PROYECTO FINAL DE CARRERA

La implementación de una aplicación móvil que detecta somnolencia en la conducción

(Implementation of mobile application that detects driver drowsiness)

Estudios: Ingeniería de Telecomunicación

Autor: Federico Guede Fernández

Director/a: Mireya Fernández Chimeno

Año: 2012


UPC
telecom
BCN

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Quiero dedicar este proyecto final de carrera a todas las personas que me han acompañado en este camino desde el principio y hasta el final.

A Mireya Fernández y Juan Ramos por haber confiado en mí, haberme dado la oportunidad de participar en este proyecto, por sus consejos y por la dirección de este proyecto. Agradezco enormemente que hayan tenido siempre abierta la puerta de su despacho para cualquier consulta. También agradezco a Miguel Ángel García por sus consejos.

Gracias también a los compañeros del laboratorio Giuseppe, Benjamín, Hoostins, Francisco, Francesc, Aleix, Marco, Ricardo, Josep Oriol y Raúl.

A mis padres Federico y Carmina, a mi hermana Clara y a mi cuñado Pablo y al resto de mi familia por su cariño y afecto a lo largo de toda la vida.

Y por último a todos mis amigos y compañeros de carrera por todos los buenos momentos que hemos compartido a lo largo de la carrera.

Resumen

El objetivo de este proyecto es el diseño e implementación de una aplicación para smartphone con los sistemas operativos Windows Mobile y Android, que detecte la somnolencia durante la conducción mediante el procesado de la señal respiratoria del conductor. Para ello, el dispositivo se conecta vía Bluetooth a un sistema de registro de señal respiratoria basado en una banda inductiva, que envía las muestras de respiración al smartphone donde la aplicación los procesa en tiempo real obteniendo el estado de somnolencia actual. Dada la diversidad de los sistemas operativos para smartphone y la proliferación de los mismos, este proyecto también pretende que el diseño y la implementación sean lo más portables posibles entre los distintos sistemas operativos para móviles. La aplicación desarrollada funciona correctamente para smartphones en ambos sistemas operativos, y en la actualidad es utilizada para hacer pruebas de campo, para en un futuro próximo usarlo en flotas de conductores profesionales. La portabilidad del núcleo de la aplicación entre distintos sistemas operativos es elevada, debiendo sólo realizarse cambios en los parámetros que son para cada sistema operativo como la interacción gráfica y conexión Bluetooth.

Abstract

The scope of this project is the design and implementation of a Windows Mobile and Android smartphone application that detects drowsiness while driving by processing of driver respiratory signal. For this, the smartphone connects via Bluetooth to a signal recording system based on an inductive band that sends breathing samples to the smartphone where the application processes them in real time to obtain the current state of drowsiness. Due to the diversity and the proliferation of smartphone operating systems, this project is also intended that the design and implementation are as portable as possible between the different mobile operating systems. The developed application works properly on smartphones in both operating systems, and currently is used for field tests. In the near future it will be used in fleets of professional drivers. The portability of the core of the application is high between different operating systems and should only make changes in the parameters for each operating system as graphical interaction and Bluetooth connection.

Resum

L'objectiu d'aquest projecte és el disseny i implementació d'una aplicació per smartphone amb els sistemes operatius Windows Mobile i Android, que detecti la somnolència durant la conducció mitjançant el processament del senyal respiratòria del conductor. Per això, el dispositiu es connecta via Bluetooth a un sistema de registre de senyal respiratòria basat en una banda inductiva, que envia les mostres de respiració al smartphone on l'aplicació els processa en temps real obtenint l'estat de somnolència actual. Atesa la diversitat dels sistemes operatius per smartphone i la proliferació d'aquests, aquest projecte també pretén que el disseny i la implementació siguin el més portables possibles entre els diferents sistemes operatius per a mòbils. L'aplicació desenvolupada funciona correctament per a smartphones en ambdós sistemes operatius, i en l'actualitat és utilitzada per fer proves de camp, per en un futur pròxim usar-lo en flotes de conductors professionals. La portabilitat del nucli de l'aplicació entre diferents sistemes operatius és elevada, i només realitzar canvis en els paràmetres que són per a cada sistema operatiu com la interacció gràfica i connexió Bluetooth.

1. Introducción

1.1 Introducción

Es conocida la extensión del uso de dispositivos móviles entre la población. El objetivo de este proyecto es el diseño e implementación de una aplicación para móviles que implementa un algoritmo de detección de somnolencia en la conducción a partir de la señal respiratoria. Esta aplicación tiene la finalidad de poder avisar al conductor en caso de somnolencia y de esta forma evitar los accidentes.

Dormir adecuadamente y durante el tiempo necesario es mucho más importante de lo que en principio se pudiera pensar para las tareas que conllevan riesgo, como es el caso de la conducción. El conducir requiere una serie de habilidades que se reducen bastante cuando falta el descanso. Cada conductor asume una responsabilidad respecto de su propia seguridad y la de otras personas.

A las 24 horas de privación se produce un incremento del tiempo de reacción e hipersensibilidad a la distracción (dificultad de mantener la atención, sobre todo en tareas continuas y rutinarias, como es el caso de la conducción). Si la privación continúa más tiempo, se observan síntomas de confusión, dificultad de expresión, etc.

Los efectos de la pérdida de sueño son acumulativos y más notorios en la ejecución de tareas monótonas, como algunos tipos de conducción. En ocasiones los mecanismos utilizados para afrontar este déficit de sueño pueden hacer al conductor inconsciente de la pérdida de sueño acumulada, lo que lo convierte en especialmente vulnerable a episodios de somnolencia repentina, fundamentalmente en situaciones donde la alerta se encuentra muy disminuida (por ejemplo, durante un largo período de inmovilidad debido a la postura durante la conducción), pudiendo aparecer una somnolencia irresistible e, incluso, breves episodios de sueño [1].

En el año 2010 la somnolencia fue la causante de un 30% de los accidentes de tráfico según la DGT. Este porcentaje es aún mayor si tenemos en cuenta que sólo las personas que sobrevivieron a estos accidentes pueden contar dentro de la estadística y que aproximadamente 11 millones de conductores adultos admiten haber tenido un accidente o casi un accidente porque se durmieron o estaban demasiado cansados para conducir [1].

En el sistema propuesto para la detección de somnolencia, la aplicación se conecta vía Bluetooth con el perfil puerto serie, a un sensor de respiración. El sensor envía las muestras de respiración al smartphone, donde se está ejecutando la aplicación que los recibe, procesa, registra, muestra un indicador a modo de semáforo y en caso de que el estado de somnolencia sea crítico emite un pitido y genera y envía un SMS de emergencia.

A nivel de dispositivos móviles hay una gran variedad de fabricantes y sistemas operativos, y por ello, antes de la implementación es necesario hacer un análisis del mercado de los sistemas operativos y un análisis de las principales características de los sistemas operativos. De esta forma viendo la diversidad de lenguajes y entornos de desarrollo, a la hora de programar la aplicación, se busca en el diseño e implementación la sencillez a la hora de portar la aplicación a los distintos sistemas operativos y al mismo tiempo sea lo suficientemente eficiente para la ejecución del algoritmo de detección de somnolencia en tiempo real.

1.2 Objetivos

Este proyecto tiene dos objetivos principales. El primero de ellos es la implementación de una aplicación para los smartphone de Windows Mobile y Android que detecte la somnolencia en la conducción. Más concretamente la aplicación se debe conectar por Bluetooth a una banda respiratoria que envía los valores de la señal respiratoria al smartphone y la aplicación debe leer estos datos, procesarlos y avisar en tiempo real si el estado de somnolencia es crítico.

El segundo de los objetivos es que tanto el diseño como la implementación de la aplicación sea lo más portable posible a otros sistemas operativos distintos de Windows Mobile y Android, sin comprometer el rendimiento de la aplicación.

1.3 Estructura de la memoria

La memoria está divida en 6 capítulos. El primer capítulo es de introducción. El segundo capítulo trata sobre el estudio de la cuota de mercado de los distintos sistemas operativos. El tercer capítulo es un análisis de las características de los distintos sistemas operativos: características destacables, arquitectura, lenguaje y entorno de programación, documentación y distribución.

El cuarto capítulo es el núcleo de la memoria, donde explica los requisitos detallados de la aplicación, el diseño y la implementación de la aplicación. Este capítulo está dividido según las capas y módulos de la aplicación detallando las peculiaridades de la programación tanto para Windows Mobile como para Android.

El quinto capítulo explica las diferentes pruebas realizadas para verificar el funcionamiento correcto de la aplicación.

El sexto y último capítulo contiene las conclusiones.

2. Análisis de mercado

Los sistemas operativos (SO) que analizamos y comparamos son iOS (Apple, iPhone), Android (Google), Windows Mobile, RIM (Research in Motion, Blackberry) y Symbian OS (Nokia), éstos son los más populares.

Analizaremos los datos de cuota de mercado de los distintos SO a nivel mundial, europeo y español. Los datos son de distintas consultorías con prestigio internacional publicados a través de notas de prensa de los informes que realizan periódicamente.

Los datos utilizados para realizar este estudio de mercado están recopilados en septiembre de 2011.

A la hora de evaluar la cuota de mercado es interesante saber el tiempo que lleva el SO en el mercado.

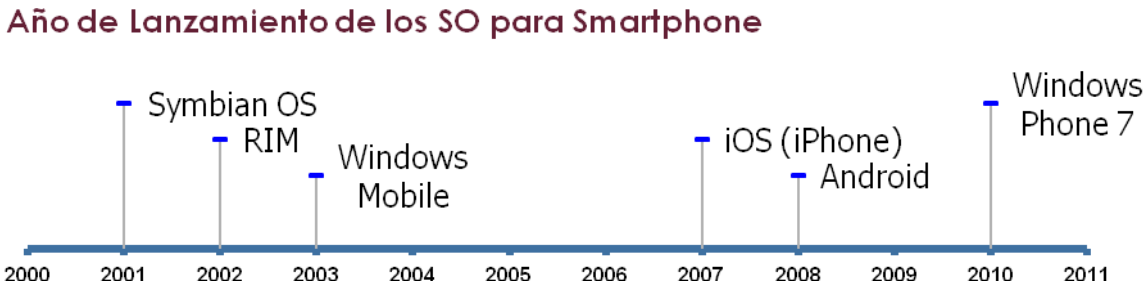


Figura 2.1: Año de lanzamiento de los sistemas operativos para smartphone

Además de las fechas de lanzamiento de los SO hay que tener en cuenta que en febrero de 2011 Nokia, principal fabricante de móviles con Symbian, llegó a un acuerdo con Microsoft para que los terminales de Nokia utilizaran el sistema operativo Windows Phone 7.

Evolución de las unidades de smartphone vs PC'S

En Junio de 2010 un grupo de analistas de Morgan Stanley presentó un estudio que predice para 2012 que el volumen de unidades de smartphones excederá al PC. Aproximadamente existen 480 millones de unidades smartphones vs 430 millones de PCs en 2012, llegando a 650 millones de smartphone en 2013.

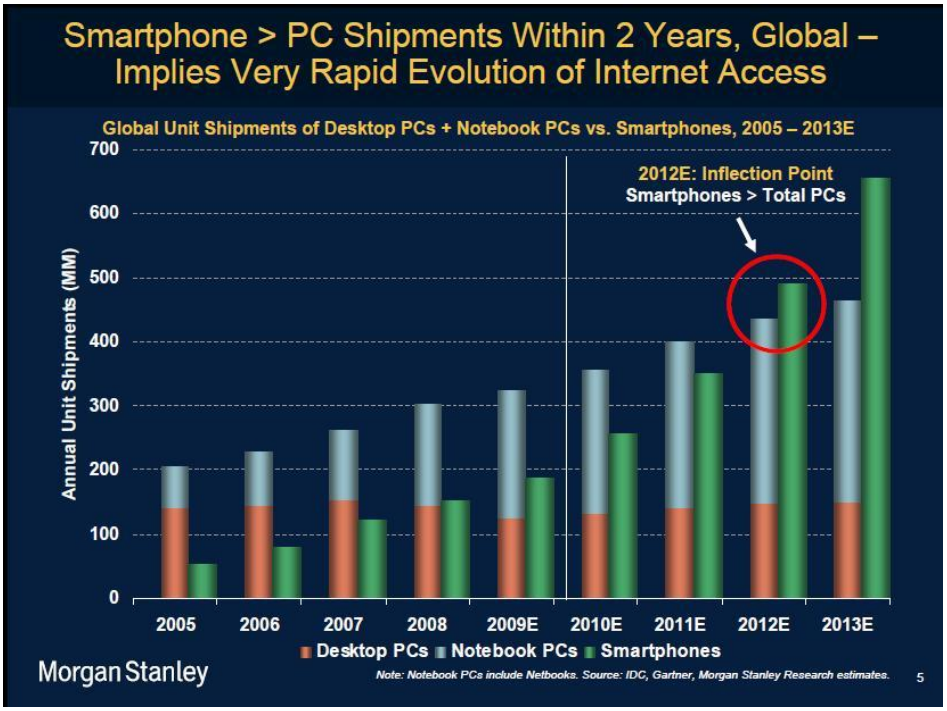


Figura 2.2: Evolución de la cuota de mercado smartphone vs PC (fuente Morgan Stanley)

El último informe de comScore con datos de Julio de 2011 dice que la cuota de mercado de los smartphones en España ha aumentado desde 31,9% en Julio del 2010 a 43,2% de julio 2011. En la siguiente figura vemos una comparación entre el resto de países de la UE. Según la media del trimestre terminado en julio de 2011, en toda la EU5 88,4 millones de usuarios declararon usar smartphones, un 44% más respecto al año anterior.

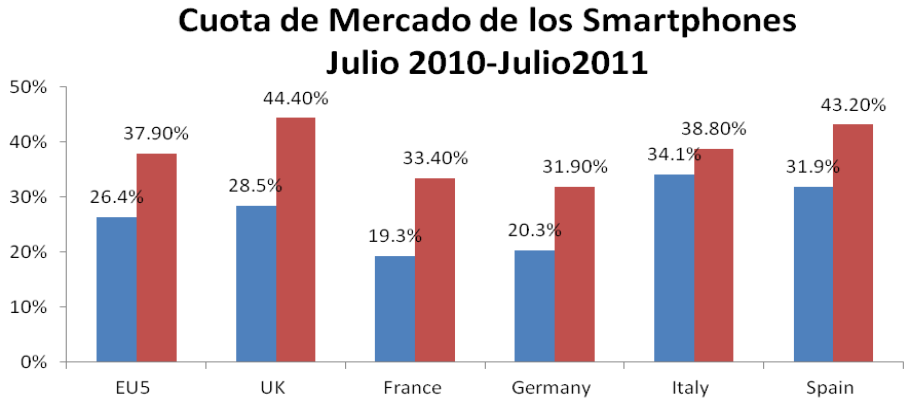


Figura 2.3: Cuota de mercado de smartphone por países en Europa julio 2010-2011

Reparto de la cuota de mercado de los sistemas operativos a nivel mundial

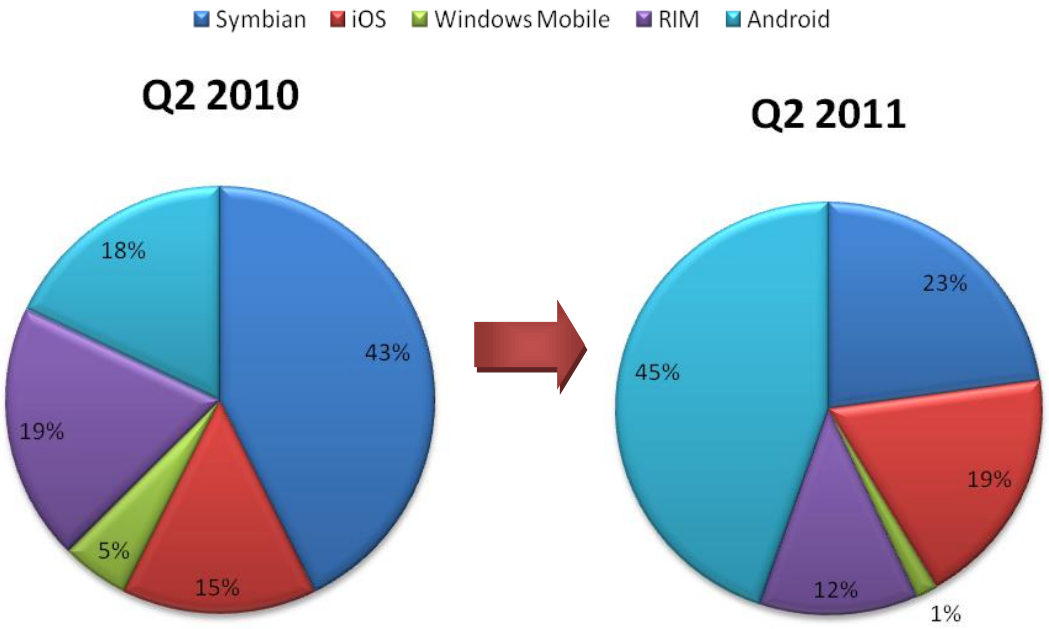


Figura 2.4: Reparto de cuota de mercado mundial

Las cifras de la evolución ventas de smartphone a nivel mundial a consumidores por sistemas operativos en el último año son proporcionados por la consultora Gartner.

Worldwide Smartphone Sales to End Users by Operating System in 2Q11 (Thousands of Units)

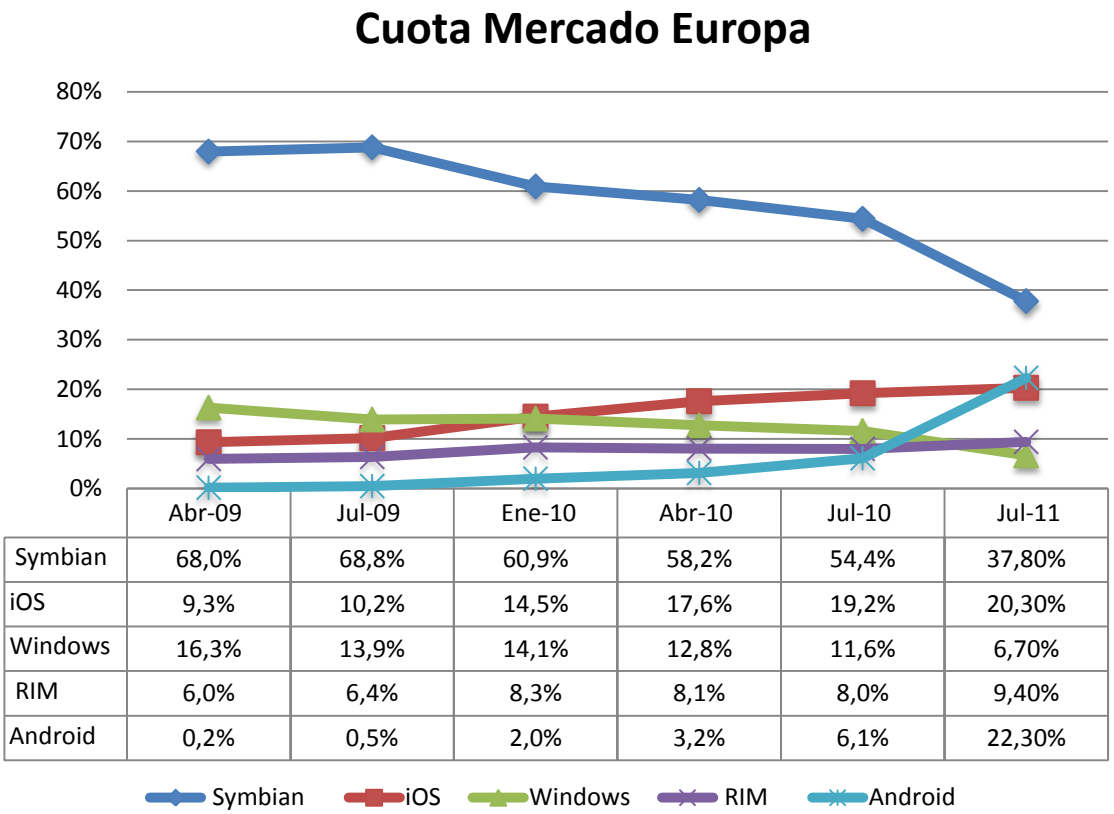
Operating System	2Q11 Units	2Q11 Market Share (%)	2Q10 Units	2Q10 Market Share (%)
Android	46,775.9	43.4	10,652.7	17.2
Symbian	23,853.2	22.1	25,386.8	40.9
iOS	19,628.8	18.2	8,743.0	14.1
Research In Motion	12,652.3	11.7	11,628.8	18.7
Bada	2,055.8	1.9	577.0	0.9
Microsoft	1,723.8	1.6	3,058.8	4.9
Others	1,050.6	1.0	2,010.9	3.2
Total	**107,740.4**	**100.0**	**62,058.1**	**100.0**

Tabla 2.1: Ventas mundiales de smartphone en segundo cuatrimestre 2011

Estos datos muestran que Android se coloca en el primer puesto al cuadruplicar el número de ventas y iOS se coloca en tercera posición, superando a Blackberry (RIM) al duplicar sus ventas.

Reparto de la cuota de mercado a nivel europeo

Este gráfico muestra la evolución de la cuota de mercado de cada SO en los últimos dos años y medio. La plataforma Android ya supera a Apple y se sitúa en el segundo puesto entre los SO para smartphones en los principales mercados europeos.



	Abr-09	Jul-09	Ene-10	Abr-10	Jul-10	Jul-11
Symbian	68,0%	68,8%	60,9%	58,2%	54,4%	37,80%
iOS	9,3%	10,2%	14,5%	17,6%	19,2%	20,30%
Windows	16,3%	13,9%	14,1%	12,8%	11,6%	6,70%
RIM	6,0%	6,4%	8,3%	8,1%	8,0%	9,40%
Android	0,2%	0,5%	2,0%	3,2%	6,1%	22,30%

Figura 2.5: Evolución de la cuota de mercado en Europa

Según la media del periodo trimestral terminado en julio de este año, 88,4 millones de suscriptores móviles en toda EU5 declararon usar smartphones, un 44% más respecto al año anterior. La plataforma Android de Google mostró el crecimiento más rápido entre las plataformas de smartphones en este periodo, aumentando su cuota de mercado de 6,1 a 22,3 puntos porcentuales. El IOS de Apple y RIM ganan poco más de un punto porcentual. Las únicas plataformas que perdieron cuota fueron Symbian y Microsoft, que disminuyeron un 16,1 y un 4,8% respectivamente.

Evolución de la cuota de mercado española

A continuación se muestran dos gráficos que reflejan la evolución del reparto de la cuota de mercado de los sistemas operativos en España en el último año.

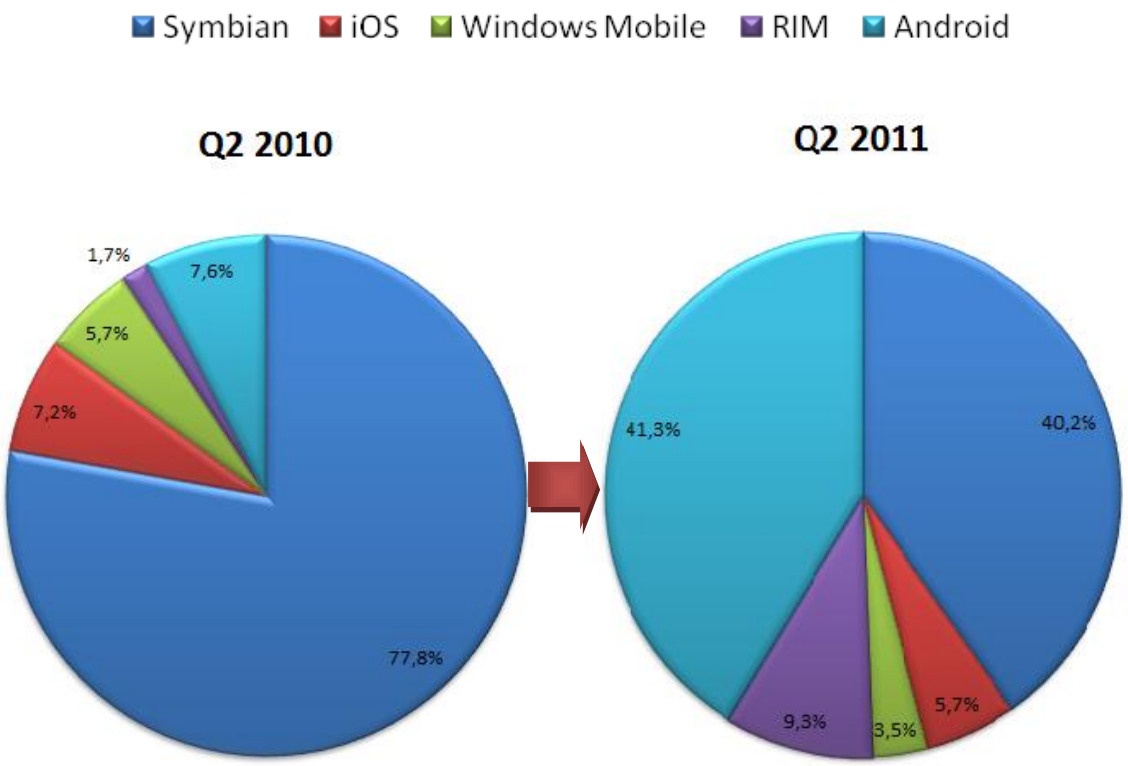


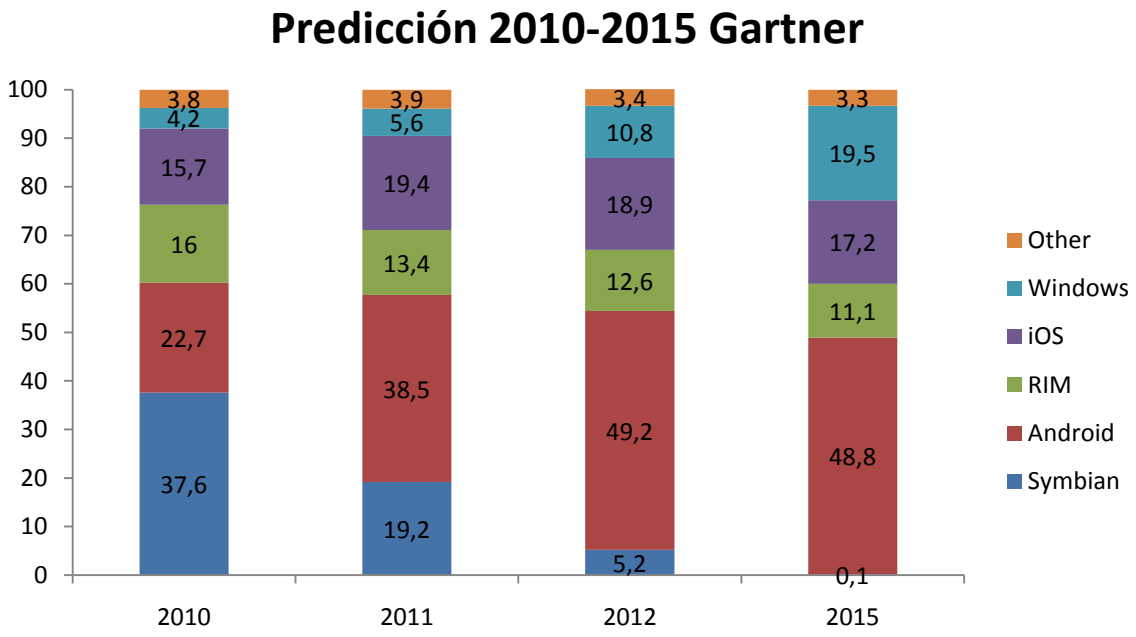
Figura 2.6: Evolución de la cuota de mercado de smartphone en España

Estos gráficos muestran que la caída de Symbian, el sistema de Nokia, en España ha sido notable en el último año pasando de un 77,8% a un 40,2%. Por otro lado igualmente notable ha sido el ascenso de Android, pasando de un 7,6% a 41,3% de cuota de mercado lo que los sitúa en el primer puesto.

