarlo se abre una ventana que nos permitirá seleccionar la pregunta del tema a visualizar, o si se desean ver todas las preguntas para un tema.

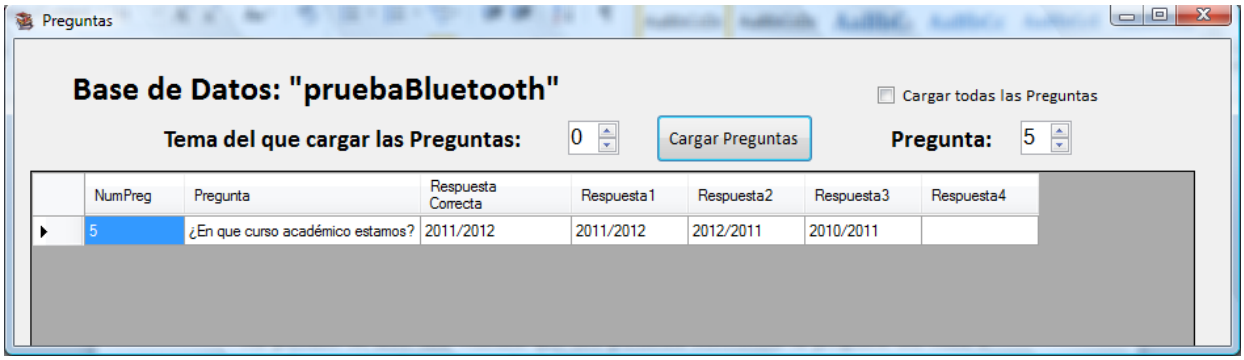

Figura 14: ventana ver preguntas de BBDD, aplicación servidor

Como se puede observar en esta figura (*Figura 14*), se tiene cargada solo la pregunta 5 de la base de datos. Se puede mostrar una pregunta, o se puede activar el *checkbox* que indica que se cargarán todas las preguntas, pero para el número de tema especificado.

Volviendo a la ventana de opciones, el botón que sigue es “Ver Alumnos”. Al pulsarlo, se abrirá una ventana para consultar datos de los alumnos presentes en la base de datos. Dicha ventana será de la forma que se muestra en la siguiente figura (*Figura 15*).

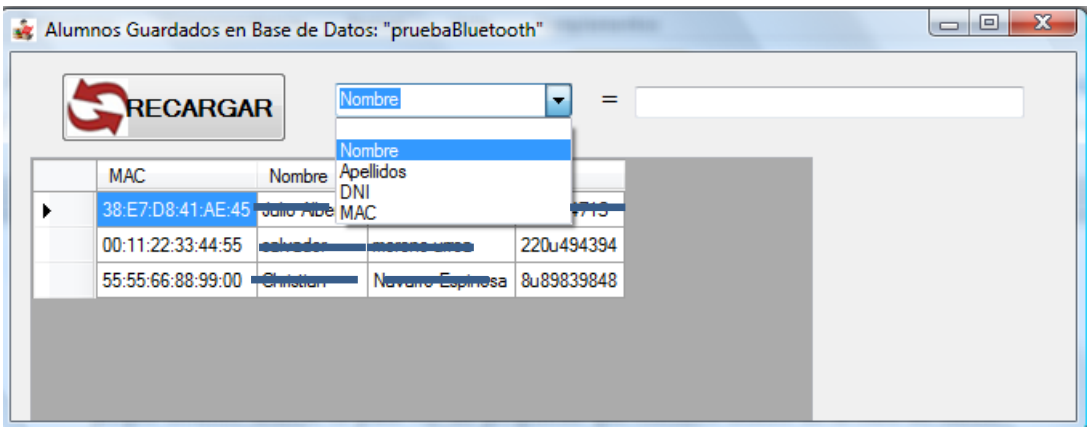

Figura 15: ventana ver alumnos BBDD

Cuando se abre la ventana, automáticamente carga los datos de todos los alumnos presentes en la tabla. Pero se pueden elegir criterios de búsqueda, es decir, en el *comboBox* que se muestra abierto, se escoge la opción de búsqueda, y se introduce en el control de texto de la derecha el criterio de búsqueda. Para que se ejecute la consulta, hay que pulsar sobre el botón RECARGAR. Mediante esta ventana, podemos buscar alumnos por nombre, por apellidos, por DNI, y por MAC del teléfono.

Pulsando sobre el penúltimo botón de la ventana de opciones, sobre el botón “Ver Respuestas Alumnos”, se abre la ventana para ver las respuestas de los alumnos.