Es destacable el retroceso de cuota de mercado que registra iOS (iPhone), que pasa del 7,2% al 5,7%, mientras que RIM escala del 1,7% al 9,3%.

Predicción de la consultora Gartner

La consultora Gartner pronosticó en Abril de 2011 que en 2015 Android se convertirá en el SO más utilizado en el mundo. La práctica desaparición de Symbian y la recuperación de Windows Phone 7 hasta alcanzar el 2º lugar con menos de la mitad de la cuota de mercado que Android.



Sistema Operativo (Unid. en Miles)	2010	2011	2012	2015
Symbian	111.577	89.930	32.666	661
Android	67.225	179.873	310.088	539.318
RIM	47.452	62.600	79.335	122.864
iOS	46.598	90.560	118.848	189.924
Windows Phone	12.378	26.346	68.156	215.998
Other	11.417	18.392	21.384	36.134

Tabla 2.2: Predicción de cuota de mercado a nivel mundial (fuente Gartner)

Predicción de la consultora IDC

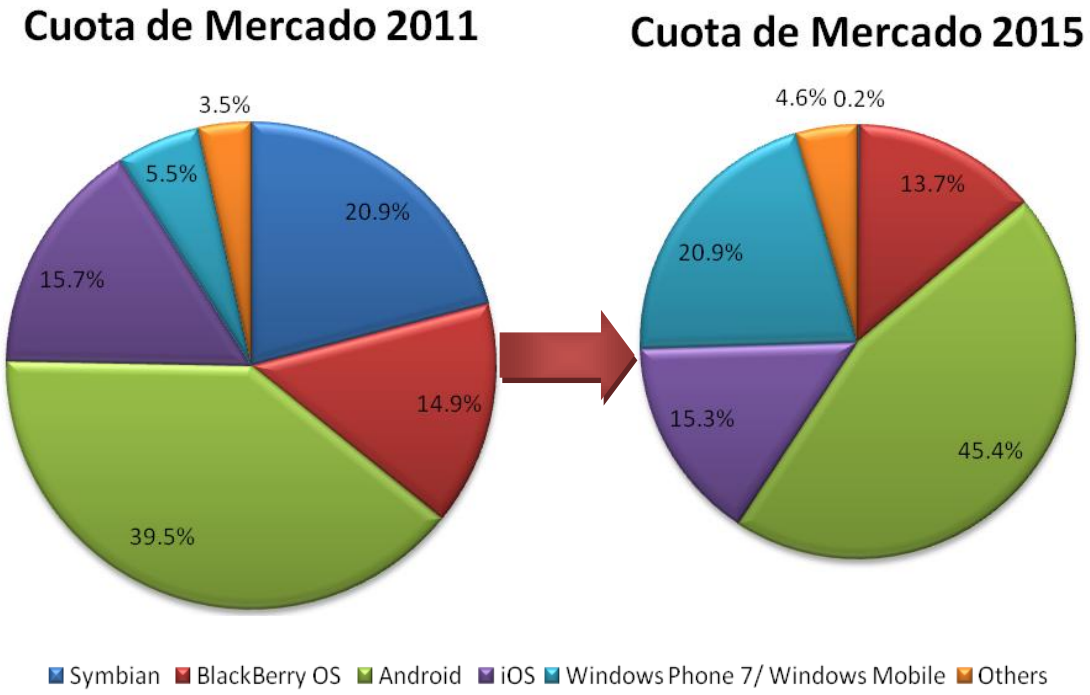


Figura 2.7: Predicción de la cuota de mercado (fuente IDC)

La consultora IDC realizó un estudio en abril de 2011 de predicción de la cuota de mercado de SO para smartphone a nivel mundial. Este informe del IDC coincide con Gartner en que Android será el sistema operativo móvil que más cuota ganará en los próximos cinco años, llegando a colocarse en el SO más utilizado con casi la mitad de toda la cuota de mercado. También coinciden en que Windows Phone se convertirá en el segundo más utilizado. Esto es debido a la alianza de Nokia con Microsoft anunciada en febrero de 2011. Nokia, la hasta el momento principal fabricante de móviles con Symbian, ha alcanzado un acuerdo con Microsoft para que los Nokia lleven el SO Windows Phone 7. Por eso también ambas consultoras predicen la práctica desaparición de móviles con Symbian 0,2% según IDC y 0,1% según Gartner.

La aplicación está implementada para Windows Mobile y Android. En el principio del proyecto sólo teníamos a nuestra disposición un dispositivo, Hp Ipaq 214 cuyo sistema operativo era Windows Mobile. Tras realizar el estudio de mercado que Android emergente en nuestro país y que iba ganando cuota de mercado a nivel mundial y europeo. Por eso es un

sistema operativo con mucho futuro y por ello se han elegido Windows Mobile y Android.

3. Análisis de los distintos sistemas operativos

Este capítulo analiza los distintos sistemas operativos para smartphone más utilizados. El análisis es sobre características principales, arquitectura, entorno de desarrollo, lenguaje de programación, documentación y distribución.

3.1 Windows Mobile

Microsoft .NET Compact Framework es un componente integral de los dispositivos Windows Mobile y Windows Embedded CE que permite generar y ejecutar aplicaciones administradas y utilizar servicios web. .NET Compact Framework incluye un Common Language Runtime (CLR) optimizado y un subconjunto de la biblioteca de clases de .NET Framework, que admite características como Windows Communication Foundation (WCF) y formularios Windows Forms. También contiene clases que están diseñadas exclusivamente para .NET Compact Framework [2].

.NET Compact Framework es el entorno en el que se ejecutan las aplicaciones administradas en los dispositivos. Proporciona acceso a las funciones subyacentes del dispositivo. Además, las aplicaciones y los componentes pueden interactuar en el dispositivo. .NET Compact Framework está diseñado para ofrecer un rendimiento óptimo bajo las restricciones de los limitados recursos de los dispositivos.

A nivel de Bluetooth soporta el perfil puerto serie.

Características:

- Controles y características adicionales para Windows Forms.
- Interoperabilidad con COM
- Direct3D y DirectDraw pars Windows Mobile
- Tipos genéricos, métodos anónimos, clases parciales.
- Librerías administradas para Pocket Outlook, Telefonía, puertos serie, acceso registro
- Threading mejorado.
- Clase SoundPlayer

- Con .NET Compact Framework 2.0 se introduce la clase SystemState dentro del ensamblado Microsoft.WindowsMobile.Status.dll nos permite monitorizar una gran cantidad de estados: Bluetooth, estado batería, altavoces, SMS,...

Arquitectura:

NET Compact Framework hereda la arquitectura .NET Framework completa de Common Language Runtime para ejecutar código administrado. Proporciona interoperabilidad con el sistema operativo Windows CE de un dispositivo para tener acceso a funciones nativas e integrar los componentes nativos favoritos en una aplicación.

Puede ejecutar aplicaciones nativas y administradas de manera simultánea. El host del dominio de aplicación, que también es una aplicación nativa, inicia una instancia del Common Language Runtime para ejecutar el código administrado.

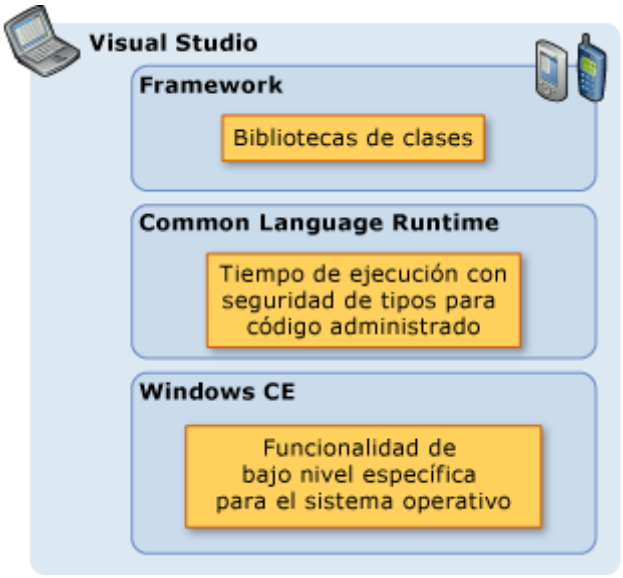


Figura 3.1: Arquitectura de Windows Mobile

Framework:

.NET Compact Framework es un subconjunto de .NET Framework pero también contiene características diseñadas en exclusiva. Ofrece prestaciones y facilidad de uso para acercar a los desarrolladores de aplicaciones nativas para dispositivos a .NET

Common Language Runtime:

También el Common Language Runtime (CLR) de .NET Compact Framework se ha vuelto a generar para permitir que los recursos restringidos se ejecuten en memoria limitada y lograr un uso eficaz de la energía.

Entre Windows CE y el Common Language Runtime existe una capa de adaptación de plataforma, que no aparece en la ilustración, para asignar las interfaces de servicios y dispositivos necesarias para CLR y Framework a los servicios e interfaces de Windows CE.

Windows CE:

.NET Compact Framework utiliza el sistema operativo Windows CE para la funcionalidad central y para diversas características específicas de dispositivos. Varios tipos y ensamblados, como los de los formularios Windows Forms, gráficos, dibujos y servicios Web, se han recompilado para que se ejecuten eficazmente en los dispositivos, en lugar de copiarse de .NET Framework completo.

.NET Compact Framework ofrece la siguiente interoperabilidad con Windows CE:

- Compatibilidad con seguridad nativa.
- Integración completa con programas de instalación nativos.
- Interoperabilidad con código nativo mediante la interoperabilidad COM y la invocación de plataformas. Un ejemplo de uso es encender el Bluetooth.

Entorno de desarrollo:

Microsoft provee en forma gratuita el "Windows Mobile 6 Professional and Standard Software Development Kits Refresh" SDK que incluye todo lo necesario para el desarrollo de aplicaciones en la plataforma Windows Mobile, pero para instalarlo necesita, como se indica en la sección de requisitos de la página de descargas, el Microsoft Visual Studio 2005 Standard Edition o superior esto implica que para desarrollar en WM es

necesario adquirir una licencia de Visual Studio 2005 o superior. Contiene un emulador de un smartphone para probar las aplicaciones desarrolladas, este emulador tiene sus limitaciones entre ellas que no soporta Bluetooth.

Lenguaje de Programación:

Se pueden desarrollar dos tipos de aplicaciones para Windows Mobile: con código nativo o con código administrado (managed code). Llamamos código nativo al código C++ que utiliza directamente la API de Windows Mobile, y código administrado al que utiliza las clases del .NET Compact Framework con C# o VB.Net. Windows Mobile es la única plataforma móvil importante que no soporta J2ME. El código nativo es más rápido y ocupa menos, además de proporcionar acceso a algunas características del hardware que son inaccesibles desde el Compact Framework. Sin embargo, en la mayor parte de los casos desarrollar código administrado es la mejor opción. El tamaño del ejecutable es cada vez menos importante, y si la velocidad es un factor crítico siempre se puede optar por programar en código nativo las partes de la aplicación que supongan un cuello de botella. Por lo demás, el desarrollo en .NET resulta mucho más fácil y cómodo. Algunas de las principales ventajas son:

- **Gestión automática de memoria**: No es necesario incluir instrucciones de destrucción de objetos. Como todo lenguaje de .Net, C# tiene a su disposición el recolector de basura del CLR.
- **Seguridad de tipos**: C# incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que evita que se produzcan errores difíciles de detectar por acceso a memoria no perteneciente a ningún objeto.

El motivo principal para usar C# y .NET es que pone a nuestra disposición todas las herramientas y servicios para llevar a cabo el proyecto: servicios de comunicación entrada/salida puerto serie, programación de WinForms y un aspecto gráfico moderno e integrado en el entorno de trabajo. Además de todo esto, se le suma la facilidad de programación del

lenguaje, su eficacia y seguridad ante fallos de programación y la cantidad de información de ayuda online que está disponible para los desarrolladores.

Documentación:

Microsoft tiene una página dedicada a los desarrolladores con todo tipo de tutoriales, videos, foros para ayudar al programador a descubrir las funcionalidades del sistema operativo. También tiene códigos de ejemplo que demuestran el uso de API.

Distribución:

Microsoft cuenta con una tienda de aplicaciones Windows MarketPlace donde os usuarios podrán buscar, navegar y comprar aplicaciones móviles desde un teléfono Windows o desde un PC con una identidad Windows Live ID.

3.2 Android

Android constituye una pila de software pensada especialmente para dispositivos móviles y que incluye tanto un sistema operativo, como *middleware* y diversas aplicaciones de usuario. Representa la primera incursión seria de Google en el mercado móvil y nace con la pretensión de extender su filosofía a dicho sector.

La licencia de distribución elegida para Android ha sido Apache 2.0, lo que lo convierte en software de libre distribución. El proyecto Android está capitaneado por Google y un conglomerado de otras empresas tecnológicas agrupadas bajo el nombre de *Open Handset Alliance* (OHA). El objetivo principal de esta alianza empresarial (que incluye a fabricantes de dispositivos y operadores, con firmas tan relevantes como Samsung, LG, Telefónica, Intel o Texas Instruments, entre otras muchas) es el desarrollo de estándares abiertos para la telefonía móvil como medida para incentivar su desarrollo y para mejorar la experiencia del usuario. La plataforma Android constituye su primera contribución en este sentido.

Los responsables del proyecto se han esforzado desde entonces en destacar que la motivación de Android es convertirse en algo más que un simple sistema operativo. Con Android se busca reunir en una misma plataforma todos los elementos necesarios que permitan al desarrollador controlar y aprovechar al máximo cualquier funcionalidad ofrecida por un dispositivo móvil (llamadas, mensajes de texto, cámara, agenda de contactos, conexión Wi-Fi, Bluetooth, aplicaciones ofimáticas, videojuegos, etc.), así como poder crear aplicaciones que sean verdaderamente portables, reutilizables y de rápido desarrollo. Android quiere mejorar y estandarizar el desarrollo de aplicaciones para cualquier dispositivo móvil y, por consiguiente, acabar con la perjudicial fragmentación existente hoy día [3].

A nivel de Bluetooth soporta el perfil puerto serie.

Características:

- Permite la representación de gráficos 2D y 3D.
- Posibilita el uso de bases de datos.
- Soporta un elevado número de formatos multimedia.
- Servicio de localización GSM.
- Controla los diferentes elementos hardware: Bluetooth, Wi-Fi, cámara, fotográfica o de vídeo, GPS, acelerómetro, infrarrojos, etc., siempre y cuando el dispositivo móvil lo contemple.
- Cuenta con un entorno de desarrollo muy cuidado mediante un SDK disponible de forma gratuita. Ofrece un plug-in para uno de los entornos de desarrollo más populares, Eclipse, y un emulador integrado para ejecutar las aplicaciones.

Arquitectura:

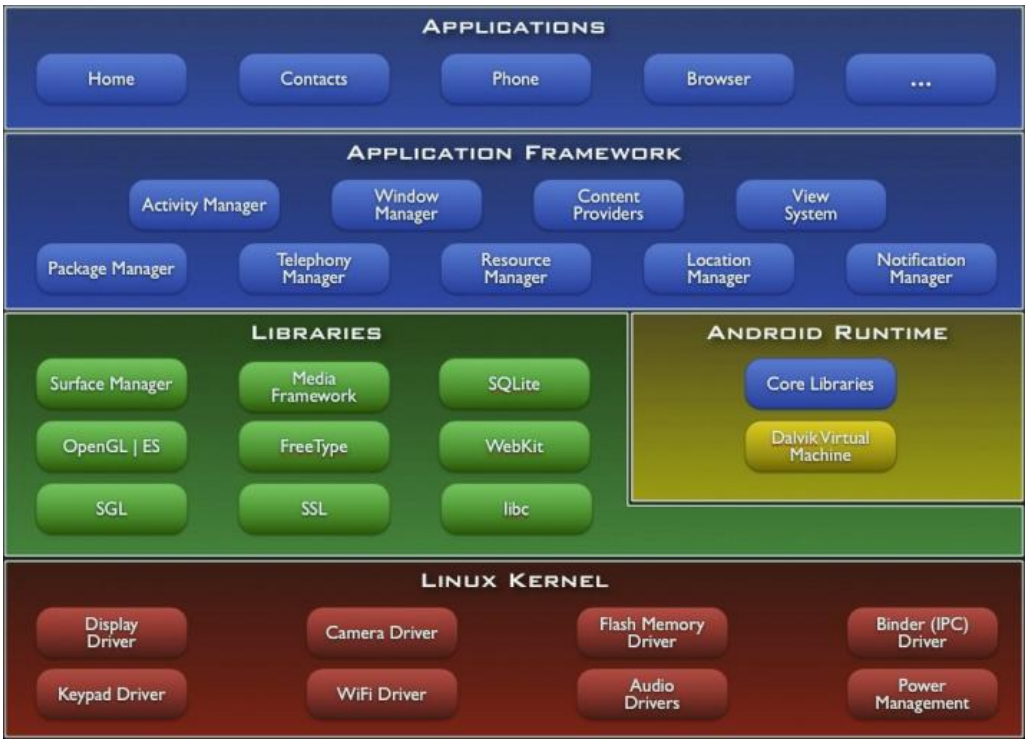


Figura 3.2: Arquitectura del sistema operativo Android [4]

Aplicaciones: Todas las aplicaciones están escritas en el lenguaje de programación Java. Entre otras, el propio sistema contiene algunas

aplicaciones básicas, como pueden ser, programa de SMS, calendario, contactos, etc.

Framework de aplicaciones: Los desarrolladores tienen acceso completo a las mismas API del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar el reúso de componentes. Éste mismo mecanismo permite que los componentes sean reemplazados por el usuario. Entre las API más importantes ubicadas aquí, se pueden encontrar las siguientes:

- Activity Manager, importante conjunto de API que gestiona el ciclo de vida de las aplicaciones en Android.
- View System, proporciona un gran número de elementos para poder construir interfaces de usuario (GUI), como listas, botones, check-boxes, tamaño de ventanas, control de las interfaces mediante tacto o teclado, etc. Incluye también algunas vistas estándar para las funcionalidades más frecuentes.
- Telephone Manager, incluye todas las API vinculadas a las funcionalidades propias del teléfono (llamadas, mensajes, etc.)

Librerías: Android incluye un set de librerías C/C++ utilizadas por varios componentes del sistema Android. Estas capacidades se exponen a los desarrolladores a través del framework de aplicaciones de Android. Algunas son: *System C library* (implementación librería C Standard), librerías de medios, librerías de gráficos, 3d, *SQLite*, entre otras.

Runtime de Android: Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual *Dalvik*. Toda la gestión de memoria y de procesos la lleva a cabo el runtime, que suspende y mata procesos sin posibilidad de intervenir desde las aplicaciones.

Núcleo - Linux: Android depende de un Linux versión 2.6 para los servicios base del sistema como seguridad, gestión de memoria, gestión

de procesos, etc. El núcleo también actúa como una capa de abstracción entre el hardware y el software.

Entorno de desarrollo:

Las herramientas para la realización de aplicaciones en Android más usuales son la plataforma de desarrollo Eclipse y un plug-in proporcionado por Google para la confección de proyectos tipo Android. El emulador de Android está contenido dentro del paquete del SDK, se trata de una herramienta creada por Google para poder probar las aplicaciones móviles sin necesidad de instalarlas en un dispositivo. El emulador tiene limitaciones y en concreto no soporta Bluetooth.

Lenguaje de desarrollo:

Las aplicaciones Android se escriben en Java, pero se ejecutan sobre una máquina virtual Dalvik, distinta a Java VM, optimizada para móviles. También se pueden programar aplicaciones en C++ a través del Native Development Kit (NDK) el uso de este lenguaje está recomendado para casos uso intensivo de CPU, que no reserven mucha memoria: simulaciones físicas, juegos.

Documentación:

Android tiene dedicada un web para desarrolladores que alberga una guía detallada para el desarrollador que explica cómo utilizar las distintas API del framework. Además, tiene un foro para desarrolladores, tutoriales que sirven como código de ejemplo y videos.

Distribución:

Android Market es la tienda de aplicaciones de Android, similar a la App Store del iPhone, disponiendo de un lugar centralizado para la descarga y compra de aplicaciones. Al contrario que en la tienda de Apple, no es necesario que Google apruebe las aplicaciones, sino simplemente registrarnos y subirlas. Con esto han conseguido que muchos desarrolladores creen aplicaciones para este sistema operativo, que aunque relativamente nuevo, es muy accesible. Para gestionar todas ellas,

hay un sistema de puntuaciones similar al usado en Youtube. Permite la descarga aplicaciones de forma gratuita y versiones de pago.

3.3 iOS

Es un sistema operativo móvil de Apple desarrollado originalmente para el iPhone, siendo después usado en todos los dispositivos iPhone, iPod Touch e iPad. A nivel de Bluetooth no soporta el perfil puerto serie.

Características:

La interfaz de usuario de iOS se basa en con el concepto de manipulación mediante gestos multitáctil. Los elementos de la interfaz se componen por deslizadores, interruptores y botones. La respuesta es inmediata y se provee de una interfaz fluida. La interacción con el sistema operativo se realiza mediante gestos como deslizar, tocar y pellizcar. Los acelerómetros y giroscopios internos son utilizados por algunas aplicaciones para responder a movimientos y gestos, como sacudir el aparato (en campos de texto es usado para deshacer y rehacer) o rotarlo (se suele usar para cambiar de posición vertical a modo apaisado).

La multitarea estaba reservada para aplicaciones por defecto del sistema. A Apple le preocupaba los problemas de batería y rendimiento si se permitiese correr varias aplicaciones de terceros al mismo tiempo. A partir de iOS 4, dispositivos de tercera generación y posteriores soportan el uso de 7 APIs para multitarea, específicamente:

1. Audio en segundo plano
2. Voz IP
3. Localización en segundo plano
4. Notificaciones push
5. Notificaciones locales
6. Completado de tareas
7. Cambio rápido de aplicaciones

Arquitectura:

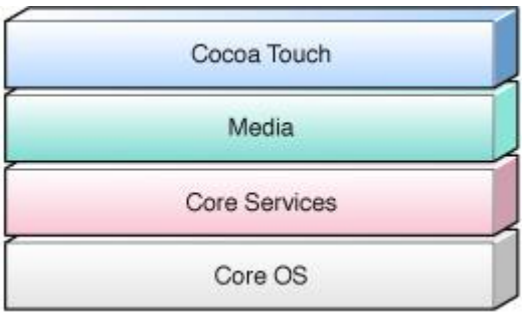


Figura 3.3: Arquitectura del sistema operativo iOS [5]

Cocoa Touch: es la capa más importante de todas en el iPhone OS, ella comprende dos frameworks fundamentales, como son el UIKit y el Foundation framework. Se puede decir que Cocoa Touch es una evolución del API, ya existente en la plataforma Mac, añadiendo características para el soporte de la tecnología multitáctil

- **UIKit:** Framework en Objective-C que provee la infraestructura para la implementación de los gráficos, la estructura de la aplicación, el control de los eventos, el manejo de la interfaz, la representación de las vistas y controles así como el soporte de texto y contenido web. Soporta a su vez código para el manejo del acelerómetro, la cámara, la librería de fotos y toda la información que posee el dispositivo.

- **Foundation Framework:** Define el acceso y manejo de objetos, provee acceso a los tipos de datos primitivos, colecciones y servicios del sistema operativo.

Media: Esta capa contiene los frameworks y servicios dependientes de Core Services y que proveen los servicios de gráficos y multimedia a la capa superior, Cocoa Touch. Incluye Core Graphics, OpenGLES, Core Animation, Core Audio y tecnologías de Video.

Core Services: Los frameworks de esta capa proveen la manipulación de strings, colecciones, el manejo de contactos y las preferencias así como las utilidades de URL o de la red. Esta capa incluye el framework Core

Foundation, el cual ofrece una abstracción a los tipos de datos, como strings y colecciones.

Core OS: Esta es el nivel que contiene el Kernel, los ficheros del sistema y la infraestructura de red, seguridad, manejo de la memoria y los drivers del dispositivo.

Entorno de desarrollo:
Para poder desarrollar dispone de tres herramientas:

1. Xcode: Es un IDE (entorno integrado de desarrollo) que permite escribir, compilar, ejecutar y depurar el código, así como organizar los ficheros por proyectos e importarlos y exportarlos de manera sencilla y eficiente. se puede elegir entre realizar el build en un simulador o en el propio terminal conectado al Mac.

2. Interfaz Builder: Es la herramienta para el desarrollo visual de la interfaz gráfica de las aplicaciones tanto para Mac como recientemente para el iPhone. Mediante drag and drop (arrastrar y soltar) se van añadiendo los componentes, controles y creando la interfaz de la aplicación

3. Instruments: es una potente herramienta de depuración y de análisis de rendimiento de las aplicaciones. Permite ejecutar la aplicación tanto en el simulador como en el teléfono y realizar trazas de la cantidad de memoria que está consumiendo en cada momento (un aspecto vital de las aplicaciones al ser ejecutadas en un entorno con más limitaciones de memoria y rendimiento que en una CPU) así como del uso de la batería que está haciendo.

Aparte de las XCode Tools, el SDK de Apple también contiene el iPhone Simulator, que no es más que un simple iPhone integrado dentro del Mac. El SDK está disponible para su descarga, pero es necesario tener un MAC para que este funcione y estar registrado en la web de desarrolladores de Apple.

Lenguaje de programación:

El lenguaje de programación es Objetive-C. Es una extensión de C para hacerlo orientado a objetos, basado en Smalltalk.

- Más limpio, pequeño y rápido de aprender que C++.
- Lenguaje utilizado por Mac OS X.
- No está estandarizado por ningún organismo internacional, Apple y NEXTSTEP son quienes han contribuido en crear este lenguaje. Ambos trabajan en mantener y mejorarlo
- Gestión de memoria manual. Por cuenta del programador como C++
- Es un lenguaje muy dinámico. Muchas decisiones se toman en tiempo de ejecución: memoria dinámica y tipos dinámicos

Documentación:

Al igual que Windows Mobile y Android, Apple tiene dedicada un web para desarrolladores que alberga una guía detallada para el desarrollador que explica cómo utilizar las distintas APIs del framework, guías de utilización de las herramientas de programación, guía de programación y videos explicativos.

Distribución:

Para poder distribuir la aplicación esta debe ser aprobada por Apple y el AppStore es el lugar donde están todas las aplicaciones que se pueden descargar. Sólo se pueden probar las aplicaciones en el emulador que viene con el SDK, si se quiere probarlas en un iPhone o subirlas al App Store para su distribución, hay que pagar una licencia de 99$.

3.4 Blackberry

Los smartphones BlackBerry® están diseñados de principio a fin como dispositivos basados en Java®: todas las aplicaciones para smartphones están escritas en Java® ME. Los smartphones BlackBerry ofrecen compatibilidad para MIDP 1.0 e CLDC 1.0, como mínimo. Asimismo, los smartphones que ejecutan BlackBerry® Device Software v4.0 o una versión superior son compatibles con MIDP 2.0/CLDC1.1 [6].