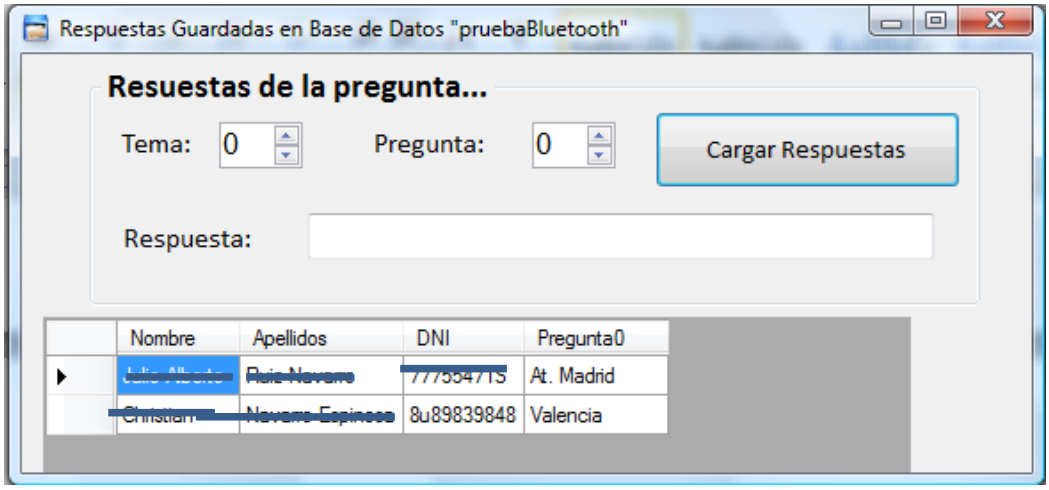

Figura 16: ventana Ver Respuestas de alumnos en aplicación servidor

En esta ventana se pueden consultar las respuestas que los alumnos han contestado a la pregunta indicada del tema especificado. Si se especifica una respuesta en el cuadro de texto, se muestran los alumnos que han contestado a la pregunta especificada del tema indicado con esa respuesta.

Por último solo queda un botón, el botón de exportar datos de la base de datos, y que al pulsarlo se abre la ventana siguiente (*Figura 17*).

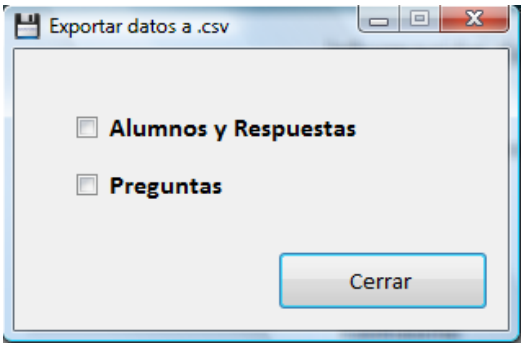

Figura 17: ventana exportar datos. aplicación servidor

Al activar el *checkBox* de cualquiera de los dos, se abrirá un cuadro de diálogo para seleccionar la ruta de guardado, y se cambiará el nombre del botón “Cerrar” a “Exportar”. Si se pulsa sobre exportar, desaparecerán esos *checkBox* y aparecerá una barra de progreso y texto que indicará el progreso de la exportación de datos. De nuevo, el nombre del botón “Exportar” cambiará a “Cancelar”. Si se pulsa sobre este último, se abortará el proceso de exportación de datos de la base de datos.

Antes de cerrar la ventana de opciones y volver a la pantalla principal, se va a comentar una última funcionalidad, la de aumentar el número de posibles respuestas de una tabla de preguntas de un tema. Para ello, volvemos a mostrar el formulario de

insertar tabla, y se activa el *checkBox* de incrementar el número de respuestas. Automáticamente, el formulario modifica los elementos necesarios.

Figura 18: ventana opciones aplicación servidor

A cambiado la posición del segundo campo numérico, se ha modificado el texto que lo acompañaba, el botón de eliminar no está visible, y se ha modificado el tamaño y el texto del botón de "Insertar Tabla" (comparar con *Figura 10*).

Se cierra esta ventana de opciones, volviendo a la ventana principal. Se puede continuar en la ventana principal mientras cualquiera de las otras ventanas esté abierta, no influye en su funcionamiento. Al estar ahora conectados a la base de datos, veamos que ha pasado en la barra de estado.

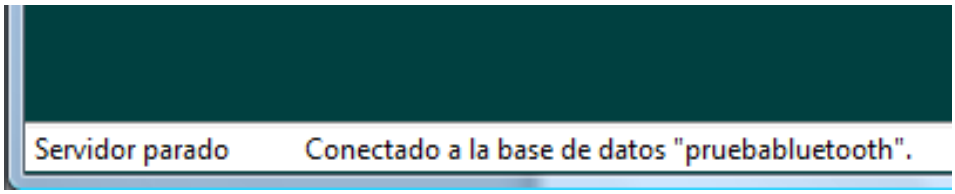

Figura 19: barra de estado actualizada. Aplicación servidor

La barra de estado se ha actualizado, puesto se está conectado a la base de datos. Se actualiza cuando se cierra la ventana de opciones.

Queda cargar la pregunta en pantalla, y adaptarla mediante los botones + y – para que se vean bien. Por tanto, estando en la ventana principal, seleccionamos la pregunta y el número de tema sobre el que se quiere realizar la pregunta, y a continuación se pulsa en el botón "Cargar Pregunta". Al mismo tiempo, iniciar la temporización a 2 minutos.

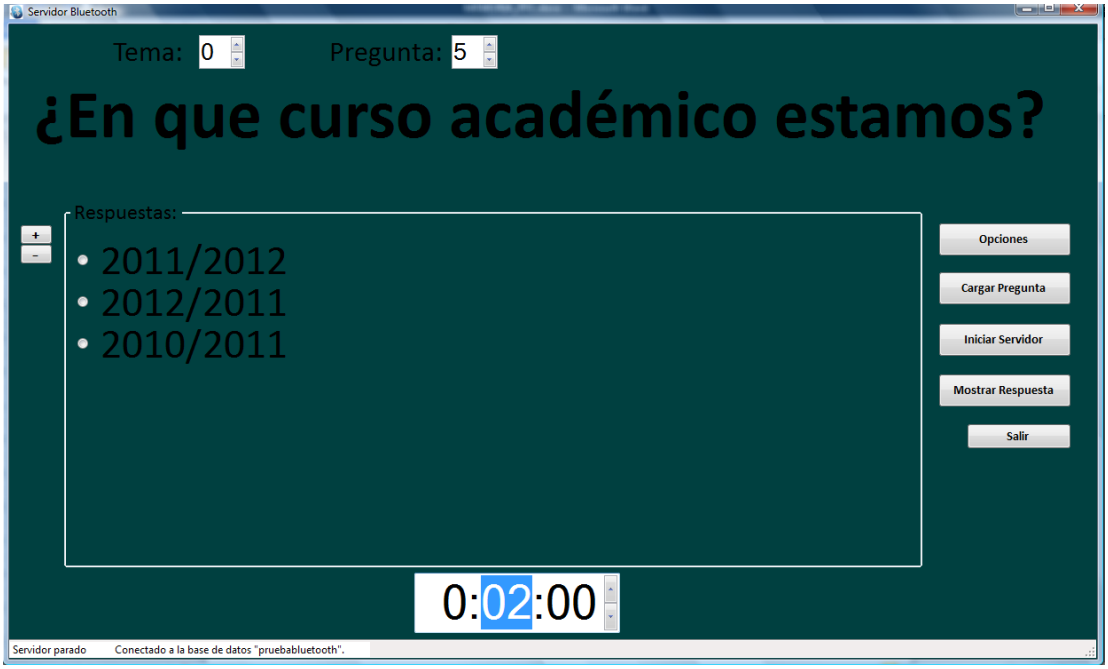

Figura 20: ventana principal preparada para iniciar servidor

Al pulsar sobre iniciar servidor, si todo está bien, comenzará el proceso de servir a los clientes, actualizándose la barra de estado, y decrementándose el contador hasta llegar a 0.