Las API de Java en los smartphones BlackBerry permiten desarrollar aplicaciones cliente enriquecidas con las siguientes características:

- Interfaces de usuario personalizadas
- Almacenamiento de datos local en el dispositivo
- Interfaces de escucha de eventos y de sistemas
- Transporte inalámbrico seguro mediante HTTP y TCP
- Cobertura de red y compatibilidad perfecta con el modo roaming
- Está implementado el perfil puerto serie de Bluetooth

Características:

- Subprocesos siempre activos en un segundo plano: A diferencia de otros marcos de desarrollo móvil, el marco de aplicaciones de BlackBerry permite que las aplicaciones sigan ejecutándose en un segundo plano (en modo de suspensión). Utilizando la tecnología Push de BlackBerry con auténtica multitarea en SO móviles para crear aplicaciones dinámicas (p. ej. clientes de mensajería instantánea).
- Periféricos Bluetooth: La introducción de la funcionalidad Bluetooth® para teléfonos BlackBerry en BlackBerry® Device Software v4.3 marcó el comienzo de una nueva era de posibilidades para los smartphones BlackBerry. Es posible diseñar aplicaciones ampliadas a varios tipos de hardware para Bluetooth, incluidos auriculares, impresoras, dispositivos de escritura y transmisores

GPS. Puesto que tanto el entorno de Java como los protocolos de Bluetooth cumplen los estándares del sector.

- Servicios basados: en la ubicación no cabe duda de que los servicios basados en la ubicación son uno de los temas de actualidad del desarrollo móvil y que los smartphones BlackBerry son la cuestión más candente dentro de este tema. La compatibilidad para JSR 179 brinda la funcionalidad GPS (disponible en algunos smartphones BlackBerry) para incorporar lo servicios basados en la ubicación en sus aplicaciones.
-

Entorno y lenguaje de desarrollo:

El enfoque basado en el desarrollo de aplicaciones en Java es recomendable para los desarrolladores que tienen experiencia en desarrollar aplicaciones en Java. El desarrollo en Java le permite crear una gran variedad de aplicaciones con numerosas características (p. ej. juegos y aplicaciones empresariales).

BlackBerry® Java Plug-in para Eclipse® amplía la plataforma de desarrollo Eclipse para que pueda crear aplicaciones en Java para smartphones BlackBerry en un entorno conocido.

Tiene disponibles simuladores de BlackBerry para comprobar el aspecto y funcionamiento de sus aplicaciones. Se pueden probar las aplicaciones en varios dispositivos y versiones del sistema operativo mediante los simuladores de smartphones BlackBerry.

Documentación:

Al igual que en iOS, Blackberry tiene una página dedicada para los desarrolladores, allí podemos encontrar tutoriales, videos, para obtener más detalles sobre cómo crear aplicaciones para smartphones BlackBerry.

Distribución:

Blackberry tienen un canal propio de distribución llamado App World donde se puede publicar la aplicación. Para aplicaciones de pago, el desarrollador se lleva el 70% de la venta. GetJar es un sitio web independiente donde puedes publicar las aplicaciones de Blackberry

3.5 Symbian

En febrero de 2011, Nokia, el principal fabricante de smartphones con este sistema operativo, anunció que migraría de Symbian a Windows Phone.

Características:

Symbian posee ciertas características que influyen de manera determinante en el desarrollo de aplicaciones. Primero, Symbian es un SO basado en ROM, no siempre ha habido posibilidades de grabar datos en la memoria del teléfono, aunque ahora generalmente se disponga de memorias flash. Segundo, ha sido diseñado para ahorrar batería.

Symbian está basado en un micro kernel. Una mínima porción del sistema tiene privilegios de kernel, el resto se ejecuta con privilegios de usuario, en modo de servidores. Una de las tareas del kernel es manejar las interrupciones y prioridades. En Symbian, cada aplicación corre en sus propios procesos y tiene acceso solo a su propio espacio de memoria. Este diseño hace que las aplicaciones para Symbian sean orientadas a “single threads” y no múltiples.

El sistema posee componentes que permiten el diseño de aplicaciones multiplataforma, esto es diferentes tamaños de pantalla, color, resolución, teclados, etc. La mayoría de estos componentes han sido diseñados en C++.

El diseño del sistema operativo permite que los aparatos con Symbian puedan estar en funcionamiento constante sin necesidad de ser reseteados, preservando la información del usuario y funcionando correctamente .Aunque esto último se está comprometiendo debido a la complejidad de los últimos equipos con Symbian y a la multitud de programas externos al SO.

Symbian implementa el perfil puerto serie de Bluetooth.

Arquitectura:

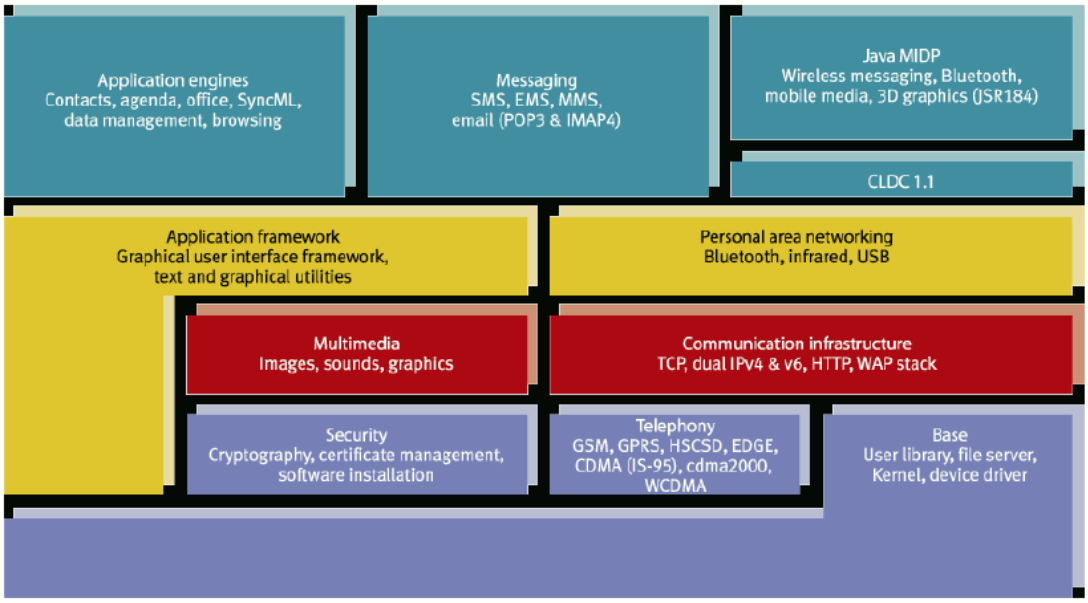


Figura 3.4: Arquitectura del sistema operativo Symbian [7]

- La *Base* incluye los componentes básicos de todo sistema operativo: kernel, gestión de la memoria y de los procesos, sistema de ficheros, manejadores de dispositivos, seguridad de bajo nivel, librerías básicas de usuario, etc.
- El Gestor de Telefonía realiza la gestión de los sistemas móviles celulares.
- Las pilas de infraestructura de comunicaciones y de red incluyen TCP/IP, GSM, GPRS, WAP, infrarrojos, Bluetooth y comunicaciones serie.
- El Gestor Multimedia se encarga tanto de gestionar la reproducción y grabación de audio como las diferentes funcionalidades de la imagen.
- El Gestor de la Seguridad se encarga de gestionar la seguridad relacionada con la descarga e instalación de las aplicaciones.
- El framework para aplicaciones contiene las librerías para la gestión de datos, texto, portapapeles, gráficos, internacionalización, MMI.

- El Gestor de Mensajería realiza la gestión del correo electrónico, los mensajes de texto, las aplicaciones de serie (menús, agenda, contactos, etc.)

Entorno y Lenguaje de Desarrollo:

Symbian cuenta con cinco interfaces de usuario o plataformas para su sistema operativo, las denominadas Serie 60, Serie 80, Serie 90, UIQ y MOAP. La mayoría de los móviles utilizan la Serie 60, todos los de Sony Ericsson trabajan bajo UIQ, así como Motorola.
El lenguaje nativo de Symbian OS es el C++ aunque no en una implementación estándar. Existen múltiples SDKs (Software Development Kit) para el desarrollo de aplicaciones, siendo los principales UIQ y S60. Algunos fabricantes ofrecen SDKs propios o extensiones a los SDK para sus productos o para familias de productos que se pueden bajar de los sitios web.

Los SDK oficiales contienen documentación, los headers, las librerías necesarias para compilar un software Symbian, emuladores basados en Windows y un compilador. Hasta la versión 8 se incluye como compilador GCC, la versión 9 usa una nueva ABI (application binary interface) y requiere un compilador distinto.

La programación en C++ para Symbian requiere el uso de técnicas especiales como descriptores o CleanupStack, esto puede hacer que programas relativamente simples sean más difíciles de implementar que en otros entornos. Actualmente las técnicas de programación necesarias para desarrollar en Symbian hacen que los programas sean propensos a errores en rutinas de bajo nivel en lugar de errores en las funcionalidades específicas de la aplicación.

El primer IDE oficial y comercial para Symbian, Codewarrior, fue reemplazado durante el 2006 por Carbide c++ un IDE basado en Eclipse desarrollado por Nokia que se ofrece la versión Carbide.c++ Developer Edition para desarrolladores de aplicaciones

Documentación:

Existen numerosas comunidades de desarrolladores para Symbian donde podemos encontrar tutoriales, foros para empezar a programar en Symbian

Distribución:

No dispone de tienda de aplicaciones, los programas se pueden descargar de sitios web de descarga de aplicaciones de los fabricantes de los teléfonos.

3.6 Windows Phone 7

Es el sistema operativo de Microsoft posterior a Windows Mobile 6.5, fue lanzado en octubre del año 2010 y rompe bastante con la filosofía anterior de Windows Mobile.

Características:

Todos los Windows Phone 7 Series, vienen con un botón dedicado para Bing y se enfocan en 6 hubs o contenidos, que permiten manejar la información de una forma más simple:

- **Personas**: Ofrece una experiencia de participación social, al reunir a contenido relevante en función de la persona, incluyendo sus feeds en tiempo real de sus redes sociales y fotos. También ofrece un lugar central desde el cual envía actualizaciones a Facebook y Windows Live en un solo paso.

- **Fotos**: Este eje hace que sea fácil compartir imágenes y vídeo a una red social en un solo paso. Windows Phone 7 Series también reúne las fotos de un usuario vía Web o PC, haciendo del teléfono el lugar ideal para ver las imágenes y videos de una persona.

- **Juegos**: Ofrece por primera vez la experiencia de Xbox LIVE en un teléfono, incluidos los juegos de Xbox Live y la capacidad de ver el avatar de un jugador, logros y el perfil de jugador.

- **Música + Video**: Trae lo mejor de Zune incluyendo el contenido de la PC de un usuario (música, video), los servicios de música en línea e incluso una radio incorporada de FM.

- **Office**. Ofrece acceso a los espacios de trabajo de Office, OneNote y SharePoint en un solo lugar. Los usuarios pueden leer, editar y compartir documentos, con la potencia adicional de Outlook Mobile.

- **Bluetooth:** no soporta el perfil del puerto serie

Arquitectura

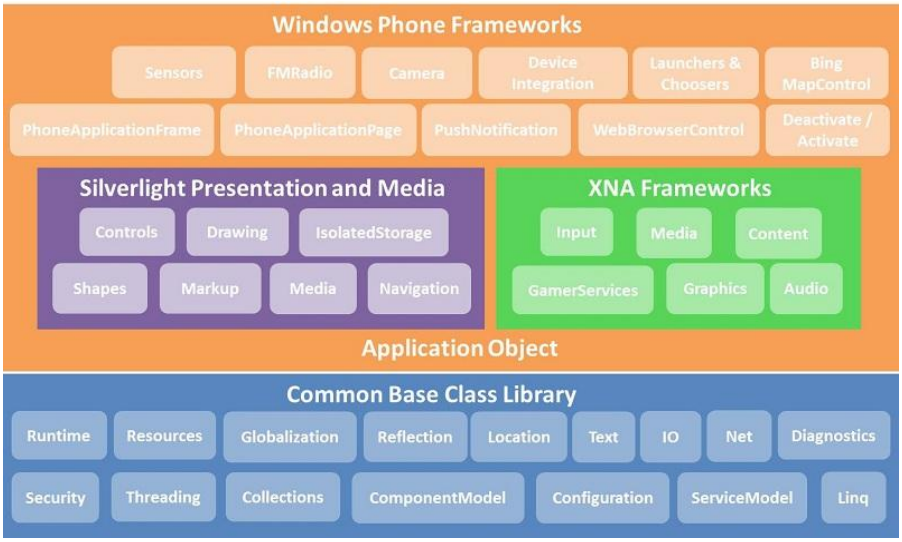


Figura 3.5: Arquitectura del sistema operativo Windows Phone 7 [8]

- **Silverlight**: es el framework ideal para crear aplicaciones de internet enriquecidas.
- **XNA Framework** está compuesto de software, servicios y recursos se centran en habilitar a los desarrolladores de juegos para que sea exitoso en el desarrollo de juegos. El XNA Framework provee un complete conjunto de APIs para el desarrollo de juegos.
- **Sensores** una variedad de sensores adquieren datos que pueden ser utilizados por los desarrolladores. Por ejemplo, entrada multitouch, acelerómetro, compás, giroscopio y micrófono son totalmente accesibles a través de APIs.
- **Medios:** ambos Silverlight y el XNA Framework proporcionan desarrollos con un modelo de programación para construir experiencias de usuarios enriquecidas que incorporan gráficos, animaciones y medios. Las APIs soportan una variedad de formatos.
- **Localización**: El servicio de localización de Microsoft para Windows Phone permite a los desarrolladores de aplicaciones acceder a la información de localización física del usuario desde una API. Los

desarrolladores pueden consultar la localización actual, captura eventos de cambio de localización.

Entorno de desarrollo:

Las herramientas de desarrollo necesarias son:

- Visual Studio 2010 Express para Windows Phone
- Windows Phone 7 Series Add-in para Visual Studio (para los desarrolladores que ya están trabajando con Visual Studio 2010)
- Emulador de Windows Phone 7 Series
- XNA Game Studio 4.0

Lenguaje de desarrollo:

Los lenguajes de programación son C# y Visual Basic. Posteriormente se elige si se va a utilizar el framework XNA, Silverlight o una combinación de los dos.

Documentación:

Numerosas guías, código de ejemplo y aplicaciones de ejemplo existen para ayudar a los desarrolladores para facilitar el esfuerzo de desarrollo para Windows Phone. Foros, blogs, y sitios webs están disponibles para desarrolladores para hacer preguntas y compartir información con la mayoría de comunidad de Windows Phone.

Distribución:

Después de haber completado una aplicación, un desarrollador puede hacer el programa disponible a otros usuarios a través del Windows Phone Marketplace.

A la hora de implementar la aplicación para Windows Mobile elegimos como lenguaje de programación C# por facilidad de implementación, al poder acceder a las APIs de .NET Compact Framework y al haber mucha documentación disponible y poder desarrollarlo desde el Microsoft Visual

Studio y al contrario que Java no necesita ejecutarse sobre una máquina virtual que podría comprometer el rendimiento de la aplicación.

4. Análisis de requisitos, diseño e implementación

Este capítulo es el central de toda la memoria. Explica primero los requisitos que debe cumplir la aplicación. Luego explica el diseño de la aplicación para que cumpla los requisitos. Finalmente explica a nivel de implementación de la aplicación, las principales semejanzas y diferencias entre las dos plataformas en las está implementada la aplicación, Windows Mobile y Android.

La aplicación está implementada para Windows Mobile y Android. En el principio del proyecto sólo teníamos a nuestra disposición un dispositivo, Hp Ipaq 214 cuyo sistema operativo era Windows Mobile y elegimos C# por facilidad de implementación, al poder acceder a las APIs de .NET Compact Framework y al haber mucha documentación disponible y poder desarrollarlo desde el Microsoft Visual Studio y al contrario que Java no necesitaba ejecutarse sobre una máquina virtual que podría comprometer el rendimiento de la aplicación. Tras realizar el estudio de mercado vimos que Android era un sistema operativo emergente y por ello el diseño y la implementación lo hemos hecho en Windows Mobile y Android.

4.1 Requisitos de la aplicación

Los requisitos que debe cumplir esta aplicación son:

1) El diseño y la implementación deben ser, en la medida de lo posible portables entre distintos sistemas operativos: en nuestro caso Windows Mobile y Android.

2) Conectarse a un dispositivo vía Bluetooth que esté dentro del alcance.

3) Transmitir y recibir datos vía Bluetooth mediante el protocolo perfil del puerto serie [9]. Los datos recibidos están muestreados a una frecuencia de 40 Hz.

4) Procesar los datos recibidos utilizando el algoritmo Ks.

5) Visualizar en la pantalla una imagen indicadora del estado de somnolencia en función del resultado del algoritmo.

6) Emitir un aviso acústico si el estado de somnolencia es crítico.

7) Enviar un SMS de emergencia si el estado de somnolencia es crítico.

8) Registrar en un archivo de texto los datos recibidos y los resultados obtenidos al aplicar el algoritmo.

9) La aplicación debe ser robusta y utilizar de forma eficiente los recursos del sistema operativo.

4.2 Diseño

Este apartado explica la fase de diseño de la aplicación, detallando la arquitectura del sistema y la funcionalidad de las principales clases y cómo interactúan entre ellas.

La aplicación está diseñada teniendo en cuenta los requisitos definidos previamente. La fase de diseño no aborda los detalles de la implementación sino que propone una idea general de la funcionalidad que encapsula.

El sistema está diseñado de forma modular utilizando la Programación Orientada a Objetos y la Programación en Capas, que consiste en dividir el código fuente según su funcionalidad principal. Ésto facilita la programación porque el desarrollador avanza en la programación del proyecto de forma ordenada, debido a que podrá avanzar de forma más segura en el desarrollo, al estar la aplicación general en varios módulos y capas, que pueden ser tratados de manera independiente. Facilita el mantenimiento, porque a la hora de actualizar, modificar y añadir componentes, evita que otras partes de la aplicación se vean afectadas.

También facilita la portabilidad de la aplicación en distintos SO porque sólo habrá que modificar aquellos módulos dependientes del SO, reduciendo el tiempo de desarrollo y de pruebas.

Arquitectura de la aplicación

El móvil se conecta con la banda respiratoria vía Bluetooth. El móvil envía una secuencia de inicialización a la banda. Cuando la banda recibe correctamente esta secuencia, empieza a enviar datos al móvil. Entonces que el móvil procesará los datos recibidos y calculará el estado de somnolencia que quedará registrado y emitirá un sonido en caso de estado crítico de somnolencia.

La banda respiratoria se comunica mediante Bluetooth y soporta únicamente el perfil del puerto serie, Serial Port Profile (SPP), por ello la aplicación utiliza este perfil.

Figura 4.1: Arquitectura de la aplicación

Diagrama de flujo de la aplicación

El diagrama de flujo nos da una visión de conjunto sobre el funcionamiento del programa. Donde podemos ver las funciones que realiza la aplicación y la secuencia que siguen dentro de la aplicación. Las funciones que están rellenadas de color azul significan que están implementadas sólo para la versión en Windows Mobile, en verde significa sólo Android y el resto significa que están implementadas en ambas plataformas.

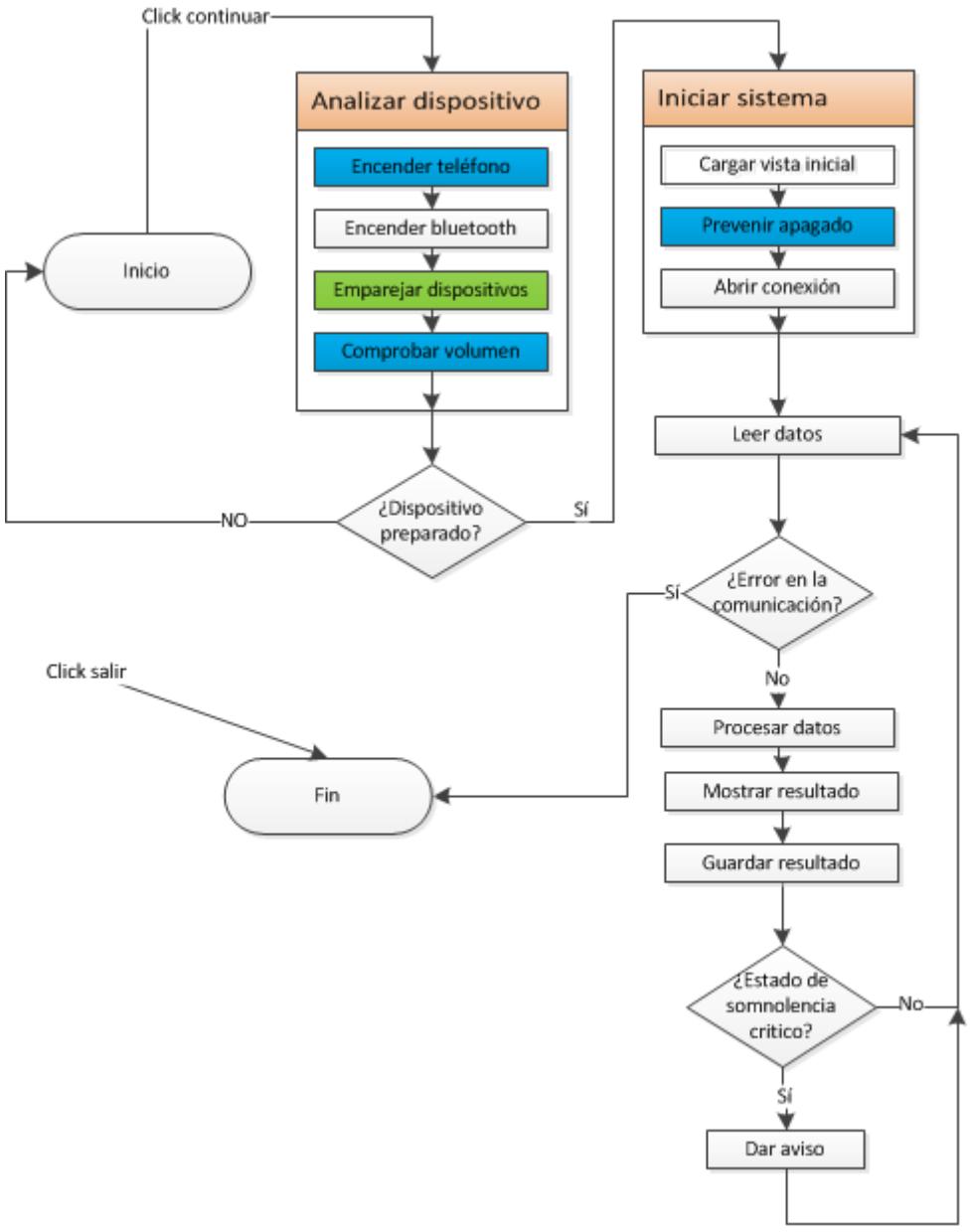


Figura 4.2: Diagrama de flujo de funcionamiento de la aplicación

Diseño de la aplicación

Los componentes de la aplicación los podemos agrupar en tres grandes capas: Vista, Lógica y Comunicación. La Vista contiene los módulos que se encargan de interactuar con el usuario. La capa Lógica es donde se encuentran los módulos que realizan la mayor parte del trabajo interno, en esta capa contiene la lógica de la aplicación y la funcionalidad de enlazar entre la capa de comunicación y la vista y también independiza los propios módulos de la lógica de las otras capas. Por último la capa de comunicación, que contiene el módulo de recepción de datos que se encarga de monitorizar la conexión y leer los datos recibidos.

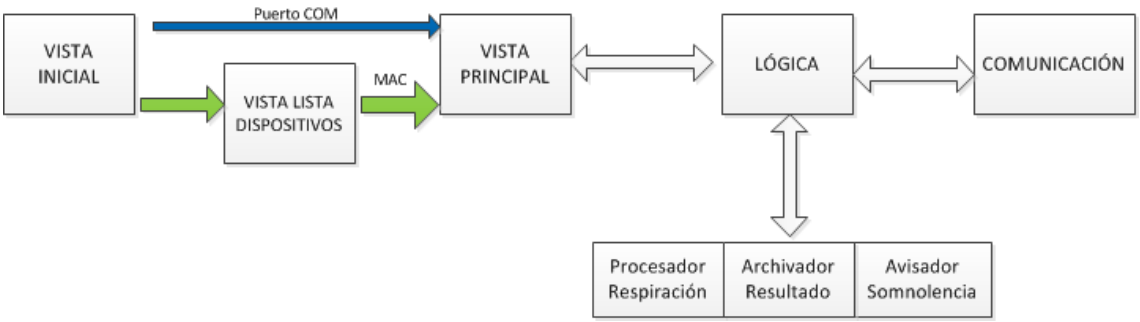


Figura 4.3: Diagrama de clases de la aplicación

Una buena práctica del diseño de interfaces de usuario es evitar realizar en el hilo principal de la aplicación, donde reside la Interfaz de Usuario, operaciones que requieran un largo procesado u operaciones bloqueantes de E/S [10]. De lo contrario la aplicación no podría responder a las acciones del usuario y daría la sensación de que la aplicación no responde. Esta aplicación separa en un hilo distinto del principal de la aplicación, las operaciones de la recepción de datos vía Bluetooth, a continuación en este mismo hilo secundario se realiza el procesado de los datos, almacenamiento de resultados, y el inicio y parada del sistema de aviso. Los detalles de cómo está implementado están explicados en el apartado siguiente.

La capa de Lógica es el núcleo de la aplicación, se encarga de actualizar el estado de somnolencia con los datos recibidos. El algoritmo está

implementado en la clase ProcesadorRespiracion que calcula el estado de somnolencia. Este módulo envía a la Vista el estado calculado; avisa si es necesario mediante la clase AvisadorSomonlencia y almacena el resultado mediante la clase ArchivadorResultados. Por otro lado también gestiona los eventos que recibe de la interfaz grafica.

La capa de Comunicación se encarga de monitorizar la conexión: abrir la conexión, enviar la secuencia para el inicio de la transmisión, leer datos disponibles en el buffer, extraer los valores de la trama de bytes recibidos, detectar posibles fallos que ocurran en la comunicación y avisar, llamar al controlador de la lógica para tratar las muestras recibidas

Para el inicio de la comunicación Bluetooth es necesario el previo emparejamiento de los dispositivos, esto tiene consecuencias distintas en el diseño de la aplicación para Android y Windows Mobile:

- El SDK de Android provee de una serie de API Bluetooth que permiten descubrir los dispositivos, emparejarlos y crear asociaciones.
- El framework .NET de Windows Mobile no soporta el descubrimiento de dispositivos y servicios. Sin embargo, existen librerías externas que suplen estas carencias. Las API's del Framework .NET permiten acceder al nivel del puerto serie. La aplicación únicamente utiliza el framework de .NET, esto asegura un mejor control del rendimiento y evita que librerías desarrolladas por terceros puedan comprometer el funcionamiento la aplicación.

Diseño de la interfaz

El programa tiene dos pantallas: una inicial que es la de bienvenida y requiere la interacción del usuario para iniciar el funcionamiento de todo el sistema y la pantalla principal cuya misión es indicar el estado de somnolencia actual. El algoritmo define tres estados, uno de inicialización y dos distintos de somnolencia: despierto y dormido. El estado que corresponde al momento presente es el que muestra al usuario.