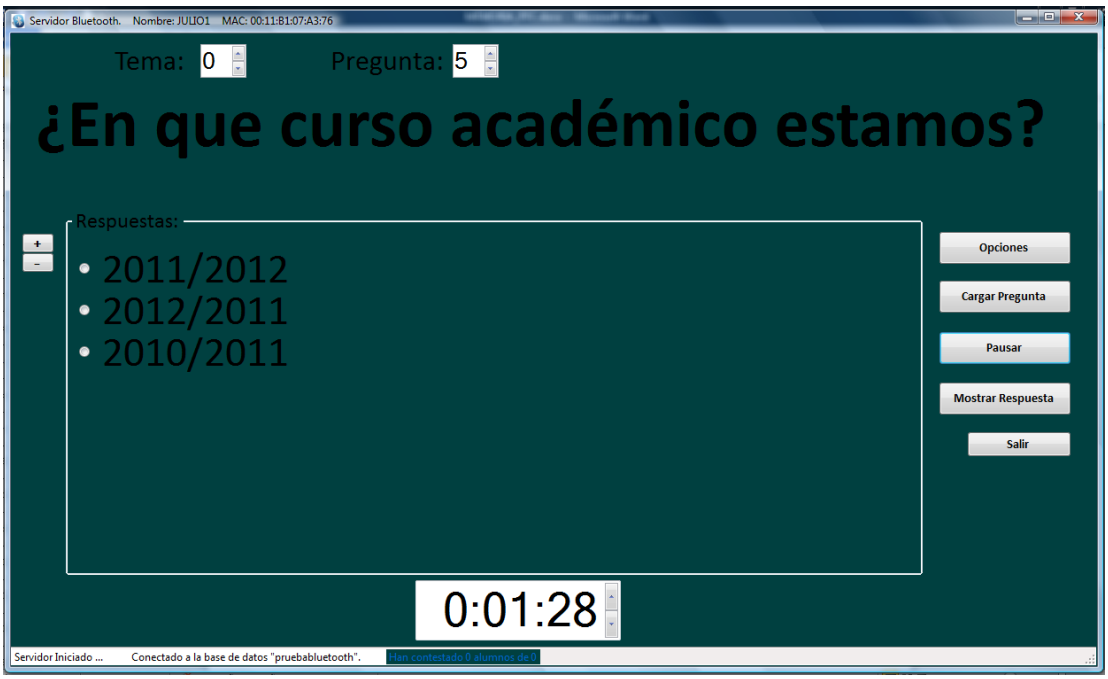

Figura 21: servidor iniciado

Vamos a fijarnos en unos detalles de la ventana. En primer lugar, en el título se ha puesto el nombre del adaptador Bluetooth y la dirección MAC. En la barra de estado se muestra que el servidor está iniciado, y además se lleva la cuenta de las personas que han contestado y las que han solicitado pregunta.

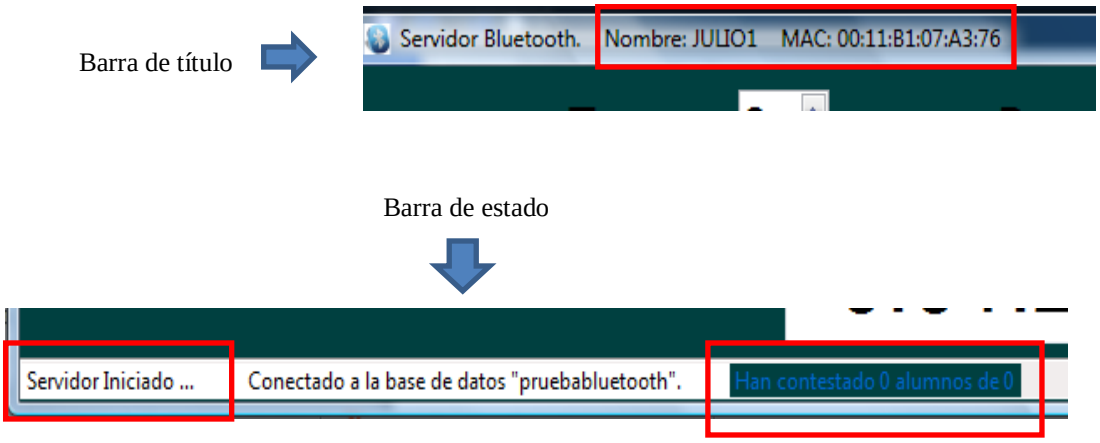

Figura 22: cambios en barra estado ventana principal servidor

Cuando se cumple el tiempo, se notifica al usuario, y en caso de que no hayan contestado todos los que han solicitado pregunta, también se le indica. En este caso, como nadie ha solicitado pregunta y por lo tanto no ha contestado, se estaría en la situación en la que todos los que han solicitado pregunta han contestado, por tanto la ventana resultante es la siguiente.

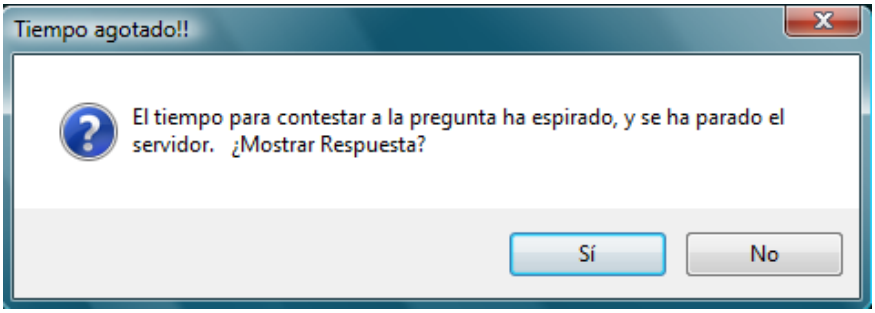

Figura 23: tiempo cumplido en servidor para contestar pregunta

Si se pulsa sobre sí se muestra en pantalla la respuesta correcta. Cuando muestra la respuesta correcta, lo que hace es resaltar la que es correcta haciéndola más grande, y poniéndola en cursiva.