La pantalla inicial, también se encarga de llamar al analizador del dispositivo que prepara el dispositivo e informa de la acciones a realizar o del estado del dispositivo (batería y nivel sonido)

Para la aplicación en Android existe en una pantalla intermedia, para el listado de dispositivos emparejados y los dispositivos disponibles al que el usuario desea asociarse. En Windows Mobile como no permite bajar a tanto nivel en la VistaInicial se selecciona el puerto previamente configurado.

Debido a la versatilidad del tamaño de la pantalla del dispositivo móvil, la aplicación soporta dos tipos de resolución en Windows y tres en Android.

La interfaz gráfica está diseñada siguiendo el documento "Human Guidelines Interface", que recoge recomendaciones destinadas a mejorar la experiencia para los usuarios, haciendo interfaces de usuario más intuitivas, compatibles y constantes. Estas recomendaciones están especificadas tanto para Windows Mobile[1] como para Android[2].

[1] Create Compatible User Interfaces: http://msdn.microsoft.com/en-us/library/bb677147.aspx

[2] User Interface Guidelines http://developer.android.com/guide/practices/ui_guidelines/index.html

4.3 Implementación

Este capítulo explica las principales diferencias en la programación de las distintas funcionalidades de la aplicación para ambas plataformas. Primero explica la parte de la interfaz gráfica de usuario, luego la parte que implementa la comunicación del móvil con la banda respiratoria y por último la parte de Lógica que contiene el procesado y registro de los datos, el sistema de avisos y por otro lado la configuración del dispositivo. Todas ellas están implementadas siguiendo las condiciones de diseño comentadas anteriormente. En el anexo B está el código fuente de la aplicación para Android y en el anexo C está el código fuente para Windows Mobile.

Las aplicaciones en los sistemas operativos Windows Mobile y Android no son iguales a las aplicaciones en los sistemas operativos de escritorio: sólo hay una aplicación en primer plano que normalmente ocupa toda la pantalla. Las aplicaciones están formadas por Actividades en Android y Formularios en Windows Mobile, que representan la pantalla que una aplicación muestra al usuario.

En Android al arrancar una nueva aplicación, ésta pasa a primer plano situando la vista encima de la que hubiera, formándose así una pila de actividades. El botón Back (< -) cierra la Actividad en primer plano y recupera la de la cima de la Pila (cerrando la aplicación en su caso).

En Windows Mobile el botón back ha sido de deshabilitado porque cierra la aplicación.

4.3.1 Interfaz gráfica de usuario

La interfaz gráfica de usuario (GUI) es la encargada de mostrar la información de la aplicación mediante imágenes y objetos gráficos y gestionar la interacción con el usuario. La interacción se realiza mediante las pantallas o ventanas de la aplicación. Los pasos seguidos para la programación han sido iterativamente:

1. Diseñar la interfaz de una parte de la aplicación, utilizando los widgets disponibles en el toolkit de desarrollo: posicionar los widgets, establecer sus dimensiones y modificar sus características visuales y funcionales (títulos, colores, comportamiento).
2. Realizar la captura de los eventos de la interfaz que permitan implementar la funcionalidad requerida.
3. Implementar cada uno de los manejadores correspondientes a los eventos capturados.

Tanto para Windows Mobile como para Android existen dos formas alternativas de diseñar la interfaz:

- Para Windows Mobile la primera consiste en crear una instancia de la clase `Form`, y añadir mediante código los correspondientes controles a la colección `Controls` correspondiente. Esta colección almacena todos los controles incluidos en el formulario. Para Android, en lugar de la clase `Form` utiliza la clase `Activity` y en lugar de `Controls` se utilizan `Views`.

- La otra es separar el diseño de formulario del aspecto funcional de la aplicación. El formulario está diseñado con el editor visual del entorno de desarrollo y la parte funcional mediante código.

El problema que tiene la primera alternativa es que no es posible reutilizar el formulario creado en otro punto de la aplicación. Por otro lado es una buena práctica del diseño de software que el código de la aplicación quede separado del de diseño, porque facilita la labor de detección de errores y actualización de la aplicación tanto por el autor como posibles

futuros desarrolladores que tengan que reinterpretar y modificar el código.

El diseño de la interfaz gráfica tanto para Windows Mobile como para Android está basado en la segunda alternativa, teniendo en cuenta las peculiaridades de ambas plataformas.

La captura de eventos del usuario la está implementada haciendo uso de mecanismos de gestión de eventos de .NET y del framework de Android. El gestor de eventos se encarga de recibir todos los eventos de la aplicación y llamar al manejador que tiene asignado para cada evento.

Los manejadores de eventos se encargan de realizar el conjunto de acciones asociadas a un evento determinado. Si ningún método gestiona un evento determinado, la acción del usuario que produce ese evento no obtiene respuesta.

Por otro lado para ambas plataformas se ha definido la orientación de la pantalla en vertical con un ángulo de orientación en 0 para obtener una correcta visualización.

Windows Mobile

La GUI está implementada con dos clases `VistaInicial.cs` y `VistaPrincipal.cs` ambas son una clases que heredan de la clase `Form`. El método `InitializeComponent()` define las propiedades y disposición de los controles dentro del formulario. Este método se ejecuta cada vez que se instancia el formulario.

Una vez creado el formulario, el diseñador de formularios nos permite diseñar la interfaz gráfica visualmente y genera automáticamente el código necesario para crear la interfaz.

El formulario está definido por dos ficheros uno de ellos con extensión `.Designer.cs`, y el otro con extensión `.resx`. El fichero con extensión `.Designer.cs` contiene todo el código fuente del formulario generado automáticamente por el diseñador `de` formularios. Visual Studio utiliza un tipo parcial para dividir el código fuente de la clase del formulario en dos,

la parte generada automáticamente, y la que introduce el desarrollador para gestionar los eventos de los componentes de la ventana. El código que genera la interfaz gráfica se implementa dentro del método `InitializeComponent()`, que es llamado desde el constructor de la clase.

El diseñador de formularios también nos permite seleccionar los eventos que queremos capturar de un control y asignarles un manejador para tratarlo.

Define la orientación de la pantalla de vertical a cero grados, según la especificación de la enumeración ScreenOrientation.

Este gráfico muestra el ciclo de Vida de un formulario

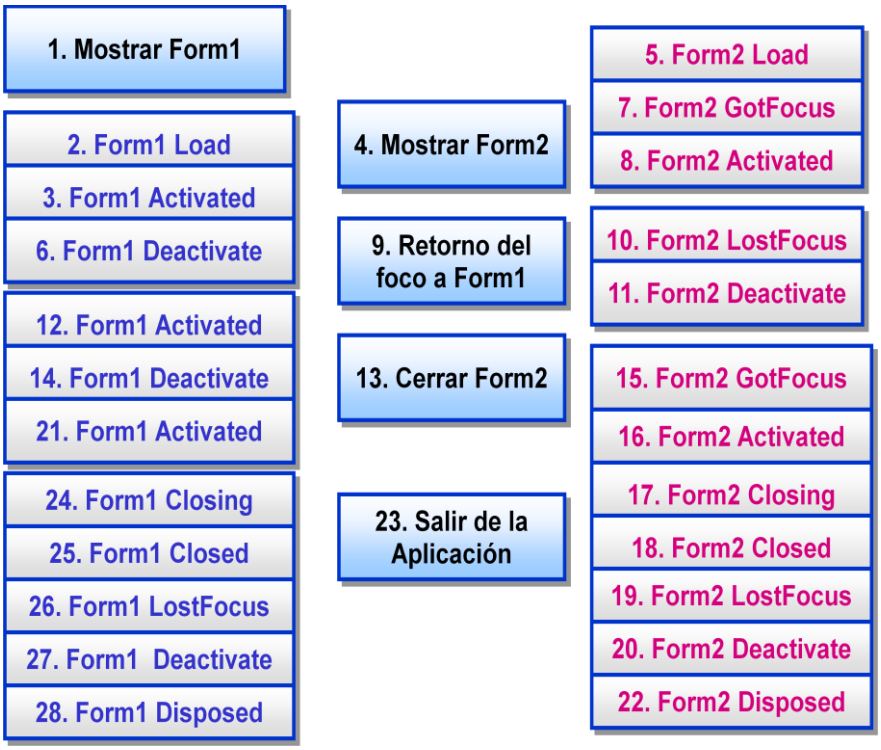


Figura 4.4: Ciclo de vida de un formulario [11]

Android

De forma equivalente a los formularios de Windows Mobile en Android existen las actividades. La aplicación tiene tres actividades: `VistaInicial.java`, `VistaListaDispositivos.java` y `VistaPrincipal.java`, que heredan de la clase Activity.

La GUI de una Actividad está creada mediante Vistas (`Views`), que es la clase que representa a un elemento básico de UI en Android, equivalente a controles en Windows Mobile. Las vistas están agrupadas en Diseños (`Layouts`) que muestran la aplicación.

Los layouts están diseñados con el editor gráfico que viene con el SDK de Android que genera un archivo `.xml`. El layout se establece en el método `onCreate()` de la Actividad. Una vez establecido, se obtienen las referencias a las Views, contenidas en el Layout llamando al método `findViewById()`.

La implementación gestión de los eventos de la UI es distinta según el tipo de fuente del evento: widgets, teclado o botones hardware. Las tres actividades de la aplicación tratan los eventos mediante:

- EventHandlers: Maneja los eventos de entrada sin importar dónde cuales son los elementos UI de la pantalla. No están necesariamente asociados a una vista: Botón atrás, tocar la pantalla. Los callbacks donde se tratan estos eventos están definidos en la actividad: `onKeyDown()` ,`onKeyUp()`, `onTouchEvent()`,…

- EventListerners: Maneja los eventos generados por elemento de UI específico. El método setOnxxxListener(). Permite utilizarlos de dos maneras una que se conoce como anónima, permite disitintos callbacks para un mismo evento y otra como parte de la actividad, sólo creamos un callback para un evento. Esta última reduce la carga de clases y la reserva de memoria de objetos.

En Android la rotación automática llama al método onCreate(), lo que implica el reinicio de la actividad, esto penalizaría el rendimiento de la aplicación. Por eso la aplicación desactiva esta opción, y además define la orientación como vertical. Para conseguirlo cada una de las actividades de la aplicación en el archivo `AndroidManifest.xml` define la propiedad `android:screenOrientation = "portrait"`.

Layout Relative soluciona el tema de la posición relativa, la parte del tamaño de las imágenes de estado se soluciona definiendo imágenes de distinto tamaño según la resolución de la pantalla. [12]
En Windows Mobile a partir del tamaño de la pantalla colocamos los elementos gráficos y definidos los dos bitmaps diferentes una para pantallas mayores que 480x240 y otras para menores.

En el siguiente grafico podemos ver el funcionamiento de la pila de Actividades de Android:

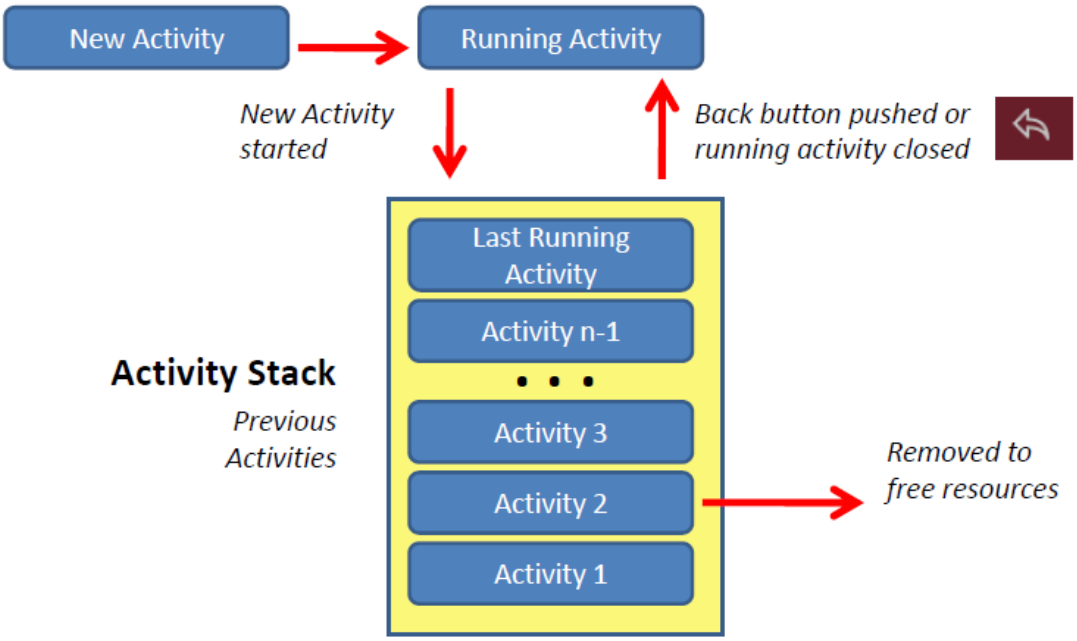


Figura 4.5: Pila de actividades de Android [13]

Clase VistaInicial

Además de contener la parte gráfica se ha encarga de hacer la llamada al AnalizadorDispositivo que prepara el estado del Bluetooth y del teléfono. Mediante diálogos informa si el nivel acústico es bajo y si el nivel de batería es bajo, esta información se la muestra antes de conectar el dispositivo, para que el usuario pueda modificar lo que considere oportuno antes realizar la conexión.

VistaInicial.cs

La implementación de la VistaInicial contiene los siguientes controles, menú y diálogos:

- Dos controles **Label** que se utilizan para dar al usuario información útil. El primero muestra el nombre de la aplicación y el segundo para informar al usuario del tipo de datos (Puerto Serie) de la lista desplegable.

- El control **ComboBox** se utiliza para mostrar el nombre de los puertos serie disponibles en un cuadro combinado desplegable. La parte superior es un cuadro de texto que muestra el elemento seleccionado. La segunda parte es un cuadro de lista que muestra una lista de elementos, de los cuales el usuario puede seleccionar uno.

- El control **StatusBar** muestra información acerca de los pasos que el usuario debe realiza e informa de las operaciones que realiza la aplicación.

-El menú utilizado es **SoftKeys,** aparecen dos softkeys en una barra localizada en la parte inferior de la pantalla del dispositivo. Muestran acciones permitidas al usuario. Gestiona el evento click de los dos elementos del menú.

- Tres diálogos **MessageBox** uno para mostrar la información del dispositivo, otro para informar de un al abrir la conexión y otro para confirmar la petición de salir del sistema y **DialogResult** para recoger la interactuación del usuario.

Figura 4.6: Pantalla de inicio Windows Mobile

Figura 4.7: Diálogo de pantalla de inicio

El manejador del evento Click de la opción "Continuar" llama al AnalizadorDispositivo y si el resultado es correcto llama crea una instancia de VistaPrincipal y la inicia. En caso de error muestra el error mediante un diálogo.

El manejador del evento Click de la opción "Salir" cierra la aplicación.

VistaInicial.java

La implementación de la VistaInicial.java accede al diseño o layout definido previamente en el fichero `inicio.xml`. Este layout es del tipo **RelativeLayout**, que se caracteriza por ser muy flexible y los elementos se posicionan dependiendo de la posición de otros. Además contiene los siguientes widgets o views:

- Dos cuadros de texto de tipo **TextView**, uno con el mensaje de bienvenida y el otro describe lo que tiene que hacer el usuario para continuar.
- Dos botones utilizando la clase **Button**, representan dos acciones que permite esta clase, continuar o salir.

Esta clase gestiona el evento Click mediante **EventListener** de forma anónima.

Por un lado, el manejador asignado al botón Continuar llama al método enciendeBluetooth() y si el resultado es correcto lanza la actividad VistaListaDispositivo con el método `startActivityForResult()` que llama a `onActivityResult()` cuando finaliza la actividad lanzada. Utiliza este método porque es necesario que una vez finalizada, devuelva a VistaInicial, la dirección MAC del dispositivo seleccionado para poder conectarse al dispositivo.

Por otro lado el manejador del botón Salir simplemente cierra la aplicación.

Esta clase también gestiona el evento click del botón hardware atrás, que cierra la aplicación.

Además está clase implementa un diálogo de tipo **`AlertDialog`**, que permite editar el título, texto y el número de botones, este diálogo lo muestra para pedir confirmación de la acción de cerrar la aplicación.

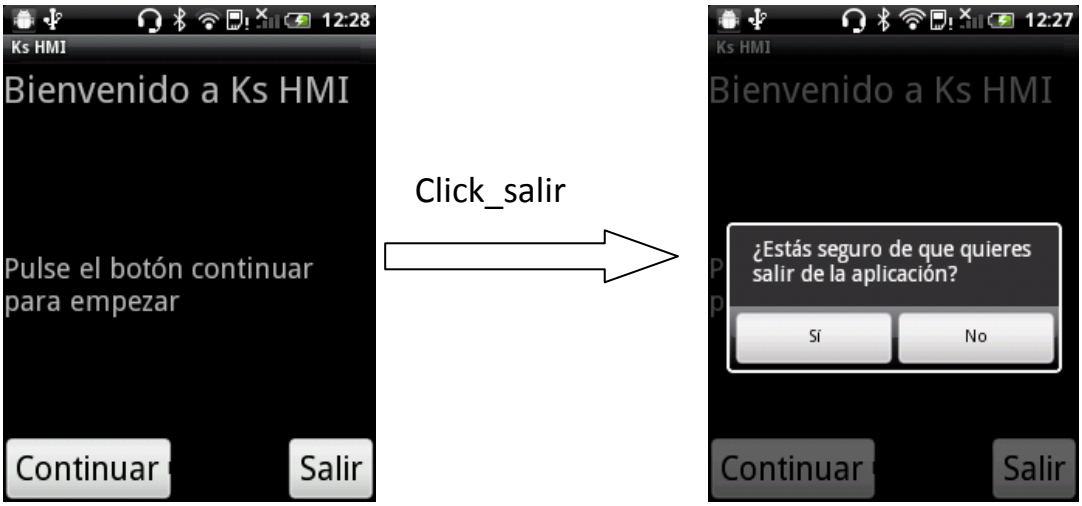


Figura 4.8: La primera pantalla de Android de inicio y la otra es el diálogo de salir de la aplicación

Clase VistaListaDispositivos

Esta clase se encarga de ofrecer al usuario la posibilidad de seleccionar el dispositivo al que desea que el smartphone se conecte. Por un lado mostrará la lista de dispositivos emparejados previamente, por otro lado dispondrá de un botón que inicia la búsqueda de más dispositivos dentro el alcance. Esta clase sólo está implementada para Android porque en Windows Mobile la pila de Bluetooth depende del fabricante, Microsoft o Broadcom (Widcomm) esto imposibilitaría la portabilidad entre fabricantes y tener que utilizar una librería de terceros puede comprometer el funcionamiento de la aplicación.

VistaListaDispositivos.java

Esta clase utiliza el diseño contenido en el fichero **`devicelist.xml`**. Utiliza un layout de tipo **LinearLayout**, que está definido para que apile los elementos verticalmente está estructurado de la siguiente forma:

- Dos `ListView`, estos widgets muestran los elementos en una lista vertical deslizable, los elementos que muestran son los dispositivos emparejados y la otra los dispositivos encontrados.
- Dos `TextView` están colocados a modo de etiqueta encima de cada lista para indicar la diferencia.
- Un `Button` que permite iniciar la búsqueda de dispositivos. El evento asociado a la pulsación del botón está asociado a un EventListener de forma anónima.

Esta clase tiene la peculiaridad que utiliza dos `ArrayAdapters` que gestionan las vistas de los datos que se muestran en las listas, de manera que resulte amigable al usuario.

Además esta clase registra dos **BroadcastReceiver** que detectan y reaccionan uno cuando se descubren nuevos dispositivos y el otro cuando termina la búsqueda dispositivos.

Figura 4.9: Una pantalla muestra dispositivos emparejados y la otra buscando dispositivos dentro del radio de alcance

Clase VistaPrincipal

Esta clase actualiza el estado de somnolencia, indicado desde la capa Lógica de la aplicación, además muestra mensajes de error al usuario, ya sea debido a la pérdida de la conexión, un error en el formato de los datos o alguna otra excepción que requiera el cierre inmediato de la aplicación. Estos mensajes de error conllevan el cierre de la aplicación, por ello se muestran mediante un diálogo que requiere la interacción del usuario, para asegurar que el usuario sabe porqué se cierra la aplicación. Además informa al usuario sobre el estado en el que se encuentra el programa mediante una barra de estado.

Un mapa de bits situado en el centro de la pantalla, y que ocupa gran parte de ella, muestra el estado de somnolencia. Cada uno de los tres estados que distingue el algoritmo está representado por una imagen distinta. Estas imágenes, están previamente diseñadas y están guardadas dentro de la carpeta del programa. La vista de la aplicación muestra únicamente el que corresponde al estado actual.

El framework de cada SO provee una clase **ResourceManager** que permite incluir en el proyecto archivos que no son código fuente (sonido, ficheros, imágenes) en tiempo de desarrollo y utilizarlos en tiempo de ejecución. Para los distintos entornos de desarrollo el **ResourceManager** tiene su modo de gestionarse pero la funcionalidad es la misma tanto para Visual Studio [14] como para el SDK de Android [15].

Por otro lado, para mejorar la experiencia al usuario, cuando la aplicación refresca el estado de somnolencia lo acompaña de un efecto de parpadeo o flash, para que el usuario tenga feed-back aunque el estado no cambie.

La información del estado de usuario la actualizará el hilo que gestiona la recepción y procesado de datos distinto del principal de la aplicación. Ni Android ni Windows Mobile permiten actualizar la UI desde un hilo secundario, porque si dos o más hilos estuvieran manipulando un elemento de la interfaz podrían crearse inconsistencias. Entonces es necesario un sistema para acceder a la UI de forma segura. Cada SO provee al programador de herramientas alternativas para acceder de forma segura a la interfaz desde un hilo secundario. Esta clase es la única que utiliza estas herramientas, de modo que la forma de acceder a la interfaz es transparente a los módulos que no son de la interfaz gráfica.

VistaPrincipal.cs

Implementa una función que teniendo en cuenta el tamaño de la pantalla calcula la posición de los elementos gráficos. De esta forma el indicador está situado en el centro de la pantalla en los distintos dispositivos que ejecutan la aplicación. Esta interfaz contiene los siguientes elementos:

- Menú del mismo tipo que el utilizado en VistaInicio.cs
- **StatusBar,** barra de estado indica si se están perdiendo muestras, si el algoritmo se está incializando o si está mostrando el estado actual de somnolencia.

- **Panel** sirve para agrupar controles y sobre pinta el estado de somnolencia.
- Un elemento **Timer** necesario para prevenir el apagado automático del smartphone lo que conllevaría a una pérdida de conexión en algunos smartphones y el consiguiente fallo de la aplicación. Corre en el mismo thread que la GUI porque así se actualiza correctamente.
- Dos tipos de diálogos **MessageBox,** uno para avisar de fallos de la aplicación que muestra un mensaje que explica la causa del fallo (pérdida conexión, error del formato de los datos, memoria dispositivo insuficiente,…) y otro para confirmar la petición de cerrar la aplicación.

Para pintar la imagen en el formulario utiliza un objeto de la clase Bitmap y otro de la clase Graphics. Primero utiliza el método **CreateGraphics** del Panel para obtener una referencia a un objeto **Graphics** que representa la superficie de dibujo de dicho Panel. Después sobre el objeto Graphics llama al método `Clear(SystemColors.Window)` para conseguir el efecto flash o parpadeo, posteriormente llama a `DrawImage(btm, pie.btmX, pie.btmY);` para dibujar el Btimap en la posición central calculada anteriormente.

A modo de demostración los indicadores que puede mostrar la aplicación son:

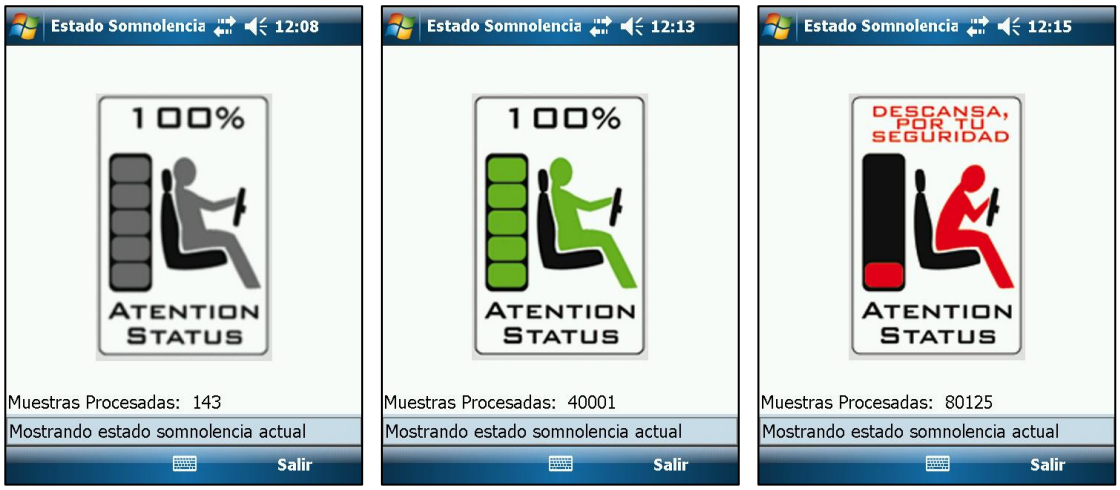


Figura 4.10: Tres pantallas con los tres estados: incialización, estado atento y estado somnolencia crítico en Windows Mobile.

Esta clase implementa los métodos públicos que permiten actualizar los controles de la interfaz gráfica de usuario. Para realizar llamadas seguras utiliza el método Control.Invoke(), como argumento se pasa un delegado y si es necesario una cadena de objetos. El hilo pasa el delegado de forma explícita y el hilo principal de la aplicación, que contiene el identificador del control, ejecuta el método indicado en el delgado. Un delegado es una clase que tiene que tener el mismo prototipo que el método que quiere ejecutarse para modificar la interfaz. El siguiente código es un ejemplo del uso de delegados explícitos en la implementación de la aplicación.

Delegados explícitos:

```
public delegate void TwoIntCallback(int  estadoCalculado,int
numeroMuestras);
private void SetContadoresSemaforos(int estadoCalculado, int
numeroMuestras)
{
      Graphics grpCuadro = this.panel2.CreateGraphics();
      switch (estadoCalculado){
       case -1:
       stbEstadoSistema.Text = "Se está incializando el sistema...";
       lblValorDif.Text = estadoCalculado.ToString();
       lbNumeroMuestras.Text = numeroMuestras.ToString();
       grpCuadro.DrawImage(btmSemaforoInicio, pie.btmX, pie.btmY);
       grpCuadro.Dispose();
       break;

      case 0:
      stbEstadoSistema.Text = "Mostrando estado somnolencia actual";
      lblValorDif.Text = estadoCalculado.ToString();
      lbNumeroMuestras.Text = numeroMuestras.ToString();
      grpCuadro.DrawImage(btmSemaforoVerde, pie.btmX, pie.btmY);
      grpCuadro.Dispose();
      break;

      case 1:
      stbEstadoSistema.Text = "Mostrando estado somnolencia actual";
      lblValorDif.Text = estadoCalculado.ToString();
      lbNumeroMuestras.Text = numeroMuestras.ToString();
      grpCuadro.DrawImage(btmSemaforoRojo, pie.btmX, pie.btmY);
      grpCuadro.Dispose();
      break;
     }
```

En el hilo secundario:

```
private void ActualizarContadoresSemaforo(int estadoCalculado, int
muestrasRecibidas)
{
  if (!this.IsDisposed)
  {
    this.Invoke(new TwoIntCallback( this.SetContadoresSemaforos),
    new object[] { estadoCalculado, muestrasRecibidas });
  }
}
```

VistaPrincipal.java

La implementación de la VistaPrincipal.java accede al layout definido previamente en el fichero vistaresultados.xml. Este layout es del tipo RelativeLayout que contiene los siguientes widgets o views:

- Un ImageView, colocado en el centro de la pantalla, este elemento sirve para mostrar una imagen que tiene asociada. El método setImageResource() de esta clase permite asignarle una imagen. Las imágenes que representa son indicadores de somnolencia que están guardados en la carpeta de recursos"\res\drawable" con el nombre n0.png,n1.png, n5.png, que corresponden fase inicialización, estado despierto, fatiga y dormido respectivamente

- Un TextView, está colocado en la parte inferior de la pantalla a modo de barra de estado indica si se están perdiendo muestras, si el algoritmo se está incializando o si está mostrando el estado actual de somnolencia.

De la misma forma que en la clase VistaInicial.java gestiona el evento click del botón hardware atrás, que cierra la aplicación e implementa diálogo de tipo **`AlertDialog`**, para pedir confirmación de la acción de cerrar la aplicación. Además implementa otro diálogo para avisar de fallos de la aplicación que muestra un mensaje que explica la causa del fallo (pérdida conexión, error del formato de los datos, fallo crítico, etc.)

Provoca un efecto parpadeo intercalando un bitmap negro entre actualizaciones de la imagen de estado de somnolencia, de este modo el usuario tiene sensación de que el programa está funcionando con normalidad.

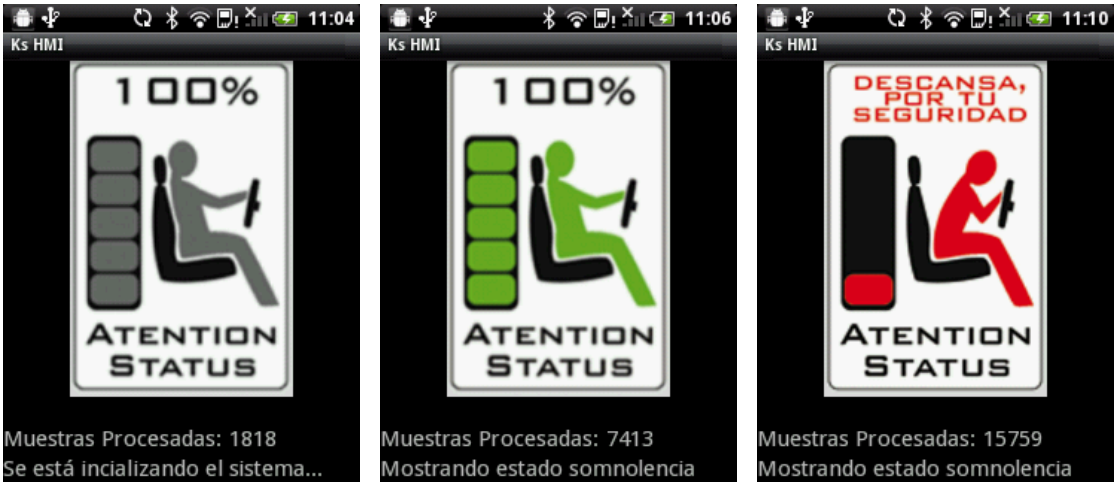


Figura 4.11: Tres pantallas con los tres estados: inicialización, estado atento y estado somnolencia crítico en Android

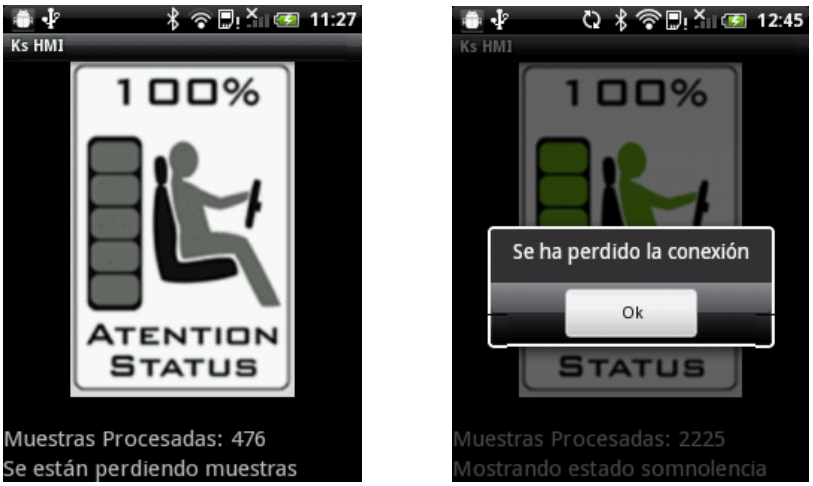


Figura 4.12: Dos pantallas una indica que se están perdiendo muestras y la otra se ha perdido la conexión

Esta clase implementa los métodos públicos para modificar la interfaz gráfica desde un hilo distinto del principal de la aplicación. Para ello define un objeto de la clase Handler que nos provee el framework de Android y sobre ella utiliza los siguientes métodos:

1. Ejecuta `obtainMessage(int what)`: devuelve un objeto de tipo Message, identificado por el parámetro `what`. A este objeto se le

puede añadir información mediante un objeto tipo `Bundle`, que se le añade ejecutando sobre el objeto `Message` el método `setData(Bundle)`. Esta clase lo utiliza para enviar el estado de somnolencia, necesario para actualizar la interfaz.

2. Ejecuta `sendMessage(Message msg)`: envía el mensaje que recibe como parámetro a la cola del Handler.
3. Para procesar los mensajes enviados, implementa handleMessage(Message msg): Allí, el handler ejecuta los métodos que actualizan la interfaz.

4.3.2 Comunicación

Esta clase encapsula la gestión de la conexión Bluetooth con perfil puerto serie. Cuando la tecnología inalámbrica Bluetooth se utiliza para sustituir al cable, se emplea el Perfil de Puerto Serie (SPP, Serial Port Profile) para el canal resultante orientado a conexión. Este perfil está construido sobre el Perfil de Acceso Genérico y define cómo deben configurarse los dispositivos Bluetooth para emular una conexión a través de un cable serie utilizando RFCOMM, un protocolo de transporte sencillo que emula los puertos serie RS-232 entre dispositivos homólogos.

Las aplicaciones ejecutadas en los dispositivos son normalmente aplicaciones heredadas que esperan que la comunicación tenga lugar a través de un cable serie.

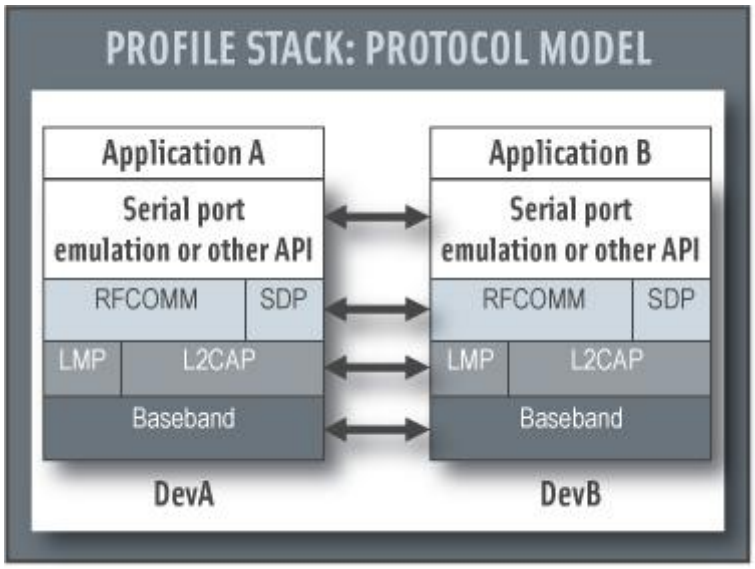


Figura 4.13: Pila de protocolos de Bluetooth del perfil de puerto serie [9].

Las principales diferencias en la implementación para ambas plataformas se deben a las distintas API disponibles según el sistema operativo. Esta clase encapsula las siguientes funciones:

- Abrir la conexión
- Recibir datos de la banda respiratoria:
    - Iniciar transmisión

  - Extraer la información de los datos recibidos
- Detectar fallos de comunicación debidos a:
  - Pérdida de conexión
  - Error en la transmisión

Los métodos de gestión de la comunicación se ejecutan en un hilo específico para que la ejecución de operaciones bloqueantes de E/S no bloquee la interacción con el usuario.

Estas funcionalidades están implementadas para Windows Mobile en la clase ControladorPuertoSerie.cs y en Android en ControladorComunicación.java. La principal diferencia es que en Windows Mobile la comunicación la realiza sobre la clase SerialPort y en Android sobre Sockets. Las diferencias en la implementación de las funcionalidades están comentadas a continuación.

Abrir la conexión

Para Windows Mobile primero obtiene una instancia del SerialPort con el nombre del puerto seleccionado y después intenta abrirlo.

En Android para solicitar la conexión se utiliza un BluetoothSocket y un BluetoothDevice El BluetoothDevice representa al dispositivo remoto al que se conecta y se obtiene a través de la dirección del dispositivo seleccionado. El BluetoothSocket solicita la conexión permitiendo la comunicación punto a punto del móvil con la banda respiratoria enviando y recibiendo información en forma de *streams* de datos.

Si establece la conexión correctamente entonces se inicia la transmisión de datos.

Recibir datos de la banda respiratoria

La aplicación envía comandos a la banda y recibe las muestras de respiración de la banda, en Windows Mobile a través del buffer de escritura y lectura del PuertoSerie que ha abierto y a través del OutputStream o del InputStream del socket que realiza la conexión en Android.

La banda bioplux implementa sobre el protocolo puerto serie, otro protocolo para la transmisión de datos binarios entre los dispositivos. Este protocolo especifica los comandos el móvil debe enviar a la banda para iniciar la transmisión y el entramado de los datos que la banda envía al móvil (Especificaciones disponibles en el Anexo A).

El móvil envía el comando para indicar el inicio de la transmisión “V”, luego espera a recibir una cadena de caracteres y por útil envía otro comando para configurar la frecuencia de muestreo 40 Hz, el número de canales 1 y número de bits por muestra 12: “@START,0040,01,12;"

A partir de ese momento el móvil continuamente lee los bytes que hay por leer, teniendo en cuenta que las muestras llegan en bloques de 4Bytes siempre lee múltiplos de 4bytes para no romper los bloques. Una vez leídos, extrae de cada bloque los 12bits que contienen el valor de la muestra de respiración y después llama al método que procesa los datos de GestiónSistema. El proceso de tratamiento de los datos está explicado en la parte de gestión del sistema.

En Windows Mobile existen dos alternativas para leer los datos del buffer de lectura del SerialPort. La primera utiliza el evento DataReceived recibidos gestionado por el SO y la otra utiliza la llamada bloqueante Read:

- El SO lanza en un thread secundario, el evento DataReceived cada vez que se reciben datos nuevos y llama al manejador definido para ese evento, que lee y trata los datos recibidos. Para leerlos se recomienda el método no bloqueante ReadExisting() y a continuación se tratan los datos. La principal desventaja es que está recomendado no hacer operaciones costosas dentro de este thread, estas operaciones deben ser ejecutadas en un thread distinto, porque si la frecuencia de muestreo de los datos es alta y las operaciones de procesado costos, según el funcionamiento de los threads, se forma una cola de threads lanzados cada vez que llegan nuevos datos esperando a ser atendidos. Esta cola es limitada y al

llegar a un número máximo genera una excepción OutOfMemoryException que provoca un mal funcionamiento de la aplicación.

- Utilizar la llamada bloqueante Read para leer datos, para no bloquear la aplicación es necesario ejecutarla en un thread distinto del principal de la aplicación. Esta alternativa evita crear múltiples threads cada vez que llegan datos, dado que siempre se utiliza el mismo.

La aplicación utiliza la llamada bloqueante Read porque utilizar la primera alternativa conllevaría añadir otro hilo y atender la cola de hilos parados lo que conlleva un descenso del rendimiento y puede provocar excepciones. Además la llamada bloqueante Read sigue el mismo patrón que utiliza Android y permite detectar si la conexión se pierde mediante el uso del timeout.

Recibir datos de la llave USB Bluetooth

La aplicación también puede conectarse a una llave USB Bluetooth, esto está implementado para poder procesar registros de respiración cada vez que se hace un cambio en el algoritmo de respiración o para poder procesar registros de respiración tomados por otras vías.

Lo que cambia con respecto a la recepción vía banda respiratoria es que no se intercambian mensajes para iniciar la transmisión, es decir una vez conectado se inicia la transmisión directamente y que no existe sobre el protocolo de Perfil Puerto Serie otro protocolo para la transmisión de datos binarios entre los dispositivos. El formato de los datos en este caso es simplemente que cada muestra de respiración viene separada de la siguiente por un salto de línea.

Detectar fallos de comunicación

A nivel de comunicación la aplicación detecta la pérdida de la conexión y los errores que se pueden producir en la transmisión de los datos. Ambos casos son críticos para el correcto funcionamiento de la aplicación por eso en caso de que ocurran avisa al usuario.

Pérdida de la conexión

Para detectar la pérdida de la conexión en Android basta con controlar la excepción IOException en la operaciones de Lectura. Sin embargo en Windows Mobile el IOException ocurre en otro caso, para detectar la pérdida de conexión utiliza la propiedad ReadTimeout del SerialPort. Por tanto si se pierde la conexión se agotará el timeout de 30 segundos y producirá la excepción TimeoutException y entonces lo notifica al usuario y se cierra la aplicación.

Error de transmisión

Según las especificaciones de la banda en cada bloque de 4Bytes los primeros 7-bits indican el número de muestra enviada, este número sirve para comprobar si debido a algún error en la transmisión se ha perdido alguna trama. Si al comprobar el número de trama la aplicación detecta que el número de trama no es el correcto avisa al usuario de que se han perdido muestras.

4.3.3 Lógica

Esta capa alberga el ControladorLogica, AvisadorSomonolencia, ProcesadorRespiración, ArchivadorResultados.

Esta capa es donde reside la lógica de la aplicación, es la más independiente del SO y la clase ControladorLogica hace la función de enlace entre la GUI y la comunicación.

ControladorLogica

Esta clase desempeña dos funciones:

- Capa intermedia de la aplicación: hace de pasarela entra entre la GUI y la comunicación. Para hacerlas independientes una de la otra.
- Por otro lado este nivel contiene la parte más independiente del SO: procesado, registro de datos y gestión de avisos y esta clase sirve de enlace para comunicarlos entre ellos y con las capas más dependientes la de comunicación y GUI.

Por tanto para comunicar la GUI con la capa de comunicación implementa los métodos:

- iniciarConexionProcesado (string deviceName) que permite al usuario iniciar la transmisión, cuando selecciona el dispositivo o el puerto.
- mostrarResultado(int nMuestrasPerdidas) sirve para mostrar al usuario el estado de somnolencia y EnviarMensajeConexionPerdida( ) avisarle de fallos que ocurran en la comunicación

El método mostrarResultado(int nMuestrasPerdidas) sirve de para separar la parte de comunicación de la GUI y llamar al AvisadorSomnolencia.

El método de tratarMuestraObtenida() es el que llama a los métodos de los módulos de procesado y registro implementados, separando la capa de comunicación de los módulos de procesado y registro.

Para cerrar la aplicación debido a la petición del usuario la vista llamará a la función del controlador parar cerrar el sistema. Una vez cerrado el sistema, volverá a la parte de la vista para cerrar la aplicación.

En caso de error en el sistema se parará y luego volverá a la vista para avisarlo y cerrar la aplicación. El Controlador cierra la conexión y libera recursos.

ProcesadorRespiracion

Esta clase implementa el algoritmo de detección de somnolencia a partir de la señal de respiración. Tiene como atributos los campos necesarios para el algoritmo. Contiene también métodos para consultar esos campos y poder pasarlos a la GUI y registrarlos en un archivo a través del ControladorLogica.

El algoritmo ks tiene tres fases: una primera de inicialización de los filtros, la segunda de inicialización de los parámetros y la tercera de cálculo del índice ks que indica el estado de somnolencia.

La fase de inicialización de los filtros sirve para eliminar el transitorio de los filtros. Los filtros utilizados son un filtro Butterworth de orden 2 paso bajo de frecuencia de corte 1 Hz y un filtro Butterworth de orden 2 paso alto de frecuencia de 0.1Hz. El tiempo que tarda la en establecerse dentro de una banda alrededor del valor final, t_s, depende de la amplitud de dicha banda, considerando una banda del ±5%:

$t_s \approx 3/\sigma = 3/\zeta\omega_n$ donde $\zeta = 1/\sqrt{2}$ y $\omega_a = \omega_n(1-\zeta^2)^{1/2}$ [16]

En nuestro caso: $\omega_a = 2*\pi *0.1$ $t_s \cong 5$

Para asegurarnos de que se ha eliminado el transitorio tomamos un tiempo cuatro veces mayor de 20 segundos.

La fase de inicialización puede durar un máximo de 15 minutos. Esta fase sirve para elegir un tramo de la señal como ventana de referencia. Analiza la señal por tramos con una ventana deslizante de 40 segundos, sobre la

que se calcula el índice de heteroestacidad de la señal. Se toma como referencia aquella que a lo largo de los 15 minutos tiene un índice de heteroestacidad menor o si tiene un índice menor que 0.03. Una vez tenemos la ventana de referencia, se gira la señal si el máximo absoluto es negativo para que el máximo absoluto sea positivo. Sobre la ventana de referencia se busca el umbral de detección de respiración, que es el valor de la muestra respiración que es el percentil 60% de amplitud, este umbral llamado threshold sirve para detectar los períodos de los ciclos de respiración. Además calcula el factor de normalización como la media de las absolutas diferencias de los períodos de respiración detectados en la ventana de referencia. Los períodos se filtran con un filtro de media móvil de 4 coeficientes.

La tercera fase es el cálculo del índice ks, cada vez que se detecta un ciclo respiratorio se calcula un nuevo índice. Primero se filtra el período del ciclo de respiración con un filtro de media móvil de 4 coeficientes y se calcula el índice como la media de las últimas 10 absolutas diferencias de los períodos detectados dividido por el factor de normalización.

El algoritmo define tres estados de somnolencia asociados al índice Ks calculado. Un estado indica que el algoritmo está en fase de inicialización, por tanto el índice no se ha de tomar en cuenta, y los otros dos estados que indican si está o no en estado de somnolencia crítico.

Para decidir si el estado de somnolencia es crítico o no se utiliza una ventana de valores del índice ks de 5 minutos, se dará alarma si el índice ks es mayor que 6 en la mayoría de valores de la ventana de 5 minutos.

El algoritmo define tres estados de somnolencia asociados al índice Ks calculado. Un estado indica que el algoritmo está en fase de inicialización, por tanto el índice no se ha de tomar en cuenta, y los otros dos estados que indican si está o no en estado de somnolencia crítico.

Para decidir si el estado de somnolencia es crítico o no se utiliza una ventana de valores del índice ks de 5 minutos, se dará alarma si el índice ks es mayor que 6 en la mayoría de valores de la ventana de 5 minutos.

Los cambios en la implementación para Windows Mobile y Android son los debidos a la sintaxis del lenguaje de programación utilizado C# y JAVA respectivamente. En concreto la diferencia es para declarar el tipo booleano en Java utiliza Boolean y en C# bool.

ArchivadorResultados

El sistema de archivos primero crea la carpeta KS_HMI donde guarda en cuatro archivos las muestras recibidas y los resultado obtenidos al procesar los datos. Estos archivos no son generados para aportar información al usuario sino que en la fase de pruebas de la aplicación, aportar información a los desarrolladores sobre el funcionamiento de la aplicación y posteriormente poder validar el funcionamiento de aplicación

Es importante saber cuánta memoria puede llegar a ocupar los archivos de texto a lo largo de tiempo de ejecución del programa para ello hay que tener en cuenta como aumenta el número de caracteres necesarios para representar los valores. Por ello hemos realizado un cálculo aproximado del tamaño que pueden llegar a tener cada uno de los archivos.

El nombre del primer archivo es TestKS_HMI_Inicializacion yyyy_MM_dd_HH_mm.txt y contiene en cada línea, separados por un espacio en blanco, siete campos que contienen información para comprobar la correcta inicialización del algoritmo:

1. Número de muestra procesada es de tipo entero positivo: ocupa log(40*n) bytes donde n son los segundos. En un registro de 6horas el valor máximo de muestras sería 36800 con lo que ocupa 5 Bytes
2. Valor del threshold ocupa 3 bytes
3. Valor del factor de normalización ocupa 3 bytes
4. Valor del tiempo inicial de la ventana de referencia ocupa 4 bytes
5. Valor del tiempo final de la ventana de referencia ocupa 4 bytes

6. Número de pasos por umbral en la ventana de referencia ocupa 2 bytes
7. Valor del índice de heteroestacidad ocupa 11 bytes.

Además se a de contar 6 bytes para los espacios en blanco de separación entre los valores. Como sólo se escribe en este fichero durante la inicialización el tamaño que puede ocupar se calcula como 38B/muestras*36800 muestras= 1398400Bytes=1,4MBytes. Este es un tamaño aceptable teniendo en cuenta que la capacidad del smartphone es del orden de Gigabytes

El nombre del segundo archivo es TestKS_HMI_Variables yyyy_MM_dd_HH_mm.txt y contiene en cada línea, separados por un espacio en blanco, cinco campos para comprobar el correcto cálculo del índice Ks:

1. Número de muestra procesada es de tipo entero positivo: ocupa log(40*n) bytes donde n son los segundos. En un registro de 6horas el valor máximo de muestras sería 864000 con lo que ocupa 6 bytes
2. Valor de la muestra recibida es entero positivo de valor máximo 4096 ocupa 4 bytes
3. Valor de la muestra filtrada es de tipo float y puede tomar valores negativos 10 bytes
4. Valor del índice KS calculado es de tipo float y solo toma valores positivos por el algoritmo 9 bytes
5. Número de ciclos de respiración detectados es de tipo entero 4 bytes

Tenemos que sumar los 4 espacios en blanco de separación de cada campo.

En seis horas serían 37B/muestra*40muestras/s*3600*6= 31968000B =31,97MB en seis horas. Este es un tamaño tolerable para hacer pruebas.

El nombre del tercer archivo es TestKS_HMI_Pasos_Umbral yyyy_MM_dd_HH_mm.txt y contiene en cada línea, los valores de los períodos de respiración detectados en la ventana de referencia y en laútlima línea el número de pasos por el umbral detectados. Este archivo sirve para ver que los períodos sobre los cuales se calcula el factor de normalización son los mismos en la versión MATLAB que en la versión móvil. Como sólo contiene los valores de los periodos ocupa 1KB y por tanto es aceptable.

El nombre del cuarto archivo es TestKS_HMI_Decision_Alarma yyyy_MM_dd_HH_mm.txt contiene información sobre si se ha dado alarma en una ventana de 5 minutos. Para ello contiene en cada línea, separados por un espacio el valor de la suma de los periodos que suman 5 minutos que ocupan 5Bytes y un booleano que dice si da alarma o no ocupa 5Bytes. Para calcular el tamaño total del fichero tomamos como peor caso que los periodos duran 100 muestras. 11Bytes/ciclo*40muestras/s*3600s/h*6h/(100muestras/ciclo)=95040B el tamaño que ocupa el fichero son 0,95MB en 6 horas. Tamaño tolerable para pruebas.

El sistema de archivos de cada SO tienes sus peculiaridades. En Android es necesario guardarlo en un dispositivo externo. Porque de lo contrario esos datos solo pueden ser accedidos desde la aplicación y no pueden ser descargados. En caso de no haber SDCARD no se guardan los datos. En Windows Mobile guarda los resultados, en caso de que haya, en la SDCARD y si no en el directorio myDocuments, que también es accesible desde fuera de la aplicación.

Para un mejor rendimiento es mejor evitar el uso arrays multidimensionales [17], por ello se utilizan arrays unidimensionales para almacenar los valores a escribir.

Están implementadas dos funciones:

1. Crear fichero: Al iniciar la conexión se crea un fichero de texto donde se escriben los que se van obteniendo mientras dure la conexión. En Windows Mobile utiliza la API `Directory` y en Android la API `File`

2. Escribir fichero: Cada vez que lo pide la el control del sistema escribe un líneas de texto con los valores separadas por salto de línea por cada índice de los arrays. En Windows Mobile para escribir en un fichero utiliza un objeto de la clase `StreamWriter` y en Android un objeto de la clase FileWriter.

Avisador Somnolencia

La aplicación envía un aviso cuando detecta que el nivel de somnolencia calculado es crítico. La aplicación implementa un aviso acústico dirigido al usuario y envía de un SMS a un teléfono predeterminado.

Aviso Acústico

Cuando la aplicación detecta que el nivel de somnolencia es crítico inicia la reproducción continua de un sonido estridente hasta que el nivel de somnolencia deje de ser crítico.

Utilizamos un sonido estridente con licencia CreativeCommons, de corta duración [18]. La aplicación reproduce el sonido en otro hilo porque queremos que el sistema siga recogiendo y procesando datos mientras se reproduce el sonido

En Android a través del ResourceManager hemos de importar el sonido que queremos reproducir como aviso, el formato del archivo de audio es mp3. En Windows Mobile también se importa a través del ResourceManager pero el archivo debe ser .wav.

Para el manejo de archivo de audio en C# se utiliza la API SoundPlayer y en Android MediaPlayer, el funcionamiento de ambas es muy parecido.

Los pasos para reproducir el sonido son:

1. Cargar el archivo que se irá reproduciendo continuamente:

Android:

```
player = MediaPlayer.create(appContext,R.raw.beep_4);
player.setLooping(true);
```

Windows Mobile:

```
Stream audioStream = new
MemoryStream(Properties.Resources.beep_6);
player = new SoundPlayer(audioStream);
player.Load();
```

2. Reproducir el sonido continuamente:
   En Android `player.start()` y en Windows Mobile
   `player.PlayLooping()`

3. Parar el sonido y recargarlo para la próxima vez:
   Android:

```
player.stop();
player = MediaPlayer.create(getApplicationContext(),
R.raw.beep_4);
player.setLooping(true);
```

   Windows Mobile:

```
player.Stop();
player.LoadAsync();
```

Enviar SMS

En caso de estado de somnolencia critico. El sistema envía un SMS si el teléfono está activo. En Windows Mobile para comprobar si el teléfono está activo se accede mediante la API SystemState de Compact Framework y en Android la API's ServiceState y TelephonyManager.

Para enviar un mensaje utiliza en C# SmsManager y en Android SmsMesage.

4.3.4 Configurar Dispositivo

Antes de iniciar todo el sistema de comunicación, procesado y aviso. La aplicación configura en la medida de sus posibilidades que el dispositivo cumpla con las condiciones necesarias para el correcto funcionamiento

del sistema. Por tanto si es necesario inicia el Bluetooth en Android y Windows Mobile, teléfono, sonido y la batería sólo en Windows Mobile. En Windows Mobile he creado una clase llamada AnalizadorDispositivo que contiene los métodos para realizar estas configuraciones y obtener información del Hardware. Los métodos de esta clase son llamados desde la clase VistaInicial.