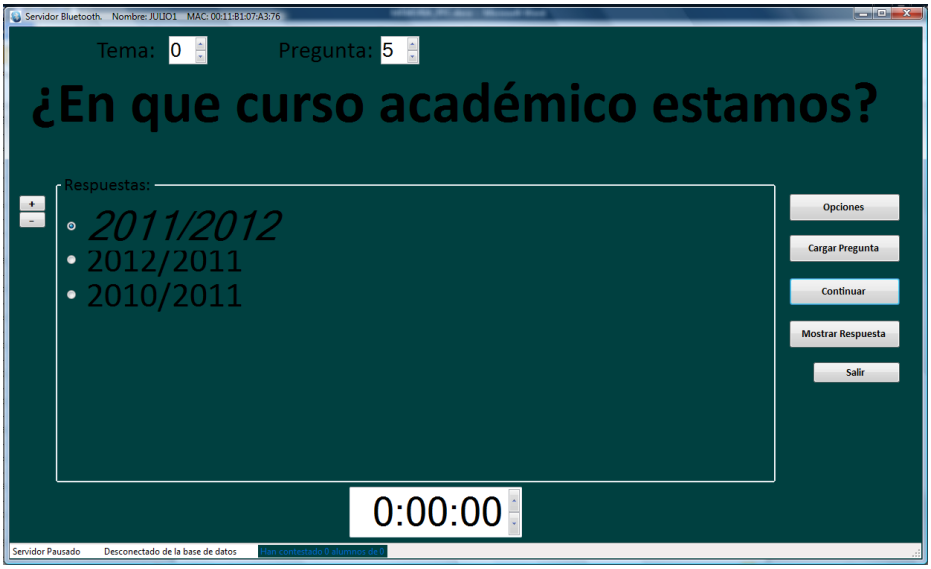

Figura 24: pregunta resaltada en servidor

Si en puesto de establecer una temporización, se inicia el servidor sin esa temporización, y cuando se estime oportuno se pausa el servidor y se pulsa sobre mostrar respuesta correcta, se resalta de igual manera.

Mientras el servidor se encuentra iniciado, un alumno puede contestar las veces que quiera a una pregunta, la base de datos se actualizará, siendo válida solo la última vez que contesta (el resto de veces sobres-escribirá la respuesta previa). Pero en la contabilización de contestados y los que han solicitado preguntas, esta situación sí se tiene prevista, y solo se contabiliza una vez, no cada vez que conteste y solicite pregunta.

CAPÍTULO 5. APLICACIÓN CLIENTE

La aplicación cliente es la aplicación ejecutada en la parte del dispositivo móvil, y que como se ha mencionado en numerosas ocasiones, se encarga de recibir la respuesta y mostrarla por pantalla al usuario, y cuando éste seleccione una respuesta, enviarla al servidor. A pesar de que principalmente realiza esta función, también hace algo más, como se habrá podido ver en el capítulo de desarrollo e implementación de la aplicación cliente (*3.5. Desarrollo de Aplicación Cliente*).

En este punto se va a centrar la atención en el funcionamiento. Cuando se ejecuta la aplicación, aparecen por pantalla una serie de botones en modo de menú. Esta ventana será la ventana principal, y tendrá la apariencia de la siguiente figura (*Figura 25****¡Error!***