Configuración del Bluetooth

Antes de que cualquier aplicación pueda utilizar la infraestructura Bluetooth debe hacer dos cosas: asegurarse de que el dispositivo sobre el que está ejecutando tiene soporte para Bluetooth y si así es comprobar que está activado, activándolo si fuese necesario.

En Windows Mobile resulta que en la ROM del dispositivo existe una dll no documentada llamada **ossvcs.dll**, que exporta funciones para controlar los Dispositivos de Radio (*Radio Devices*). Es decir, los módulos de Bluetooth, Teléfono y Red Inalámbrica (WiFi). **Ossvcs.dll** exporta las funciones por un número de orden y no por su nombre. Estas funciones pertenecen a un API no administrado, por lo que para invocarlas desde nuestro código C# necesitaremos declarar los prototipos mediante p/Invoke.

He utilizado una clase llamada MobileRadio[19], que expone un único método público estático: SetDeviceState(). Este método acepta dos argumentos de tipo entero. El primero indica el tipo de dispositivo, y el segundo informa el estado al que se desea cambiar. El tipo de dispositivo puede ser uno de los siguientes: Bluetooth, el Teléfono o la Red Inalámbrica (WiFi). No se necesita conocer de memoria los valores enteros que se pueden traspasar. La clase MobileRadio exporta también dos estructuras (RADIODEVTYPE y RADIODEVSTATE) que contienen las constantes necesarias.

Por tanto para encender el Bluetooth, comprueba a través del API SystemState si está encendido y en caso de no estarlo lo enciende llamando al método estático

```
MobileRadio.SetDeviceState(MobileRadio.RADIODEVTYPE.BLUETOOTH,
MobileRadio.RADIODEVSTATE.ON)
```

Además muestra al usuario un mensaje avisando de que se asegure que los dispositivos estén emparejados previamente.

En Android para poder acceder a la API Bluetooth es necesario declarar al menos uno de los permisos dentro del *manifest* de nuestra aplicación. Los permisos disponibles son dos:

- BLUETOOTH, que permite únicamente realizar conexiones Bluetooth y transferir datos.
- BLUETOOTH_ADMIN, que permite, además de realizar conexiones Bluetooth y transferencias de datos, manipular las opciones del sistema en lo referente a Bluetooth, buscar otros dispositivos y realizar vínculos.

Estas tareas básicas de comprobación y configuración son llevadas a cabo por la clase *BluetoothAdapter*. Los pasos a seguir se resumen en obtener el *BluetoothAdapter* que representará a nuestro dispositivo y activar el sistema Bluetooth.

Configuración del Teléfono

Como el sistema tiene que avisar enviando un SMS en caso de somnolencia. Por tanto es necesario comprobar que el dispositivo tenga teléfono, si lo tiene apagado avisar de que lo va a encender y mirar si tiene la SIM.

En Windows Mobile comprueba que el dispositivo tenga teléfono y si no está encendido. En ese caso lo enciende utilizando la misma clase MobileRadio, el mismo método que para Bluetooth pero cambiando los parámetros para que encienda el teléfono

En Android no se puede encender el teléfono por software pero sí se puede consultar el estado del teléfono. El AvisadorSomnolencia utiliza las APIs de TelephonyManger y ServiceState para comprobar si tiene tarjeta

SIM y si el móvil está en estado de servicio, guarda esta información para enviar o no un SMS cuando el estado de somnolencia lo requiera.

Configuración de la Batería

La aplicación debe permanecer activa hasta que el usuario decida cerrar la aplicación. Para disminuir consumo de batería, los dispositivos móviles pueden estar configurados para apagarse si durante un intervalo de tiempo el usuario no realiza ninguna acción sobre el dispositivo. En ese caso pasan a un estado de hibernación, en estado el sistema operativo puede cerrar las conexiones. Windows Mobile y Android gestionan la batería de forma distinta.

En Windows Mobile si el usuario esta un tiempo sin accionar se apaga el dispositivo y con ello la aplicación finaliza. Por ello la aplicación lee del registro del sistema el tiempo de hibernación y utilizamos ese tiempo para programar un timer, que realiza una llamada al método **SystemIdleTimerReset()** que previene el apagado automático[20].

En Android cuando pasa un tiempo sin que el usuario accione la pantalla o algún botón se apaga la pantalla y la aplicación sigue funcionando.

5. Pruebas de validación del sistema

Las pruebas de validación se han desarrollado conectando el smartphone a un sistema de medida de la señal respiratoria basado en una banda inductiva (ref. anexo A).

Como existe una versión previa del sistema de detección de somnolencia que funciona en PC y que ya ha sido validado, se realizarán una serie de medidas para comprobar que los resultados obtenidos con el smartphone son similares a los obtenidos con la aplicación del PC.

La principal problemática para la validación es que la versión del PC, escrita en MATLAB, no funciona en tiempo real, por ello calcula el índice Ks teniendo en cuenta todo el registro respiratorio. Sin embargo en la versión móvil al ejecutarse en tiempo real, con las primeras muestras se calculan unos parámetros de inicialización y una vez calculados se empieza a calcular el índice Ks.

5.1 Despliegue de la aplicación

En Android el entorno de desarrollo permite generar a partir de un proyecto un archivo .apk, que se instala en el smartphone a través del programa del ordenador para sincronizar el smartphone y el ordenador o directamente desde el smartphone. En Windows Mobile el entorno de desarrollo permite generar un archivo .cab, que se instala desde el smartphone, para esta aplicación es necesario es necesario tener instalado el Compact Framework .NET 3.5.

5.2 Pruebas realizadas

Las pruebas consisten en dejar correr la aplicación durante un tiempo y verificar la correcta detección de somnolencia. La aplicación genera cuatro archivos de texto con las variables necesarias para comprobar el funcionamiento de la aplicación. Los datos están analizados usando el programa MATLAB para ver en una gráfica los datos recibidos y procesados a lo largo del tiempo.

La primera campaña de medidas realizada consiste en una serie de medidas de corta duración a 7 personas, con la finalidad de comprobar la correcta inicialización del algoritmo. Comprobando que los parámetros de inicialización calculados por la versión móvil y la del PC eran semejantes. Esto ha servido para acabar de ajustar el algoritmo del móvil con la finalidad de obtener los mismos resultados que en la versión MATLAB del PC.

A continuación se muestra un gráfico con la estadística de las pruebas de inicialización realizadas:

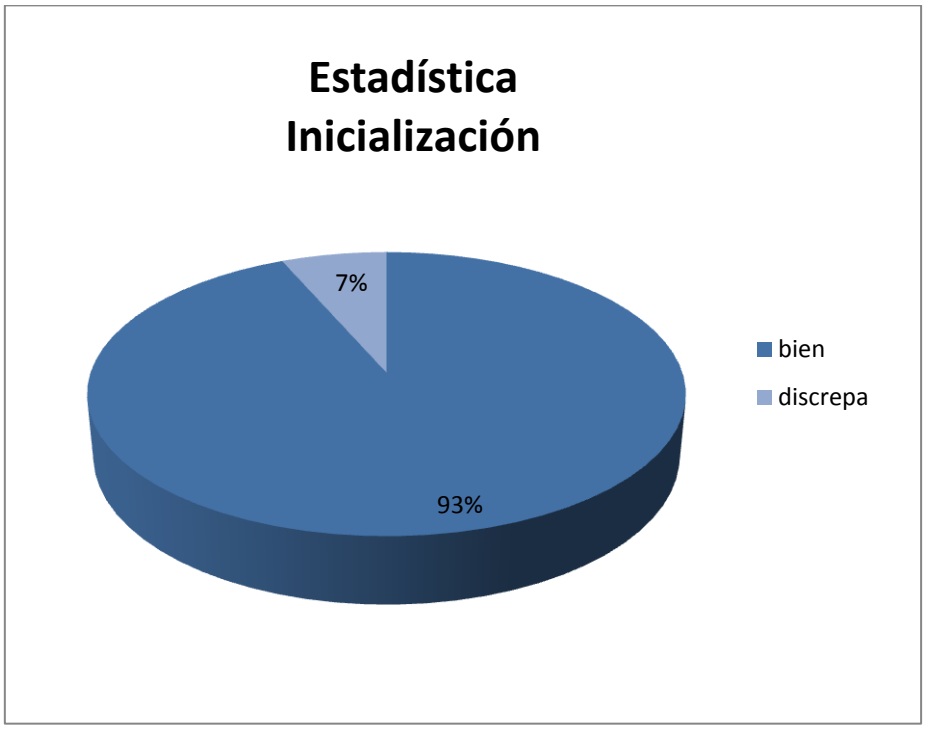


Figura 5.1: Estadística de la pruebas de inicialización

De un total de 76 pruebas realizadas, en el 93% de los casos los resultados son semejantes en la versión móvil y en la versión PC y se produce discrepancia en un 7% de las pruebas. Estas discrepancias no afectan sustancialmente al posterior cálculo del índice Ks.

Las pruebas están realizadas con distintas personas y la banda respiratoria colocada en la zona pectoral y centrada. Además de verificar que los parámetros de inicialización del algoritmo son correctos y hay que

verificar que los resultados de detección de somnolencia también son correctos.

Para ello hay que comparar los valores del índice Ks calculados por el móvil con los que calcula la versión PC en MATLAB. A continuación se muestran tres gráficos donde se ve el valor del índice Ks en pruebas de larga duración realizadas a cuatro personas distintas.

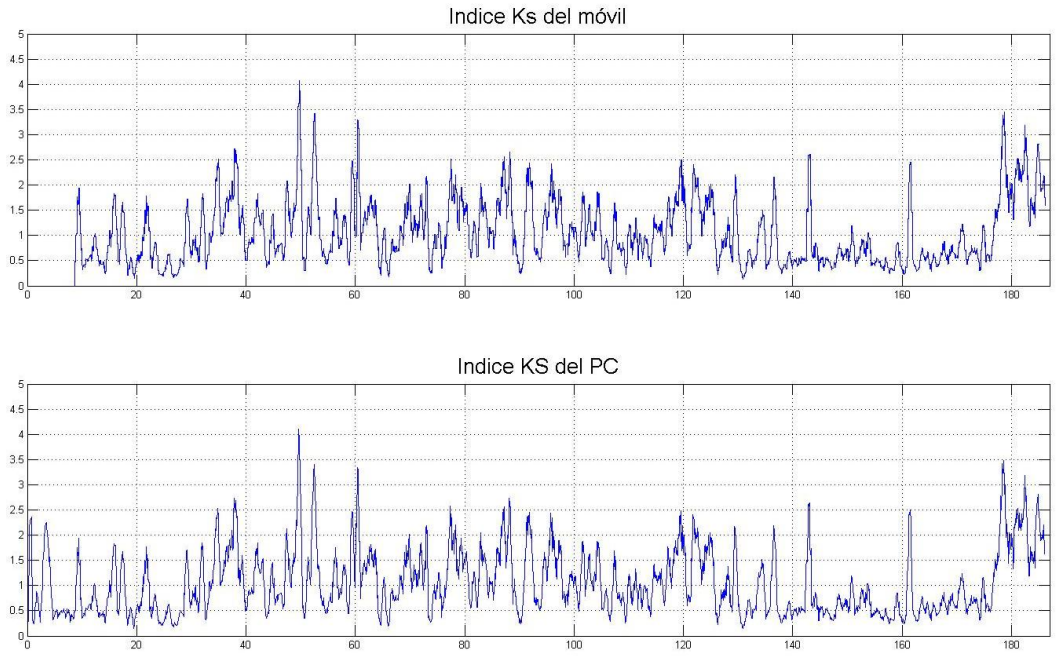


Figura 5.2: Evolución a lo largo del tiempo del índice Ks calculado por el móvil y por el PC sujeto 1

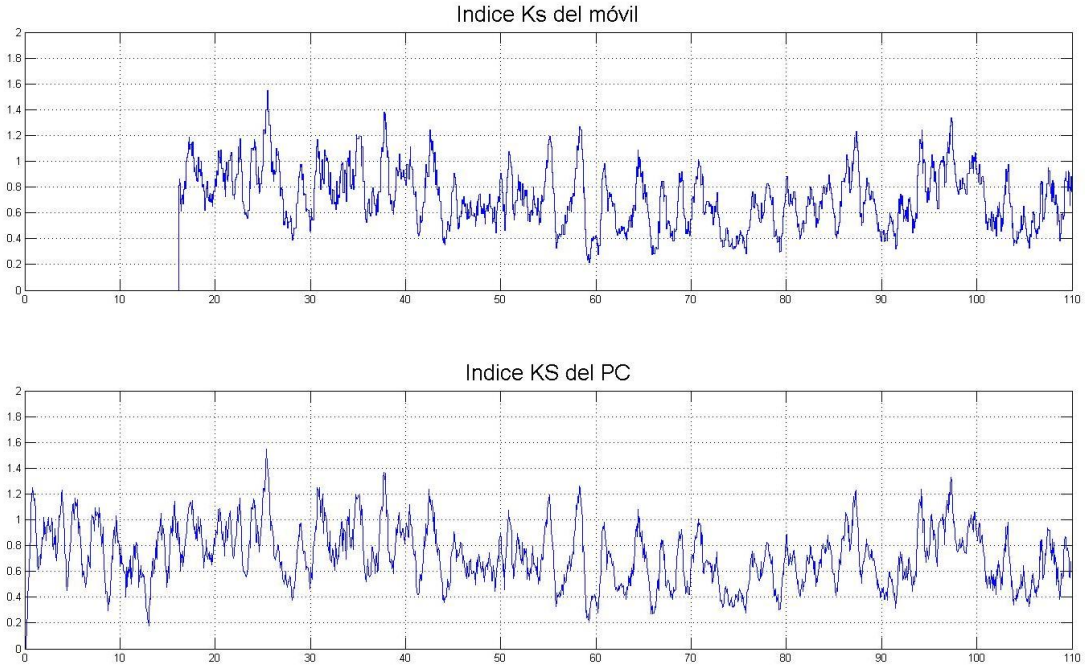


Figura 5.3: Evolución a lo largo del tiempo del índice Ks calculado por el móvil y por el PC sujeto 2

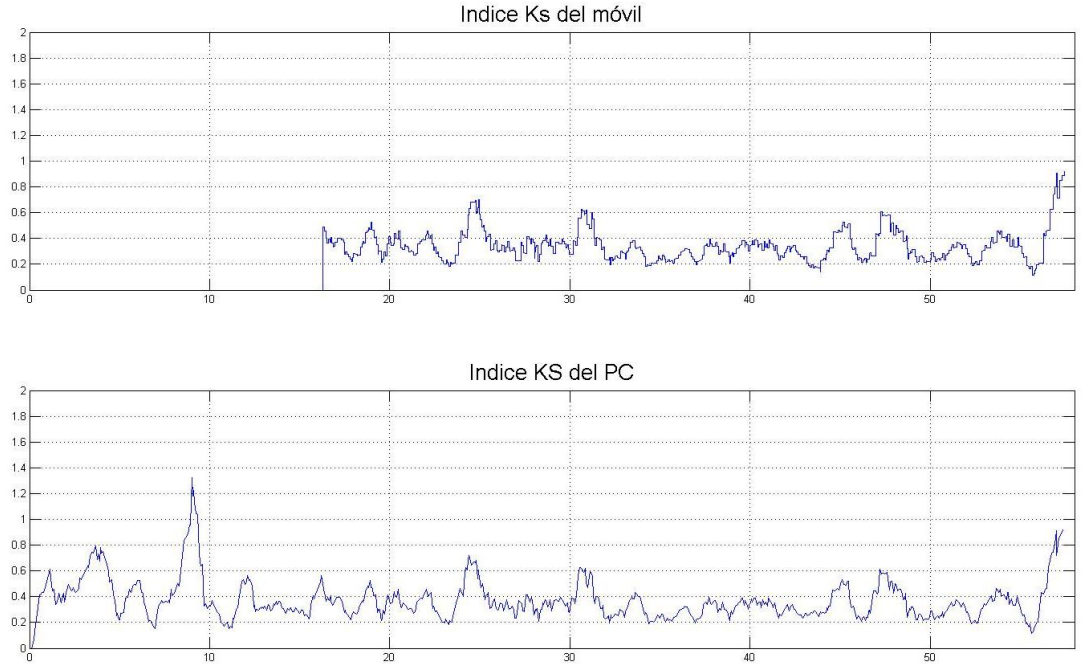


Figura 5.4 Evolución a lo largo del tiempo del índice Ks calculado por el móvil y por el PC sujeto 3

El eje y indica el valor del índice Ks y el eje x el tiempo transcurrido en minutos. Se puede apreciar que la diferencia en los valores del índice Ks en los primeros minutos corresponde al tramo de inicialización del algoritmo. La aplicación móvil, al ser en tiempo real, durante la fase de inicialización no calcula el índice Ks. El programa de PC está escrito en MATLAB, al no ser en tiempo real sí que calcula el índice Ks de todo el registro porque primero calcula los parámetros de inicialización y luego calcula el índice Ks de todo el registro de respiración.

También hemos realizado alguna prueba con somnolencia, el valor del índice ks es superior a 6. En el siguiente gráfico podemos ver las gráficas de evolución del índice ks a lo largo del tiempo tanto en la versión móvil como en PC.

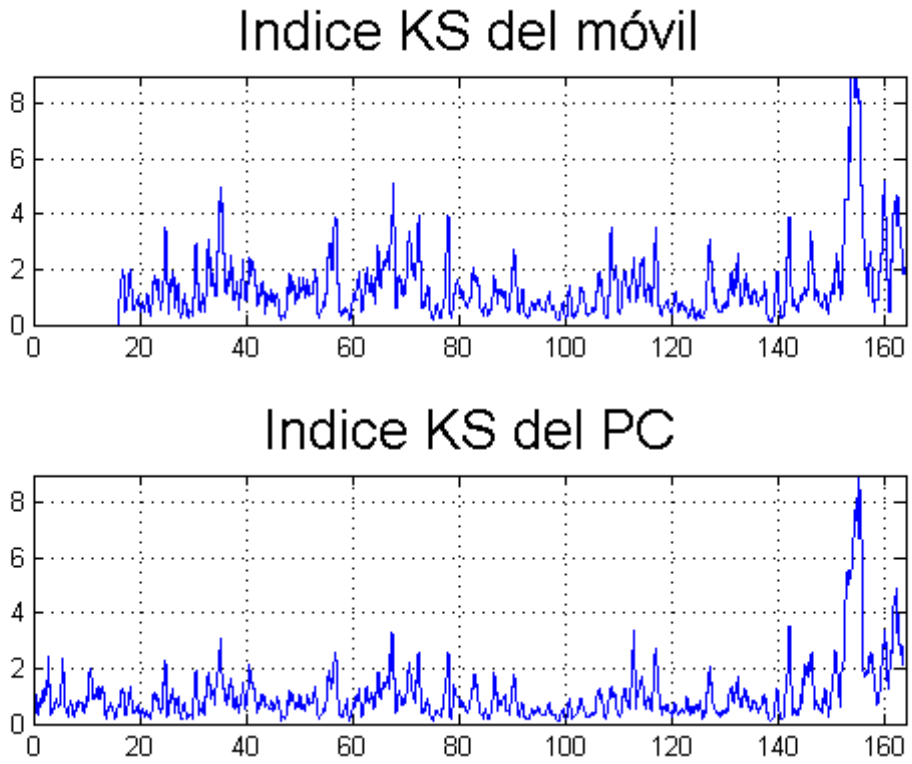


Figura 5.5 Evolución a lo largo del tiempo del índice Ks calculado por el móvil y por el PC sujeto 4 con detección de somnolencia

6. Conclusiones

Se ha conseguido desarrollar una aplicación que adquiere la señal proveniente de un sistema de medida de la respiración, y la procesa en tiempo real para la detección de somnolencia en conductores.

La portabilidad de la aplicación entre sistemas operativos es bastante elevada aunque presenta algunas limitaciones. A la hora de implementar la aplicación para ambas plataformas Windows Mobile y Android ha habido que reescribir la aplicación, las partes que más diferencias presentan son las de la interfaz gráfica de usuario y la parte de comunicación Bluetooth. El resto de partes del sistema presentan las diferencias debidas a la sintaxis de C# y JAVA y al funcionamiento de las API's ofrecidas por cada sistema operativo.

6.1 Líneas de trabajo futuras

Al realizar las pruebas con distintos sujetos nos dimos cuenta de la conveniencia de añadir en el futuro un algoritmo de evaluación de calidad de la señal. Este algoritmo serviría para detectar si la persona se ha colocado bien o mal la banda respiratoria o si la persona está hablando.

Esta aplicación es la base de un proyecto de mayor envergadura que tiene por objetivo medir la adherencia a hábitos de vida saludables de distintos segmentos de población, en el cual se tendrán que medir y procesar todo tipo de señales con los propios sensores del smartphone (actividad) o con sensores externos (variabilidad del ritmo cardíaco, respiración) que enviarán la información a través del Bluetooth. Posteriormente se enviará el resultado a una estación central para su almacenamiento y el análisis multivariable posterior.

7. Anexos

Anexo A. Especificaciones de la banda respiratoria Bioplux

BioPlux Protocol (version 2)

The bioPlux protocol is a binary communication protocol between the bioPlux device and the host. The protocol is typically implemented over a Bluetooth virtual serial port, but it can be implemented over any other binary communication medium. The protocol consists in the commands from the host to the bioPlux, and the data from the bioPlux to the host.

Commands

The commands are the following (expressed by their ASCII character):

V – request for a string with the firmware version Z – start the acquisition mode at 1000 Hz, 6/8 channels, 12-bit samples R – stop the acquisition mode > – set digital output to high < – set digital output to low @ – beginning of extended command string (version 2 only)

The extended command string is available in version 2 only. This string has the format: @command[,parameters];

The string always begins with the character '@' and always ends with the character ';' and has no terminating null character. Following the beginning character, the 'command' field currently can have only one possibility:

– START : start the acquisition mode This command requires a 'parameters' field with the format: FFFF,CC,BB where:

FFFF is the requested sampling frequency in Hz and it is represented as an integer with four decimal digits. This parameter must be between 36 Hz and 1000 Hz.
CC is a bit-mask for selecting which analog channels will be sampled and it is represented as two hexadecimal digits. This parameter is one byte where the LSB refers to channel 1 and the MSB refers to channel 8. If a bit is set, the corresponding channel will be sampled.
BB is the sample resolution, i.e., the number of bits in each sample and it is represented as two decimal digits. The possible values are 8 and 12.

As an example, the string '@START,0060,0F,12;' starts the acquisition of channels 1 to 4 at 60 Hz and using 12-bit resolution.

The acquisition mode is stopped with the 'R' command.

Data

The data sent from the bioPlux device can be of two types:
Response to 'V' command. The bioPlux device should respond with the string: "BioPlux EMG v.*X*\r\n" (\r and \n are bytes CR (decimal 13) and LF (decimal 10), no quotes and no null-terminating character). The character '*X*' represents the protocol version number, and currently

can be 1 or 2 only.

During acquisition mode, the bioPlux device sends a data frame with the format and at the frequency defined by the command that started the acquisition mode.

The data frame sent by the bioPlux device contains the following information:

7-bit frame sequence number;
1-bit digital input state;
analog sampled values;
8-bit CRC (polynomial $x^8 + x^2 + x + 1$) calculated for the whole frame except the CRC.

The frame sequence number is an incrementing 7-bit unsigned integer running from 0 to 127, and then overflowing to 0. The host should verify this number, since a difference of more than one unit between two consecutive received frames (correcting the overflow from 127 to 0) means that there was some missing sent frames between these two received frames. This loss of frames can occur near the limit of the Bluetooth transmission range.

The frame sequence number and the digital input are packed in one byte. The sequence number occupies the lower 7 bits and the digital input is placed in the MSB.

The analog sampled values block can have 8-bit or 12-bit resolution. In 8-bit mode, each byte in this block is the sampled value of each selected analog channel, from channel 1 to channel 8. Each value is an unsigned integer, ranging from 0 to 255.

In 12-bit mode, each value is an unsigned integer, ranging from 0 to 4095. In the analog sampled values block, every two 12-bit values are packed in 3 bytes (24 bits), according to the following scheme:

Byte 0 (bits 7:0) – value A (bits 7:0) Byte 1 (bits 3:0) – value A (bits 11:8) Byte 1 (bits 7:4) – value B (bits 3:0) Byte 2 (bits 7:0) – value B (bits 11:4)

The order of the selected channels is the same as in the 8-bit mode. If the number of selected channels is odd, the value of the last selected channel is transmitted as the first value of the values pair described above, but since there is no second value, only two bytes are transmitted and the upper 4 bits of the second byte are filled with zeroes.
If the acquisition mode was started with 'Z' command, the first six channels are sampled every 1 ms (sampling rate of 1000 Hz) and the last two channels are sampled every 8 ms (sampling rate of 125 Hz), so the sample values of the last two channels are present only in one out of eight data frames. To accomplish this requirement, if the frame sequence number is multiple of eight, the frame has all the eight channels, otherwise it has only the first six channels. In this mode, the format of the data frame is the following:
Byte 0 - bits 6:0 – frame sequence number bit 7 – digital input state
Byte 1 and byte 2 (3:0) – channel 0
Byte 2 (7:4) and byte 3 – channel 1
Byte 4 and byte 5 (3:0) – channel 2
Byte 5 (7:4) and byte 6 – channel 3
Byte 7 and byte 8 (3:0) – channel 4
Byte 8 (7:4) and byte 9 – channel 5
If the frame sequence number is multiple of 8 :
Byte 10 and byte 11 (3:0) – channel 6
Byte 11 (7:4) and byte 12 – channel 7
Byte 13 – CRC
Otherwise :
Byte 10 – CRC

Notes

Before sending the first command to the bioPlux device, the host should send a 'V' command and wait for the string response, ignoring the bytes possibly received before the waited string. This also guarantees to the host that the bioPlux device has the proper firmware version and is functioning properly. Due to latency in the communications channel, the arriving data frames do not cease immediately after the 'R' command. The host should send a 'V' command and wait for the string response, flushing the data frames received meanwhile from the input buffer. This way, the input buffer is kept empty to receive responses to subsequent commands.