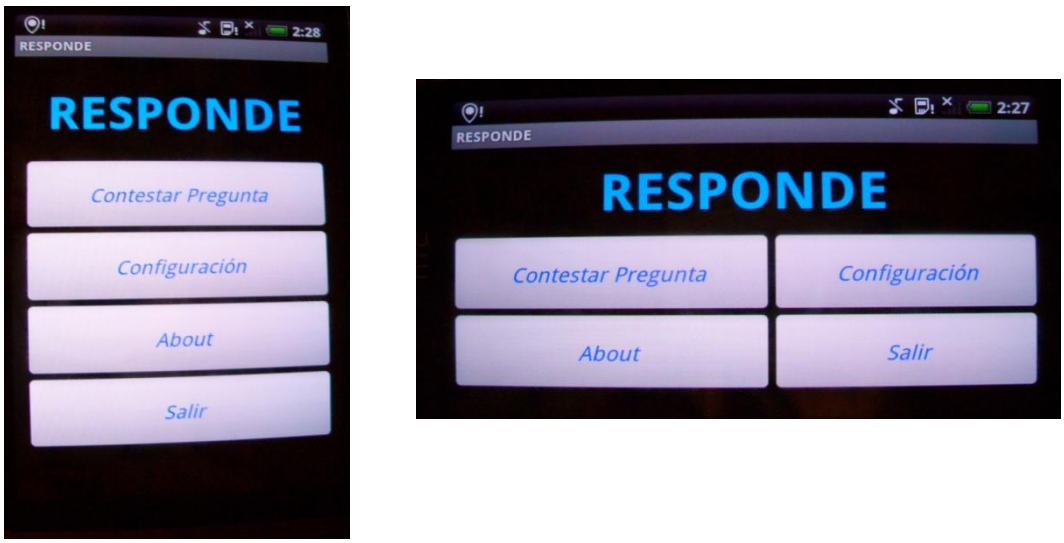

Figura 25: ventana principal de aplicación cliente

No se encuentra el origen de la referencia.).

Aunque aparecen dos ventanas, son la misma solo que, cuando se gira el teléfono y se pone en horizontal, los botones se recolocan para estar accesibles al usuario. Cuando vuelve al estado vertical, la pantalla vuelve a girarse.

El nombre de los botones es bastante intuitivo. Si se pulsa sobre el botón "Contestar Pregunta", se parará a la ventana donde se cargará la pregunta en pantalla (*Figura 26*). Mientras no se cargue por pantalla la pregunta, aparecerá el texto que se muestra en la figura. Siempre se mantiene activo el botón "Atrás" por si en cualquier momento el usuario decide volver a la ventana principal. Es posible que se atienda a este botón debido a que, como se ha explicado en la implementación (*3.5. Desarrollo de Aplicación Cliente*), se ha creado una clase específica para atender la comunicación Bluetooth, y en segundo plano, para que el usuario no pierda el control de la aplicación.

Si llega la información de la pregunta por parte del servidor, se representa por pantalla, sustituyendo el texto mostrado "Cargando Pregunta…" por el del enunciado de

la pregunta. Si hay posibles respuestas entre la que hay que elegir la correcta, aparecerán por pantalla, y cuando se seleccione la deseada, automáticamente se preparará para enviarla. Si por el contrario se trata de una pregunta en la que, como respuesta, hay que introducirla por teclado, entonces aparecerá el botón de enviar, de manera que cuando se haya introducido se pulsará este botón. Si ocurre algún error durante la obtención de la pregunta, o el envío de la respuesta, se notifica por pantalla y se habilita un botón para reintentarlo.

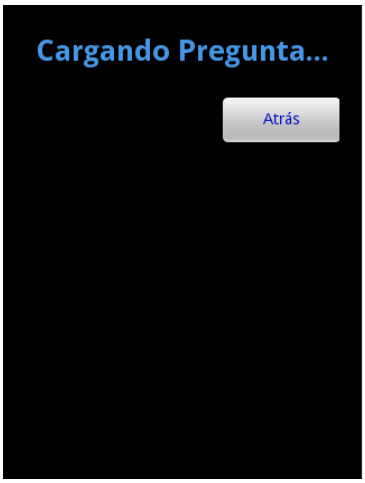

Figura 26: ventana de cargar pregunta de aplicación cliente

Como se puede observar, se ha cargado la pregunta y se ha enviado la respuesta al servidor, y todo ello sin necesidad de rellenar nada sino la respuesta en sí. Esto es gracias a la ventana de configuración creada para almacenar los datos necesarios del servidor y del usuario, de forma que el usuario no tenga que estar continuamente introduciendo los datos y realizando búsquedas para seleccionar al servidor. Con entrar a la ventana de configuración y rellenar los datos del servidor y los propios del usuario, se llevará a cabo una ejecución de la aplicación más dinámica.

La ventana de configuración mencionada es la que aparece a continuación.

Figura 27: ventana configuración aplicación cliente

Los tres últimos campos se corresponden con la información del usuario, es decir, nombre, apellidos y DNI. Para introducir los datos de usuario basta con pulsar sobre el campo correspondiente, y se abrirá un control de texto para introducir el valor.

Se puede observar como los dos primeros campos son de información del servidor, concretamente el nombre y la dirección MAC del adaptador Bluetooth del servidor. Si se intenta seleccionar estos dos campos, al contrario que ocurría con los datos de usuario, se verá que no se pueden editar. Para ello, y por más comodidad, se utiliza el campo nombrado "Seleccionar Servidor", que crea una ventana de búsqueda de dispositivos Bluetooth.