Anexo B. Código fuente de la aplicación Android

Contenido del fichero VistaInicial.java

```
package vista;

import upc.KS_HMI.R;
import android.app.Activity;
import android.app.AlertDialog;
import android.bluetooth.BluetoothAdapter;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class VistaInicial extends Activity implements OnClickListener {
	/** Called when the activity is first created. */

	private Button btnSalir, btnConectar;
	private final int IPC_ID = 1122;
	// Intent request codes
	public static final int REQUEST_CONNECT_DEVICE = 1;
	public static final int REQUEST_ENABLE_BT = 2;

	private AlertDialog alertSalir;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.inicio);
		btnSalir = (Button) findViewById(R.id.btnSalir);
		btnConectar = (Button) findViewById(R.id.btnConectar);
		btnSalir.setOnClickListener(this);
		btnConectar.setOnClickListener(this);
		// mControladorLogica=new ControladorLogica();
		AlertDialog.Builder builder = new AlertDialog.Builder(this);
		builder.setMessage("Are you sure that you want to leave the
application?");
		builder.setCancelable(false);
		builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {

			public void onClick(DialogInterface dialog, int id) {
				finish();
			}
		});
		builder.setNegativeButton("No", new DialogInterface.OnClickListener() {

			public void onClick(DialogInterface dialog, int id) {
				dialog.cancel();
			}
		});
		alertSalir = builder.create();
	}

	public void onClick(View target) {
		if (target == btnConectar) {
			if (encenderBluetooth()) {
				verDispositivos();
			}

		} else if (target == btnSalir) {
			alertSalir.show();
		}
	}

	private boolean encenderBluetooth() {
		boolean encendido = false;
```

```
		BluetoothAdapter mBluetoothAdapter = BluetoothAdapter
				.getDefaultAdapter();
		if (mBluetoothAdapter == null) {
			avisarErrorBluetooth();
		} else {
			if (!mBluetoothAdapter.isEnabled()) {
				// En caso de que no este encendido lo enciende
				Intent enableIntent = new Intent(
						BluetoothAdapter.ACTION_REQUEST_ENABLE);
				startActivityForResult(enableIntent,
						VistaInicial.REQUEST_ENABLE_BT);
			} else {
				encendido = true;
			}
		}
		return encendido;
	}

	private void verDispositivos() {
		/*
		 * Lanza el VistaListaDispositivos para ver los dispositivos y hacer la
		 * búsqueda de dispositivos
		 */
		Intent serverIntent = new Intent(this, VistaListaDispositivos.class);
		startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);

	}

	@Override
	public boolean onKeyDown(int keyCode, KeyEvent event) {
		if (keyCode == KeyEvent.KEYCODE_BACK && event.getRepeatCount() == 0) {
			event.startTracking();
			return true;
		} else if (keyCode == KeyEvent.KEYCODE_SEARCH
				&& event.getRepeatCount() == 0) {
			event.startTracking();

			return true;
		}
		return super.onKeyDown(keyCode, event);

	}

	@Override
	public boolean onKeyUp(int keyCode, KeyEvent event) {
		if (keyCode == KeyEvent.KEYCODE_BACK && event.isTracking()
				&& !event.isCanceled()) {
			alertSalir.show();
			return true;
		} else if (keyCode == KeyEvent.KEYCODE_SEARCH && event.isTracking()
				&& !event.isCanceled()) {

			return true;
		}

		return super.onKeyUp(keyCode, event);
	}

	@Override
	protected void onActivityResult(int requestCode, int resultCode, Intent data) {
		super.onActivityResult(requestCode, resultCode, data);
		try {
			switch (requestCode) {
			case REQUEST_ENABLE_BT:
				if (resultCode == Activity.RESULT_OK) {
					verDispositivos();
				} else {
					// User did not enable Bluetooth or an error
occured
```

```
                                        Toast.makeText(this,
R.string.bt_not_enabled_leaving,
                                                        Toast.LENGTH_SHORT).show();
                                        // finish();
                                }
                        case REQUEST_CONNECT_DEVICE:
                                // When DeviceListActivity returns with a device to connect
                                if (resultCode == Activity.RESULT_OK) {

                                        // Get the device MAC address
                                        String deviceAddress = data.getExtras().getString(

        VistaListaDispositivos.EXTRA_DEVICE_ADDRESS);
                                        // create an Intent to talk to ActivityBluetooth
                                        Intent miIntentMPLO = new Intent(VistaInicial.this,
                                                        VistaPrincipal.class);
                                        Bundle miBundle = new Bundle();
                                        miBundle.putString(ACTIVITY_SERVICE,
deviceAddress);
                                        miIntentMPLO.putExtras(miBundle);
                                        // Llamo a la nueva actividad y espero resultados
                                        startActivityForResult(miIntentMPLO, IPC_ID);
                                        // Get the BLuetoothDevice object
                                }
                                break;
                        case IPC_ID:
                                finish();
                        default:
                                break;
                        }
                } catch (Exception e) {
                        Toast.makeText(getBaseContext(), e.getMessage(),
Toast.LENGTH_LONG)
                                        .show();
                }// try
        }// onActivityResult

        @Override
        protected void onDestroy() {
                // Clean up any resources including ending threads,
                // closing database connections etc.
                super.onDestroy();

        }

        public void avisarErrorBluetooth() {
                Toast.makeText(this, R.string.no_bt_device, Toast.LENGTH_LONG).show();
                finish();

        }

}
```

Contenido del fichero VistaListaDispositivos.java

```
package vista;

import java.util.Set;

import upc.KS_HMI.R;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Window;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;

public class VistaListaDispositivos extends Activity implements OnClickListener {

	// Return Intent extra
	public static String EXTRA_DEVICE_ADDRESS = "device_address";

	// Member fields
	private Button scanButton;
	private BluetoothAdapter mBtAdapter;
	private ArrayAdapter<String> mPairedDevicesArrayAdapter;
	private ArrayAdapter<String> mNewDevicesArrayAdapter;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		// Setup the window
		requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
		setContentView(R.layout.device_list);

		// Set result CANCELED incase the user backs out
		setResult(Activity.RESULT_CANCELED);

		// Initialize the button to perform device discovery
		scanButton = (Button) findViewById(R.id.button_scan);
		scanButton.setOnClickListener(this);

		// Initialize array adapters. One for already paired devices and
		// one for newly discovered devices
		mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this,
				R.layout.device_name);
		mNewDevicesArrayAdapter = new ArrayAdapter<String>(this,
				R.layout.device_name);

		// Find and set up the ListView for paired devices
		ListView pairedListView = (ListView) findViewById(R.id.paired_devices);
		pairedListView.setAdapter(mPairedDevicesArrayAdapter);
		pairedListView.setOnItemClickListener(mDeviceClickListener);

		// Find and set up the ListView for newly discovered devices
		ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);
		newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
		newDevicesListView.setOnItemClickListener(mDeviceClickListener);
```

```
        // Register for broadcasts when a device is discovered
        IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
        this.registerReceiver(mReceiver, filter);

        // Register for broadcasts when discovery has finished
        filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        this.registerReceiver(mReceiver, filter);

        // Get the local Bluetooth adapter
        mBtAdapter = BluetoothAdapter.getDefaultAdapter();

        // Get a set of currently paired devices
        Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();

        // If there are paired devices, add each one to the ArrayAdapter
        if (pairedDevices.size() > 0) {

findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
                for (BluetoothDevice device : pairedDevices) {
                        mPairedDevicesArrayAdapter.add(device.getName() + "\n"
                                        + device.getAddress());
                }
        } else {
                String noDevices = getResources().getText(R.string.none_paired)
                                .toString();
                mPairedDevicesArrayAdapter.add(noDevices);
        }
}


public void onClick(View v) {
        if (v == scanButton) {
                doDiscovery();
                v.setVisibility(View.GONE);
        }
}

@Override
protected void onDestroy() {
        super.onDestroy();

        // Make sure we're not doing discovery anymore
        if (mBtAdapter != null) {
                mBtAdapter.cancelDiscovery();
        }

        // Unregister broadcast listeners
        this.unregisterReceiver(mReceiver);
}

/**
 * Start device discover with the BluetoothAdapter
 */
private void doDiscovery() {

        // Indicate scanning in the title
        setProgressBarIndeterminateVisibility(true);
        setTitle(R.string.scanning);

        // Turn on sub-title for new devices
        findViewById(R.id.title_new_devices).setVisibility(View.VISIBLE);

        // If we're already discovering, stop it
        if (mBtAdapter.isDiscovering()) {
                mBtAdapter.cancelDiscovery();
        }

        // Request discover from BluetoothAdapter
        mBtAdapter.startDiscovery();
}

// The on-click listener for all devices in the ListViews
private OnItemClickListener mDeviceClickListener = new OnItemClickListener() {
```

```
		public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {
			// Cancel discovery because it's costly and we're about to connect
			mBtAdapter.cancelDiscovery();

			// Get the device MAC address, which is the last 17 chars in the
			// View
			String info = ((TextView) v).getText().toString();
			String address = info.substring(info.length() - 17);

			// Create the result Intent and include the MAC address
			Intent intent = new Intent();
			intent.putExtra(EXTRA_DEVICE_ADDRESS, address);

			// Set result and finish this Activity
			setResult(Activity.RESULT_OK, intent);
			finish();
		}
	};

	// The BroadcastReceiver that listens for discovered devices and
	// changes the title when discovery is finished
	private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
		@Override
		public void onReceive(Context context, Intent intent) {
			String action = intent.getAction();

			// When discovery finds a device
			if (BluetoothDevice.ACTION_FOUND.equals(action)) {
				// Get the BluetoothDevice object from the Intent
				BluetoothDevice device = intent

	.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
				// If it's already paired, skip it, because it's been
listed
				// already
				if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
					mNewDevicesArrayAdapter.add(device.getName() + "\n"
							+ device.getAddress());
				}
				// When discovery is finished, change the Activity title
			} else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED
					.equals(action)) {
				setProgressBarIndeterminateVisibility(false);
				setTitle(R.string.select_device);
				if (mNewDevicesArrayAdapter.getCount() == 0) {
					String noDevices = getResources().getText(
							R.string.none_found).toString();
					mNewDevicesArrayAdapter.add(noDevices);
				}
			}
		}
	};

}
```

Contenido del fichero VistaPrincipal.java

```
package vista;
import logica.ControladorLogica;
import upc.KS_HMI.R;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;

import android.os.Bundle;
import android.os.Handler;
import android.os.Message;

import android.view.Gravity;
import android.view.KeyEvent;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

public class VistaPrincipal extends Activity {

	/** Called when the activity is first created. */
	private ControladorLogica intproc;
	private TextView tvNumeroMuestras;
	private TextView tvBarraEstado;
	private ImageView ivEstadoSomnolencia;
	private AlertDialog alertSalir;
	private AlertDialog.Builder adMensaje;

	// Message types sent from the VistaPrincipal Handler
	private static final int MESSAGE_MUESTRAS_PERDIDAS = 1;
	private static final int MESSAGE_ACTUALIZAR_SEMAFORO = 2;
	private static final int MESSAGE_DEVICE_NAME = 3;
	private static final int MESSAGE_TOAST = 4;
	private static final int MESSAGE_MUESTRAS_RECIBIDAS = 5;
	public static final String DEVICE_NAME = "device_name";
	public static final String TOAST = "toast";
	public static final String MUESTRAS_RECIBIDAS = "muestras";
	public static final String ESTADO_CALCULADO = "estado";
	protected static final String VALOR_MUESTRAS_RECIBIDAS = "valor";
	private int estadoPintado;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		Intent myIntentLocal = getIntent();
		String adress = myIntentLocal.getExtras().getString(ACTIVITY_SERVICE);
		setContentView(R.layout.vistaresultados);
		this.estadoPintado = -1;
		tvNumeroMuestras = (TextView) findViewById(R.id.txtNumeroMuestras);
		tvBarraEstado = (TextView) findViewById(R.id.txtEstado);
		ivEstadoSomnolencia = (ImageView) findViewById(R.id.imageViewIndicador);
		AlertDialog.Builder builder = new AlertDialog.Builder(this);
		builder.setMessage("Are you sure that you want to leave the
application?");
		builder.setCancelable(false);
		builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {

			public void onClick(DialogInterface dialog, int id) {
				intproc.cerrarInterfazProcesamiento();
			}
		});
		builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
```

```
                public void onClick(DialogInterface dialog, int id) {
                        dialog.cancel();
                }
        });
        alertSalir = builder.create();

        adMensaje = new AlertDialog.Builder(this);
        adMensaje.setCancelable(false);
        adMensaje.setNeutralButton("Ok", new DialogInterface.OnClickListener() {

                public void onClick(DialogInterface dialog, int id) {
                        intproc.salirSistema();
                }
        });

        if (intproc == null) {
                intproc = new ControladorLogica(this);
                intproc.iniciarConexionProcesado(adress);
        }

}

@Override
public void onStart() {
        super.onStart();
}

@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK && event.getRepeatCount() == 0) {
                event.startTracking();
                return true;
        } else if (keyCode == KeyEvent.KEYCODE_SEARCH
                        && event.getRepeatCount() == 0) {
                event.startTracking();

                return true;
        }
        return super.onKeyDown(keyCode, event);

}

@Override
public boolean onKeyUp(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK && event.isTracking()
                        && !event.isCanceled()) {
                alertSalir.show();
                return true;
        } else if (keyCode == KeyEvent.KEYCODE_SEARCH && event.isTracking()
                        && !event.isCanceled()) {

                return true;
        }

        return super.onKeyUp(keyCode, event);
}

/**
 * El handler consigue la información enviada desde el Controlador del
```

```
	 */
	private final Handler interfazHandler = new Handler() {

		@Override
		public void handleMessage(Message msg) {
			Integer numeroMuestras;
			switch (msg.what) {

			case MESSAGE_MUESTRAS_RECIBIDAS:
				numeroMuestras = msg.getData().getInt(MUESTRAS_RECIBIDAS);
				tvNumeroMuestras.setText(numeroMuestras.toString());
				ivEstadoSomnolencia.setImageResource(R.drawable.empty);
				break;

			case MESSAGE_MUESTRAS_PERDIDAS:
				numeroMuestras = msg.getData().getInt(MUESTRAS_RECIBIDAS);
				ivEstadoSomnolencia.setImageResource(R.drawable.n0);
				tvBarraEstado.setText(R.string.stbmuestrasPerdidas);
				break;

			case MESSAGE_ACTUALIZAR_SEMAFORO:
				numeroMuestras = msg.getData().getInt(MUESTRAS_RECIBIDAS);
				Integer estCalculado =
msg.getData().getInt(ESTADO_CALCULADO);
				tvNumeroMuestras.setText(numeroMuestras.toString());

				switch (estCalculado) {
				case -1:

	ivEstadoSomnolencia.setImageResource(R.drawable.n0);

	tvBarraEstado.setText(R.string.stbInicializandoSistema);
					break;
				case 0:

	ivEstadoSomnolencia.setImageResource(R.drawable.n1);
					tvBarraEstado.setText(R.string.stbMostrandoEstado);
					break;
				case 1:

	ivEstadoSomnolencia.setImageResource(R.drawable.n5);
					tvBarraEstado.setText(R.string.stbMostrandoEstado);
					break;
				default:
					break;
				}
				break;
			case MESSAGE_DEVICE_NAME:
				// save the connected device's name
				String mConnectedDeviceName = msg.getData().getString(
						DEVICE_NAME);
				String text = "Connected to " + mConnectedDeviceName;
				Toast toast2 = Toast.makeText(getApplicationContext(),
text,
						Toast.LENGTH_SHORT);
				toast2.setGravity(Gravity.CENTER, 0, 0);
				toast2.show();

				break;
			case MESSAGE_TOAST:
				mostrarMensajeConfirmacion(msg, TOAST);
				break;
			}
		}
```

```
};

public void mostrarMensajeConfirmacion(Message msg, String TOAST) {
        adMensaje.setMessage(msg.getData().getString(TOAST));
        AlertDialog dialog = adMensaje.create();
        dialog.show();

}

public void AvisarConexonPerdida() {
        Message msg = interfazHandler.obtainMessage(MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(TOAST, getString(R.string.conexion_perdida));
        msg.setData(bundle);
        interfazHandler.sendMessage(msg);
}

public void AvisarMuestrasPerdidas() {
        Message msg = interfazHandler.obtainMessage(MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(TOAST, getString(R.string.muestrasPerdidas));
        msg.setData(bundle);
        interfazHandler.sendMessage(msg);
}

public void avisarDispositivoIncompatible() {
        Message msg = interfazHandler.obtainMessage(MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(TOAST, getString(R.string.dispositivoincompatible));
        msg.setData(bundle);
        interfazHandler.sendMessage(msg);

}

public void AvisarConexionFallida() {
        Message msg = interfazHandler.obtainMessage(MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(TOAST, getString(R.string.unable_connect));
        msg.setData(bundle);
        interfazHandler.sendMessage(msg);
}

public void AvisarErrorFormato() {
        Message msg = interfazHandler.obtainMessage(MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(TOAST, getString(R.string.error_formato));
        msg.setData(bundle);
        interfazHandler.sendMessage(msg);
}

public void EnviarNombreDispositivoConectado(String deviceName) {
        Message msg = interfazHandler.obtainMessage(MESSAGE_DEVICE_NAME);
        Bundle bundle = new Bundle();
        bundle.putString(DEVICE_NAME, deviceName);
        msg.setData(bundle);
        interfazHandler.sendMessage(msg);

}
```

```
	private void ActualizarContadoresSemaforo(int dif, int estadoDif,
			int muestrasRecibidas) {
		Message msg = interfazHandler
				.obtainMessage(MESSAGE_ACTUALIZAR_SEMAFORO);
		Bundle bundle = new Bundle();
		bundle.putInt(MUESTRAS_RECIBIDAS, muestrasRecibidas);
		bundle.putInt(ESTADO_CALCULADO, estadoDif);
		msg.setData(bundle);
		interfazHandler.sendMessage(msg);
	}

	private void ActualizarContadoresFlash(int muestrasRecibidas) {
		Message msg = interfazHandler.obtainMessage(MESSAGE_MUESTRAS_RECIBIDAS);
		Bundle bundle = new Bundle();
		bundle.putInt(MUESTRAS_RECIBIDAS, muestrasRecibidas);
		msg.setData(bundle);
		interfazHandler.sendMessage(msg);
	}

	public void ActualizarVistaPinta(int dif, int estadoCalculado,
			int muestrasRecibidas) {

		if (estadoCalculado != estadoPintado || (muestrasRecibidas % 20 != 0)) {
			estadoPintado = estadoCalculado;
			ActualizarContadoresSemaforo(dif, estadoCalculado,
					muestrasRecibidas);
		} else {
			ActualizarContadoresFlash(muestrasRecibidas);
		}

	}

	public void SalirAplicacion() {
		finish();
	}

	@Override
	protected void onDestroy() {
		super.onDestroy();
	}

}
```

Contenido del fichero ControladorComunicacion.java

```
package comunicacion;

import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

import logica.ControladorLogica;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;

/**
 * Se encarga de monitorizar la conexión y pasar los datos a la interfaz según
 * el tipo de dispositivo
 **/
public class ControladorComunicacion {

	// Member fields
	private BluetoothAdapter mAdapter;
	private BluetoothDevice mDevice;
	private ControladorLogica intProc;
	private ConnectThread mConnectThread;
	private ConectadoBanda mConnectedBanda;
	private ConectadoOrdenador mConnectedOrdenador;
	private boolean estaEsperando;
	private boolean estaConectado;

	public ControladorComunicacion(ControladorLogica intProc) {
		mAdapter = BluetoothAdapter.getDefaultAdapter();
		this.intProc = intProc;
		estaConectado = false;
	}

	/**
	 * Start the ConnectThread to initiate a connection to a remote device. -
	 * device - The BluetoothDevice to connect
	 */
	public synchronized void iniciarLectura(String deviceAddress) {

		BluetoothDevice device = mAdapter.getRemoteDevice(deviceAddress);
		// Cancelar cualquier thread que intente realizar la conexion
		if (mConnectThread != null) {
			mConnectThread.cancel();
			mConnectThread = null;
		}

		// Cancelar cualquier thread que este conectado
		if (mConnectedBanda != null) {
			mConnectedBanda.cancel();
			mConnectedBanda = null;
		}
		if (mConnectedOrdenador != null) {
			mConnectedOrdenador.cancel();
			mConnectedOrdenador = null;
		}
		if (!device.getName().contentEquals("bioplux")
				&& !device.getName().contentEquals("BP2mini")
				&& !device.getName().contentEquals("TOSH-ESEM")
				&& !device.getName().contentEquals("samantha")
				&& !device.getName().contentEquals("BIOZ1")) {

			dispositivoincompatible();
		} else {
			// Iniciar el thread para conectar con el dispositivo dado
			mConnectThread = new ConnectThread(device);
			mConnectThread.start();
```

```
        }
    }

    /**
     * Start the ConnectedThread to begin managing a Bluetooth connection -
     * socket - The BluetoothSocket on which the connection was made - device -
     * The BluetoothDevice that has been connected
     */
    private synchronized void connected(BluetoothSocket socket,
                    BluetoothDevice device) {
            mDevice = device;
            estaConectado = true;
            intProc.avisarDispositivoConectado(device.getName());
            // Start the thread to manage the connection and perform transmissions
            if (mDevice.getName().contentEquals("bioplux")) {
                    // Banda blanca
                    mConnectedBanda = new ConectadoBanda(socket, this);
                    mConnectedOrdenador = null;
                    mConnectedBanda.leerDatos();

            } else if (mDevice.getName().contentEquals("BP2mini")) {
                    // Banda blanca
                    mConnectedBanda = new ConectadoBanda(socket, this);
                    mConnectedOrdenador = null;
                    mConnectedBanda.leerDatos();
            }

            else if (mDevice.getName().contentEquals("TOSH-ESEM")) {
                    mConnectedOrdenador = new ConectadoOrdenador(socket, this);
                    mConnectedBanda = null;
                    mConnectedOrdenador.leerDatos();
            } else if (mDevice.getName().contentEquals("samantha")) {
                    mConnectedOrdenador = new ConectadoOrdenador(socket, this);
                    mConnectedBanda = null;
                    mConnectedOrdenador.leerDatos();
            } else if (mDevice.getName().contentEquals("BIOZ1")) {
                    mConnectedOrdenador = new ConectadoOrdenador(socket, this);
                    mConnectedBanda = null;
                    mConnectedOrdenador.leerDatos();
            }

    }

    /**
     * Stop all threads
     */
    public synchronized void stop() {
            if (mConnectThread != null) {
                    mConnectThread.cancel();
                    mConnectThread = null;
            }
            if (mConnectedBanda != null) {
                    mConnectedBanda.cancel();
                    mConnectedBanda = null;
            }
            if (mConnectedOrdenador != null) {
                    mConnectedOrdenador.cancel();
                    mConnectedOrdenador = null;
            }
    }

    /**
     * Indicate that the connection attempt failed and notify the UI Activity.
     */
    private void connectionFailed() {
            intProc.enviarMensajeConnectionFailed();
    }

    /**
     * Indica que el dispositivo seleccionado es inompatible
     */
    private void dispositivoincompatible() {
            intProc.enviarMensajeDispositivoIncompatible();
```

```
	}

	/**
	 * Indicate that the connection was lost and notify the UI Activity.
	 */
	void connectionLost() {
		intProc.enviarMensajeConexionPerdida();
	}

	/**
	 * This thread runs while attempting to make an outgoing connection with a
	 * device. It runs straight through; the connection either succeeds or
	 * fails.
	 */
	private class ConnectThread extends Thread {
		private final BluetoothSocket mmSocket;
		private final BluetoothDevice mmDevice;

		public ConnectThread(BluetoothDevice device) {
			mmDevice = device;
			BluetoothSocket tmp = null;
			Method m;
			// Get a BluetoothSocket for a connection with the given
			// BluetoothDevice
			try {
				m = mmDevice.getClass().getMethod("createRfcommSocket",
						new Class[] { int.class });
				tmp = (BluetoothSocket) m.invoke(mmDevice, 1);
				// tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
			} catch (SecurityException e) {
				connectionFailed();
			} catch (NoSuchMethodException e) {
				connectionFailed();
			} catch (IllegalArgumentException e) {

				connectionFailed();
			} catch (IllegalAccessException e) {
				connectionFailed();
			} catch (InvocationTargetException e) {

				connectionFailed();
			}
			mmSocket = tmp;

		}

		@Override
		public void run() {
			setName("ConnectThread");
			// Always cancel discovery because it will slow down a connection
			mAdapter.cancelDiscovery();
			// Make a connection to the BluetoothSocket
			try {
				// This is a blocking call and will only return on a
successful
				// connection or an exception
				mmSocket.connect();

			} catch (IOException e) {
				connectionFailed();
				// Close the socket
				try {
					mmSocket.close();
				} catch (IOException e2) {

				}

				return;
			}
```

```
                        // Empieza el Thread conectado
                        connected(mmSocket, mmDevice);
                }

                public void cancel() {
                        try {
                                mmSocket.close();
                        } catch (IOException e) {
                        }
                }
        }

        public void FinalizarLectura() {
                estaConectado = false;
                if (estaEsperando)
                        this.stop();
        }

        public boolean EstaConectado() {

                return estaConectado;
        }

        public void salirSistema() {
                intProc.salirSistema();

        }

        public void EnviarMensajeErrorFormato() {
                intProc.enviarMensajeErrorFormato();

        }

        public void tratarMuestraObtenida(int muestra) {
                intProc.tratarMuestraObtenida(muestra);

        }

        public void MostrarResultado(int nMuestrasPerdidas) {
                intProc.mostrarResultado(nMuestrasPerdidas);
        }

}
```

Contenido del fichero ConectadoBanda.java

```
package comunicacion;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import android.bluetooth.BluetoothSocket;

public class ConectadoBanda{
	private final BluetoothSocket mmSocket;
	private final InputStream mmInStream;
	private final OutputStream mmOutStream;
	private ControladorComunicacion mLsp;
	private int bufferSize = 4096;
	private int nTramaAnterior;
	byte mensaje[] = new byte[4];

	public ConectadoBanda(BluetoothSocket socket, ControladorComunicacion lsp) {
		mmSocket = socket;
		mLsp = lsp;
		InputStream tmpIn = null;
		OutputStream tmpOut = null;
		nTramaAnterior = -1;
		// Get the BluetoothSocket input and output streams
		try {
			tmpIn = mmSocket.getInputStream();
			tmpOut = mmSocket.getOutputStream();

		} catch (IOException e) {
		}
		mmInStream = tmpIn;
		mmOutStream = tmpOut;

	}

	private void EnviarSecuenciaIncializacion() {
		int bytesLeidos;
		int bytesResiduo;
		byte[] bufferResiduo = new byte[4096];
		byte[] bcomandos = new byte[20];
		String comandos = "V";
		try {
			mmOutStream.write(comandos.getBytes());
			bytesLeidos = mmInStream.read(bcomandos, 0, bcomandos.length);
			if (bytesLeidos > 0) {
				//String readMessage = new String(bcomandos, 0,
bytesLeidos);
				bytesResiduo = mmInStream.available();
				while (bytesResiduo > 0) {
					if (bytesResiduo <= 4096)
						mmInStream.read(bufferResiduo, 0,
bytesResiduo);
					else
						mmInStream.read(bufferResiduo, 0, 4096);
					bytesResiduo = mmInStream.available();
				}
				comandos = "@START,0040,01,12;";
				mmOutStream.write(comandos.getBytes());
```

```
                }
            } catch (IOException e) {
                mLsp.connectionLost();
            }

        }


        public void leerDatos() {
            int bytesLeidos = -1;
            byte[] buffer;
            int bytesNoLeidos, bytesPorLeer;
            int resto = 0;
            int nMuestrasPerdidas = 0;
            boolean error = false;
            // Keep listening to the InputStream while connected
            buffer = new byte[bufferSize];
            EnviarSecuenciaIncializacion();
            try {
                while (mLsp.EstaConectado() && !error) {
                    bytesNoLeidos = mmInStream.available();
                    if (bytesNoLeidos < 4) {

                        bytesLeidos = mmInStream.read(buffer, 0, 4);
                        while (bytesLeidos < 4) {
                            bytesPorLeer = 4 - bytesLeidos;
                            bytesLeidos += mmInStream.read(buffer,
bytesLeidos,
                                    bytesPorLeer);
                        }
                    } else if (bytesNoLeidos <= bufferSize) {

                        resto = bytesNoLeidos % 4;
                        bytesLeidos = mmInStream.read(buffer, 0,
bytesNoLeidos
                                - resto);
                    } else {

                        bytesLeidos = mmInStream.read(buffer, 0,
bufferSize);
                    }
                    if ((bytesLeidos % 4) == 0) {
                        nMuestrasPerdidas =
ParsearDatosFormatoBanda(buffer, 0,
                                bytesLeidos);
                        mLsp.MostrarResultado(nMuestrasPerdidas);
                    } else {
                        error = true;
                        mLsp.EnviarMensajeErrorFormato();
                    }
                }
                mLsp.salirSistema();
            } catch (IOException e) {
                mLsp.connectionLost();
            }

        }

        public void cancel() {
            int bytesResiduo;
            byte[] bufferResiduo = new byte[4096];
            try {
                String comando = "R";
                mmOutStream.write(comando.getBytes(), 0, 1);
```

```
                bytesResiduo = mmInStream.available();
                while (bytesResiduo > 0) {
                        if (bytesResiduo <= 4096)
                                mmInStream.read(bufferResiduo, 0, bytesResiduo);
                        else
                                mmInStream.read(bufferResiduo, 0, 4096);
                        bytesResiduo = mmInStream.available();
                }
                mmSocket.close();
        } catch (IOException e) {

        }

    }

    public int getValorSize() {
        return bufferSize;
    }

    private int ParsearDatosFormatoBanda(byte[] buffer, int offset, int count) {
        int x = 0, y = 0, bytesParseados = 0;
        int valorMuestra;
        int nTrama;
        int muestrasPerdidas = 0;
        while (bytesParseados < count) {
                nTrama = (int) (buffer[bytesParseados] & 0x7F);
                x = (int) (buffer[bytesParseados + 1] & 0xFF);
                y = (int) (buffer[bytesParseados + 2] & 0x0F);
                valorMuestra = x + y * 256;
                muestrasPerdidas += comprobarNumeroTrama(nTrama);
                mLsp.tratarMuestraObtenida(valorMuestra);
                nTramaAnterior = nTrama;
                bytesParseados += 4;
        }
        return muestrasPerdidas;
    }



    private int comprobarNumeroTrama(int nTramaActual) {
        int difTrama = nTramaActual - nTramaAnterior;
        int muestrasPerdidas = 0;
        if (nTramaAnterior != -1
                        && !(nTramaActual == 0 && nTramaAnterior == 127)) {
                if (difTrama > 1) {
                        muestrasPerdidas = difTrama - 1;
                        // se han perdido difTrama-1
                } else if (difTrama < 0) {
                        // se han perdido (127-nTramaAnterior) + nTramaActual
                        muestrasPerdidas = nTramaActual + 127 - nTramaAnterior;
                }
        }
        return muestrasPerdidas;
    }
}
```