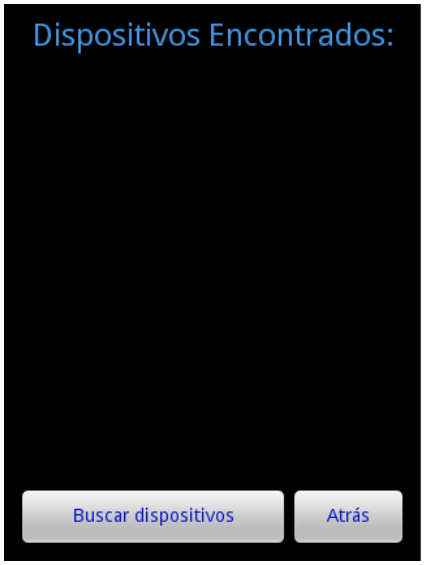

Figura 28: ventana configuración cliente, selección información servidor

Si se pulsa sobre el botón "Atrás", se volverá a la ventana de configuración anterior. Si se pulsa sobre "Buscar Dispositivos", se realizará la búsqueda de dispositivos, mostrándolos por pantalla. Si entre esos dispositivos mostrados por

pantalla se encuentra el servidor, bastará con seleccionarlo para guardar la información del servidor. En esta ventana no se realiza ninguna conexión, solo se produce la búsqueda de dispositivos Bluetooth para seleccionar la información del servidor.

En la ventana principal quedan dos botones sobre los que no se ha mencionado su funcionamiento. Uno es el botón "About", que muestra información sobre la aplicación. El otro es el botón "Salir", y se emplea para abandonar la aplicación. Cuando se pulsa sobre este botón, aparece otra ventana de confirmación (*Figura 29*). En esta última ventana, si se pulsa sobre "SI" se abandona la aplicación; si se pulsa sobre "NO", se vuelve a la ventana principal.

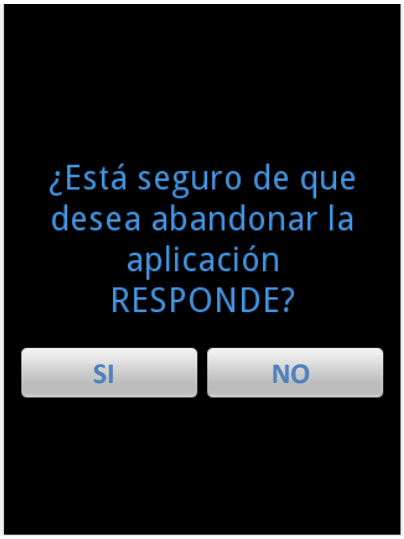

Figura 29: ventana de confirmación de abandono de aplicación cliente

CAPÍTULO 6. CONCLUSIONES Y LÍNEAS FUTURAS

En el presente capítulo se realiza un repaso global al proyecto, presentando las conclusiones finales en comparación con los objetivos marcados, y las posibles líneas de trabajo.

6.1. Conclusiones

Una vez concluidas las tareas que forman este proyecto, es momento de realizar un balance y crítica de los resultados obtenidos. Repasando los objetivos inicialmente marcados, se pueden obtener las siguientes conclusiones:

- *Aprender sobre el sistema operativo Android.*

  A lo largo del desarrollo del proyecto se ha conseguido obtener un amplio conocimiento sobre el sistema operativo *Android*. Gracias a la extensa documentación de Google, se ha podido ir conociendo la arquitectura, funcionamiento, posibilidades ofrecidas, etc. En las fases iniciales es cuando se hace más necesaria esta información, y está totalmente disponible. Mencionar también que en los foros también se dispone información interesante sobre este sistema operativo.
  Tras profundizar más en este sistema operativo, se han conocido aspectos interesantes sobre Android:
  - En primer lugar, la licencia Apache permite que toda persona interesada en el desarrollo de aplicaciones móviles y Smartphone en general pueden estudiar y distribuir el sistema Android, dando también opción al desarrollo privado de aplicaciones; el desarrollador puede elegir qué camino seguir.
  - Los comienzos no son fáciles, pero si se conoce algo del camino se avanza mejor, pues al programar en Java, lenguaje de programación bastante conocido, se comienza a programar con mayor facilidad.
  - La construcción de interfaces de usuario puede hacerse con los elementos y diseños incorporados, mediante código o a través de documentos xml.

- *Indagar en la programación sobre Smartphone*.

  Puesto que en el presente proyecto se utiliza Bluetooth, y el emulador de Android no soporta esta funcionalidad, se ha tenido la oportunidad de probar la aplicación en un Smartphone, y la experiencia es extraordinaria, y más conforme se avanza en el conocimiento de las posibilidades disponibles.

- *Incrementar la formación sobre programación orientada a objetos.*

La programación mediante lenguajes orientados a objetos da otra experiencia sobre programación, y gracias al desarrollo del presente proyecto se ha podido experimentar y disfrutar de ese modo de programación, pues en Android se programa en Java, y la aplicación servidor se ha implementado en C#. Éste último ha superado todas las expectativas con creces, pues ofrece muchas posibilidades en la programación en cuanto a amigabilidad, y por supuesto en cuanto a prestaciones.

- *Aprender un nuevo lenguaje de programación, C#.*

  Como se ha mencionado, C# ha sido una de las grandes experiencias del proyecto, pues descubrí una nueva forma de programar, que aunque dispone de muchas clases y elementos, cada día, al ponerse a programar, se descubren cosas nuevas que fomentan el seguir aprendiendo sobre este lenguaje de programación.

- *Formarse en la utilización de entornos de desarrollo.*

  El entorno de desarrollo de C#, *Visual Studio*, es otro de los motivos importantes que apoyan esta nueva experiencia, pues la verdad abre un abanico de posibilidades en la depuración, haciendo posible la visualización del valor de todas las variables, ejecuciones paso a paso, edición de código y continuar ejecutando sin ser necesario parar la ejecución.

El desarrollo e implementación del presente proyecto me ha brindado la posibilidad de adquirir nuevos conocimientos sobre nuevas herramientas de desarrollo, tecnología, un enfoque con más perspectiva a la hora de emprender nuevas cosas, etc. Así que, a pesar de todo lo pasado y dedicado, se han cumplido los objetivos, y se ha obtenido una gran experiencia.

En cuanto a las dificultades pasadas, residen en la problemática encontrada en la conexión del dispositivo Android con otro dispositivo que no lo era, el PC. Este contratiempo llevó a cambiar totalmente los esquemas, llegando incluso a plantearse optar por otra tecnología inalámbrica.

Otra gran dificultad ha sido la creación de esta memoria del proyecto fin de carrera, pues no resulta demasiado fácil para un ingeniero expresarse como un escritor.

6.2. Líneas de desarrollo Futuras

El sistema desarrollado en el presente proyecto permite ejecutarse en entornos educativos, por lo que una de las líneas sería ampliar el número de dispositivos que puedan acceder al sistema mediante el desarrollo de la aplicación para otras plataformas móviles.

Pensando también en el mundo educativo, se podrían incorporar modos de preguntas más interactivos, como fotografías, vídeos, mapas, etc.

Se podría adaptar el sistema para realizar encuestas, entrevistas, etc.

Y ampliando aún más las expectativas, puesto que se trata de un sistema en que se realiza la conexión, se intercambian los datos, y se desconecta, se podría emplear en zonas turísticas, de modo que el dispositivo móvil indique donde se encuentra, y el servidor le envía información de esa zona.

Por último, puesto que el las *piconets* creadas con Bluetooth son demasiado pequeñas, sería interesante dotar al servidor de varias tarjetas Bluetooth. Gracias a esto, con algunas modificaciones del código sería posible gestionar grupos mayores de dispositivos. Incluso yendo un poco más allá sería interesante ampliar las funcionalidades del servidor para que también contemplase los dispositivos que emplean la interface *Wifi*.

Bibliografía:

1. **http://www.educlick.es/**

   Página oficial de *Educlick*, empresa de sistemas interactivos para aulas.

2. **http://www.android.com/**

   Página oficial de *Android*. Se encuentra toda la documentación de *Android*.

3. **http://www.google.es/**

   Página oficial de *Google*. Búsqueda de información en la web.

4. **http://www.mysql.com/**

   Página oficial de *MySQL*. Documentación de la base de datos *MySQL*.

5. **MEIER, Reto. Professional 2 Application Development. Ed. Digital. Indianapolis: Wiley Publishing, 2010. 580 p. ISBN: 978-0-470-56552-0**

6. **Hello, Android. Ed. 2. Ed Burnette, 2009. 250 p. ISBN-10: 1-934356-49-2. ISBN-13: 978-1-934356-49-4**

7. **http://msdn.microsoft.com/**

   Página principal de *Microsoft Development Network Plataforms*.

8. **http://inthehand.com/**

   Página oficial de *InTheHand*. Información sobre la librería Bluetooth empleada.

9. **http://www.google.com/codesearch**

   Página para la búsqueda de código fuente que puede servir de ejemplo.

10. **http://www.bluetooth.com/bluetooth/**

    Página oficial de *Bluetooth*.