Contenido del fichero ConectadoOrdenador.java

```
package comunicacion;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import android.bluetooth.BluetoothSocket;

public class ConectadoOrdenador{
	private final BluetoothSocket mmSocket;
	private final InputStream mmInStream;
	private final OutputStream mmOutStream;
	private ControladorComunicacion mLsp;
	private int bufferSize = 4096;
	private Integer[] datosParseados;

	String[] datos;
	int muestrasLeidas;
	String readMessage;

	public ConectadoOrdenador(BluetoothSocket socket,
			ControladorComunicacion lsp) {
		mmSocket = socket;
		mLsp = lsp;
		InputStream tmpIn = null;
		OutputStream tmpOut = null;
		if (datosParseados == null) {
			datosParseados = new Integer[4096];
		}
		// Get the BluetoothSocket input and output streams
		try {
			tmpIn = mmSocket.getInputStream();
			tmpOut = mmSocket.getOutputStream();

		} catch (IOException e) {
		}
		mmInStream = tmpIn;
		mmOutStream = tmpOut;

	}


	public void leerDatos() {
		int bytesLeidos = -1;
		int bufferSize = 4096;

		byte[] buffer;
		int bytesNoLeidos;
		// Keep listening to the InputStream while connected
		buffer = new byte[bufferSize];
		try {
			while (mLsp.EstaConectado()) {
				bytesNoLeidos = mmInStream.available();
				if (bytesNoLeidos <= 0) {
					// buffer = new byte[bufferSize];
					bytesLeidos = mmInStream.read(buffer, 0,
bufferSize);
				} else if (bytesNoLeidos <= bufferSize) {
					bytesLeidos = mmInStream.read(buffer, 0,
bytesNoLeidos);
				} else {
					bufferSize = bytesNoLeidos;
					buffer = null;
					buffer = new byte[bufferSize];
					bytesLeidos = mmInStream.read(buffer, 0,
bytesNoLeidos);
				}
				ParsearDatosFormatoOrdenador(buffer,0,bytesLeidos);
				mLsp.MostrarResultado(0);
			}
			mLsp.salirSistema();
```

```
                } catch (NumberFormatException e) {
                        mLsp.EnviarMensajeErrorFormato();
                } catch (IOException e) {
                        mLsp.connectionLost();
                        // intProc.ContinuarFinalizacion();
                }

        }

        public void cancel() {
                try {
                        String comando = "R";
                        mmOutStream.write(comando.getBytes(), 0, 1);
                        mmSocket.close();
                } catch (IOException e) {

                }

        }

        public int getValorSize() {
                return bufferSize;
        }

        private int ParsearDatosFormatoOrdenador(byte[] buffer, int offset,
                        int bytesLeidos) {
                readMessage = new String(buffer, 0, bytesLeidos);
                datos = readMessage.split("\n");
                muestrasLeidas = datos.length;
                int muestraObtenida;
                for (int i = 0; i < muestrasLeidas; i++) {

                        if (datos[i].length() > 0) {
                                muestraObtenida = Integer.parseInt(datos[i]);
                                mLsp.tratarMuestraObtenida(muestraObtenida);
                        } else {
                                mLsp.EnviarMensajeErrorFormato();
                                break;
                        }

                }
                return muestrasLeidas;
        }
}
```

Contenido del fichero ControladorLogica.java

```
package logica;

import java.util.ArrayList;
import comunicacion.ControladorComunicacion;
import vista.VistaPrincipal;
import android.content.Context;
import android.telephony.TelephonyManager;

/**
 * @author Federico Guede
 * @category Esta clase es la que contiene la lógica de la aplicación
 * */
public class ControladorLogica {

	private ControladorComunicacion lsp;
	private ProcesadorRespiracion myResp;
	private VistaPrincipal vst;
	private AvisadorSomnolencia avisadorSomnolencia;
	private ArchivadorResultados ArchivoResultado;
	private int[] lNumeroMuestra;
	private int[] lValoresMuestra;
	private float[] lValoresFiltrados;
	private float[] lValoresIndice;
	private final int maxValoresGuardados = 100;
	private int nValoresGuardados;
	private int[] lNumeroCiclos;
	private float[] lValorThreshold;
	private float[] lFactorNormalizacion;
	private int[] lTiempoReferenciaInicial;
	private int[] lTiempoReferenciaFinal;
	private int[] lPasosPorUmbral;
	private float[] lHtr;
	private boolean bAvisoMuestrasPerdidas;

	public ControladorLogica(VistaPrincipal vst) {
		this.vst = vst;
		String srvcName = Context.TELEPHONY_SERVICE;
		TelephonyManager telephonyManager = (TelephonyManager) vst
				.getSystemService(srvcName);
		avisadorSomnolencia = new AvisadorSomnolencia("9320147777",
				vst.getApplicationContext(), telephonyManager);
		ArchivoResultado = new ArchivadorResultados();
		if (lsp == null)
			lsp = new ControladorComunicacion(this);
		if (myResp == null)
			myResp = new ProcesadorRespiracion(this);

		lNumeroMuestra = new int[maxValoresGuardados];
		lValoresMuestra = new int[maxValoresGuardados];
		lValoresFiltrados = new float[maxValoresGuardados];
		lValoresIndice = new float[maxValoresGuardados];
		lNumeroCiclos = new int[maxValoresGuardados];
		lValorThreshold = new float[maxValoresGuardados];
		lFactorNormalizacion = new float[maxValoresGuardados];
		lTiempoReferenciaInicial = new int[maxValoresGuardados];
		lTiempoReferenciaFinal = new int[maxValoresGuardados];
		lPasosPorUmbral = new int[maxValoresGuardados];
		lHtr = new float[maxValoresGuardados];
		nValoresGuardados = 0;
		bAvisoMuestrasPerdidas = false;
	}

	public void iniciarConexionProcesado(String deviceAdress) {
		// Intenta conectar al dispositivo
		lsp.iniciarLectura(deviceAdress);
	}

	/**
	 * Envia mensaje de conexión fallida a la GUI
	 */
	public void enviarMensajeConnectionFailed() {
```

```
		vst.AvisarConexionFallida();
	}

	/**
	 * Envia mensaje dispositivo incompatible
	 */
	public void enviarMensajeDispositivoIncompatible() {
		vst.avisarDispositivoIncompatible();
	}

	/**
	 * Envia mensaje de conexión perdida a la GUI
	 */
	public void enviarMensajeConexionPerdida() {
		if (!bAvisoMuestrasPerdidas)
			vst.AvisarConexonPerdida();
	}

	/**
	 * Envia mensaje de error en el formato de los datos a la GUI
	 */
	public void enviarMensajeErrorFormato() {
		vst.AvisarErrorFormato();
	}

	public void avisarDispositivoConectado(String deviceName) {
		vst.EnviarNombreDispositivoConectado(deviceName);

	}

	/**
	 * Este metodo hace de pasarela a la parte GUI y wrapper del Avisador
	 *
	 * @param nMuestrasPerdidas
	 *            Numero de muestras que la comunicación detecta que se han
	 *            perdido
	 */
	public void mostrarResultado(int nMuestrasPerdidas) {

		if (nMuestrasPerdidas == 0) {
			vst.ActualizarVistaPinta((int) myResp.getDif(),
					myResp.consularEstadoAlarma(),
myResp.GetNumeroMuestras());

			if (myResp.consularEstadoAlarma() == 1) {
				avisadorSomnolencia.Avisar();
			} else
				avisadorSomnolencia.PararAviso();
		} else if (!bAvisoMuestrasPerdidas && nMuestrasPerdidas > 0) {
			bAvisoMuestrasPerdidas = true;
			lsp.stop();
			vst.AvisarMuestrasPerdidas();

			avisadorSomnolencia.PararAviso();
		}
	}

	public void cerrarInterfazProcesamiento() {
		lsp.FinalizarLectura();
	}

	public void salirSistema() {

		lsp.stop();
		myResp = null;
		avisadorSomnolencia.PararAviso();
		vst.SalirAplicacion();
	}
```

```
	/**
	 * Es el wrapper de la parte de procesado de respiración y del registro de
	 * los datos
	 *
	 * @param muestra
	 *            Valor de la última muestra leida
	 */

	public void tratarMuestraObtenida(int muestra) {
		float muestraFiltrada = myResp.ProcesarMuestra(muestra);
		lNumeroMuestra[nValoresGuardados] = myResp.GetNumeroMuestras();
		lValoresIndice[nValoresGuardados] = myResp.getDif();
		lValoresMuestra[nValoresGuardados] = muestra;
		lValoresFiltrados[nValoresGuardados] = muestraFiltrada;
		lNumeroCiclos[nValoresGuardados] = myResp.getNumeroCiclos();
		if (myResp.estadoIncializacion()) {
			lValorThreshold[nValoresGuardados] = myResp.getThreshold();
			lFactorNormalizacion[nValoresGuardados] = myResp
					.getFactorNormalizacion();
			lTiempoReferenciaInicial[nValoresGuardados] = myResp
					.getTiempoReferenciaInicial();
			lTiempoReferenciaFinal[nValoresGuardados] = myResp
					.getTiempoReferenciaFinal();
			lPasosPorUmbral[nValoresGuardados] = myResp.getPasosPorUmbral();
			lHtr[nValoresGuardados] = myResp.getHtrIndex();
			nValoresGuardados++;
			if (nValoresGuardados >= maxValoresGuardados
					|| myResp.minimoAlcanzado()) {

	ArchivoResultado.EscribirArchivoIncializacion(lNumeroMuestra,
						nValoresGuardados,
						lValorThreshold, lFactorNormalizacion,
						lTiempoReferenciaInicial,
lTiempoReferenciaFinal,lPasosPorUmbral, lHtr);

				ArchivoResultado.EscribirArchivoVariables(lNumeroMuestra,
						lValoresMuestra, lValoresFiltrados,
lValoresIndice,
						nValoresGuardados, lNumeroCiclos);
				nValoresGuardados = 0;
			}
		} else {
			nValoresGuardados++;
			if (nValoresGuardados >= maxValoresGuardados) {
				ArchivoResultado.EscribirArchivoVariables(lNumeroMuestra,
						lValoresMuestra, lValoresFiltrados,
lValoresIndice,
						nValoresGuardados, lNumeroCiclos);
				nValoresGuardados = 0;
			}

		}
	}

	public void EscribirArchivoPasosUmbral(float[] vTPeriodo,
			int nPasosUmbralNormalizacion) {
		ArchivoResultado.EscribirArchivoPasosUmbral(vTPeriodo,
			nPasosUmbralNormalizacion);

	}

	public void EscribirArchivoDecisionAlarma(int sumaTiempo,
			boolean bEstadoAlarma) {
		ArchivoResultado.EscribirArchivoDecisionAlarma(sumaTiempo,
bEstadoAlarma);

	}

}
```

Contenido del fichero ProcesadorRespiracion.java

```
package logica;

import java.util.ArrayList;

/**
 * @author Federico Guede
 *
 */
public class ProcesadorRespiracion {
	private int FS = 40;
	private int RETARDO;
	private int REF_WIN_LENGTH; // Longitud de la ventana de referencia en
	                                                    // muestras 40s *fs
	private int PCTIL = 60; // Percentil valor para determinar el threshold
	private int N_SAMPLES_INIT_WINDOW;
	private int tnew_index;
	private int told_index;
	private int M_av4_index;
	private int nciclosRespiracion;
	private int umbralFase1;
	private float threshold;
	private float OldSample;
	private float difCalculado;
	private float htr;
	private int index_buff;
	private int N_samples;
	private float htr_old;
	private float[] Resp_buffer;
	private float[] min_Resp_buffer;
	private float[] aux_Resp_buffer;
	private int[] Indt;
	private int[] Indcont;
	private boolean bMinimoAlcanzado;
	// Coeficientes filtro paso bajo
	private float[] bCoeficientePasoBajo;
	private float[] aCoeficientePasoBajo;
	private float xn1Bajo = 0, xn2Bajo = 0, yn1Bajo = 0, yn2Bajo = 0;
	// Coeficientes filtro paso alto
	private float[] bCoeficientePasoAlto;
	private float[] aCoeficientePasoAlto;
	private float xn1Alto = 0, xn2Alto = 0, yn1Alto = 0, yn2Alto = 0;
	private int signo;
	private float factorNormalizacion;
	private int nMuestraReferenciaFinal;
	private int nMuestraReferenciaInicio;
	private int nPasosUmbralNormalizacion;
	private ArrayList<TiempoKS> mListaTiempoKS;
	//private ArchivadorResultados mArchivadorResultados;
	private ControladorLogica mControladorLogica;
	private boolean bEstadoAlarma;

	public ProcesadorRespiracion(ControladorLogica mControladorLogica) {
		tnew_index = 0;
		told_index = 0;
		M_av4_index = 0;
		difCalculado = 0;
		threshold = 0;
		umbralFase1 = 6;
		nciclosRespiracion = 0;
		RETARDO = 20 * FS;
		REF_WIN_LENGTH = 40 * FS;
		N_SAMPLES_INIT_WINDOW = 900 * FS;
		index_buff = 0;
		N_samples = 0;
		htr_old = Float.MAX_VALUE;
		Resp_buffer = new float[REF_WIN_LENGTH];
		min_Resp_buffer = new float[REF_WIN_LENGTH];
```

```
        aux_Resp_buffer = new float[REF_WIN_LENGTH];
        Indt = new int[11];
        Indcont = new int[4];
        mListaTiempoKS = new ArrayList<TiempoKS>();
        this.mControladorLogica=mControladorLogica;
        bMinimoAlcanzado = false;
        bEstadoAlarma = false;
        // [b,a]=butter(2,1/(40/2),'low')
        bCoeficientePasoBajo = new float[3];
        // bCoeficientePasoBajo[0] = 0.0055427172102806566;
        bCoeficientePasoBajo[0] = 0.0055427172F;

        // bCoeficientePasoBajo[1] = 0.011085434420561313;
        bCoeficientePasoBajo[1] = 0.0110854344F;

        // bCoeficientePasoBajo[2] = 0.0055427172102806566;
        bCoeficientePasoBajo[2] = 0.0055427172F;

        aCoeficientePasoBajo = new float[3];
        aCoeficientePasoBajo[0] = 1;
        // aCoeficientePasoBajo[1] = -1.778631777824585;
        aCoeficientePasoBajo[1] = -1.7786317778F;


        // aCoeficientePasoBajo[2] = 0.80080264666570777;
        aCoeficientePasoBajo[2] = 0.8008026467F;

        // > [b,a]=butter(2,0.1/(40/2),'high')
        bCoeficientePasoAlto = new float[3];
        // bCoeficientePasoAlto[0] = 0.98895424806714005;
        bCoeficientePasoAlto[0] = 0.9889542481F;

        // bCoeficientePasoAlto[1] = -1.9779084961342801;
        bCoeficientePasoAlto[1] = -1.9779084961F;

        // bCoeficientePasoAlto[2] = 0.98895424806714005;
        bCoeficientePasoAlto[2] = 0.9889542481F;

        aCoeficientePasoAlto = new float[3];
        aCoeficientePasoAlto[0] = 1;
        // aCoeficientePasoAlto[1] = -1.9777864837767638;
        aCoeficientePasoAlto[1] = -1.9777864838F;


        // aCoeficientePasoAlto[2] = 0.97803050849179607;
        aCoeficientePasoAlto[2] = 0.9780305085F;

        signo = 1;

}

/**
 * Elimina la media de los valores del array de entrada
 *
 * @param aux_Resp_buffer
 *            array de muestras de respiracion
 */
private void remove_mean(float[] aux_Resp_buffer) {
        int k;
        float media = 0.0F;
        for (k = 0; k < REF_WIN_LENGTH; k++)
                media = media + aux_Resp_buffer[k];
        media = media / REF_WIN_LENGTH;

        for (k = 0; k < REF_WIN_LENGTH; k++)
                aux_Resp_buffer[k] = aux_Resp_buffer[k] - media;
}

private void copy_buffer(float[] resp_buffer, float[] aux_Resp_buffer) {
        int k;
        for (k = 0; k < REF_WIN_LENGTH; k++)
                aux_Resp_buffer[k] = resp_buffer[k];
}
```

```
	/**
	 * Calcula el indice de heteroestacidad
	 *
	 * @param resp_buffer
	 *            array de muestras respiratorias
	 * @return indice de heteroestacidad
	 */
	private float htrstest(float[] resp_buffer) {
		int j, k;
		float square_sum = 0, cumsum = 0;
		float htrs_index = 0.0F, d_index = 0.0F;
		this.copy_buffer(resp_buffer, aux_Resp_buffer);
		this.remove_mean(aux_Resp_buffer); // Elimina la media en los datos
		for (k = 0; k < REF_WIN_LENGTH; k++) // Calcula la suma de los cuadrados

	// del array de entrada
		{
			square_sum = square_sum + aux_Resp_buffer[k] * aux_Resp_buffer[k];
		}

		for (j = 0; j < REF_WIN_LENGTH; j++) // Substract mean value and

	// calculate square sum
		{
			// Calculamos la suma acumulada de los cuadrados para cada indice
			// del array
			cumsum = cumsum + (aux_Resp_buffer[j] * aux_Resp_buffer[j]);
			d_index = (cumsum * REF_WIN_LENGTH) / square_sum - j;
			if (Math.abs(d_index) > htrs_index) {
				htrs_index = Math.abs(d_index);
			}
		}

		htrs_index = htrs_index / (REF_WIN_LENGTH);
		return htrs_index;
	}

	void insertionSort(float[] aux_Resp_buffer, int array_size) {
		int i, j;
		float index;
		for (i = 1; i < array_size; ++i) {
			index = aux_Resp_buffer[i];
			for (j = i; j > 0 && aux_Resp_buffer[j - 1] > index; j--)
				aux_Resp_buffer[j] = aux_Resp_buffer[j - 1];

			aux_Resp_buffer[j] = index;
		}
	}

	private float percentil(float[] R_buff, int pctl) {
		int k;
		float resul;
		for (k = 0; k < REF_WIN_LENGTH; k++)
			aux_Resp_buffer[k] = R_buff[k]; // Make a temporal copy of the
array
		// to sort
		insertionSort(aux_Resp_buffer, REF_WIN_LENGTH); // Sort array
		k = (pctl * REF_WIN_LENGTH / 100);
		resul = aux_Resp_buffer[k];

		return resul; // Return amplitude value that has %pctl of array samples
					// below
	}

	void shift_buffer(float[] resp_buffer, float muestra) {
		int k;
		for (k = 0; k < (REF_WIN_LENGTH - 1); k++)
			resp_buffer[k] = resp_buffer[k + 1];
```

```
                resp_buffer[k] = muestra;
        }

        /**
         * Devuelve el estado de la alarma
         *
         * @return -1 inicialización, 0 estado despierto, 1 estado dormido
         */
        public int consularEstadoAlarma() {

                if (N_samples < (N_SAMPLES_INIT_WINDOW + RETARDO) && !bMinimoAlcanzado)
                        return -1;
                else if (!bEstadoAlarma)
                        return 0;
                else
                        return 1;
        }

        public void setFrecuencia(int p) {
                FS = p;
        }

        public void setVentana(int p) {
                REF_WIN_LENGTH = p;
        }

        public void setPercentil(int p) {
                PCTIL = p;
        }

        public void setUmbralFase1(int p) {
                umbralFase1 = p;
        }

        public int GetNumeroMuestras() {

                return tnew_index + N_samples;
        }

        public float getDif() {
                return difCalculado;
        }

        private float filtrarPasoBajoMuestra(int nuevaMuestra) {
                float muestraFiltrada = nuevaMuestra * bCoeficientePasoBajo[0]
                                + xn1Bajo * bCoeficientePasoBajo[1] + xn2Bajo
                                * bCoeficientePasoBajo[2] - aCoeficientePasoBajo[1] *
yn1Bajo
                                - aCoeficientePasoBajo[2] * yn2Bajo;
                xn2Bajo = xn1Bajo;
                xn1Bajo = nuevaMuestra;
                yn2Bajo = yn1Bajo;
                yn1Bajo = muestraFiltrada;

                return muestraFiltrada;
        }

        private float filtrarPasoAltoMuestra(float nuevaMuestra) {
                float muestraFiltrada = nuevaMuestra * bCoeficientePasoAlto[0]
                                + xn1Alto * bCoeficientePasoAlto[1] + xn2Alto
                                * bCoeficientePasoAlto[2] - aCoeficientePasoAlto[1] *
yn1Alto
                                - aCoeficientePasoAlto[2] * yn2Alto;
                xn2Alto = xn1Alto;
                xn1Alto = nuevaMuestra;
                yn2Alto = yn1Alto;
                yn1Alto = muestraFiltrada;

                return muestraFiltrada;
        }

        private void GirarSeñal(float[] resp_buffer) {
                int i;
```

```
		float minimaMuestra = resp_buffer[0];
		float maximaMuestra = resp_buffer[0];
		// Buscamos el valor maximo y minimo del array de entrada
		for (i = 1; i < REF_WIN_LENGTH; i++) {
			if (resp_buffer[i] > maximaMuestra)
				maximaMuestra = resp_buffer[i];
			else if (resp_buffer[i] < minimaMuestra)
				minimaMuestra = resp_buffer[i];
		}
		if (maximaMuestra < Math.abs(minimaMuestra)) {
			signo = -1;
			// Giramos el array de entrada
			for (i = 0; i < REF_WIN_LENGTH; i++) {
				resp_buffer[i] = resp_buffer[i] * signo;
			}
		}

	}

	private float CalcularThreshold(int NuevaMuestra) {

		float muestraFiltrada = filtrarPasoBajoMuestra(NuevaMuestra);
		muestraFiltrada = filtrarPasoAltoMuestra(muestraFiltrada);
		N_samples++;
		if (index_buff < REF_WIN_LENGTH) // Si el buffer no esta lleno
                                                                        //
insertamos una muestra nueva

		{
			Resp_buffer[index_buff] = muestraFiltrada;
			index_buff++;
		} else // Si el buffer esta lleno añadimos la muestra
		{
			shift_buffer(Resp_buffer, muestraFiltrada);
			htr = htrstest(Resp_buffer); // Calculamos indice de
heteroestacidad

			if (htr < 0.03)
				bMinimoAlcanzado = true;
			if (htr < htr_old) {
				htr_old = htr;
				copy_buffer(Resp_buffer, min_Resp_buffer);
				nMuestraReferenciaFinal = N_samples;
				nMuestraReferenciaInicio = nMuestraReferenciaFinal
						- REF_WIN_LENGTH;
			}
		}

		if (N_samples == N_SAMPLES_INIT_WINDOW || bMinimoAlcanzado) {
			GirarSeñal(min_Resp_buffer);
			threshold = percentil(min_Resp_buffer, PCTIL);
			factorNormalizacion =
CalcularFactorNormalizacion(min_Resp_buffer);
		}

		return muestraFiltrada;
	}

	private float CalcularFactorNormalizacion(float[] resp_buffer2) {
		int tAnterior = 0, tActual = 0, M_av4_indice = 0;
		float periodo = 0;
		int indCont[] = new int[4];
		float vTPeriodo[] = new float[REF_WIN_LENGTH];
		float difPasosUmbral = 0.0F;
		nPasosUmbralNormalizacion = 0;
		// Detectamos flancos positivos dentro de la ventana
		for (tActual = 1; tActual < REF_WIN_LENGTH; tActual++) {
			if (resp_buffer2[tActual] > threshold
					&& resp_buffer2[tActual - 1] <= threshold) {
```

```
                                periodo = tActual - tAnterior; // Periodo
                                tAnterior = tActual;
                                if (nPasosUmbralNormalizacion > 0) {
                                        // Filtro de media móvil de 4 Periodos
                                        indCont[M_av4_indice] = (int) periodo;
                                        M_av4_indice = (M_av4_indice + 1) % 4;
                                        if (nPasosUmbralNormalizacion >= 4) {
                                                periodo = 0;
                                                for (int n = 0; n < 4; n++)
                                                        periodo = periodo + indCont[n];

                                                periodo = periodo / 4;
                                        }
                                        vTPeriodo[nPasosUmbralNormalizacion] = periodo;
                                }
                                nPasosUmbralNormalizacion++;
                        }
                }

        mControladorLogica.EscribirArchivoPasosUmbral(vTPeriodo,nPasosUmbralNormalizacion
);

                // Calculamos el factor de normalizacion en la ventana de 40s como la
                // media del valor absoluto de la diferencias de los periodos
                for (int j = 1; j < nPasosUmbralNormalizacion - 1; j++) {
                        difPasosUmbral = difPasosUmbral
                                        + Math.abs(vTPeriodo[j] - vTPeriodo[j + 1]);
                }
                return difPasosUmbral / (nPasosUmbralNormalizacion - 2);
        }

        class TiempoKS {
                private int tiempo;
                private float ks;

                public TiempoKS(int tiempo, float ks) {
                        this.tiempo = tiempo;
                        this.ks = ks;
                }

                public int getTiempo() {
                        return tiempo;
                }

                public void setTiempo(int tiempo) {
                        this.tiempo = tiempo;
                }

                public float getKs() {
                        return ks;
                }

                public void setKs(float ks) {
                        this.ks = ks;
                }
        }

        private float CalcularDif(int NuevaMuestra) {

                int T_period, n, muestrasTranscuridas;
                float muestraFiltrada;
                tnew_index++;
                muestraFiltrada = filtrarPasoBajoMuestra(NuevaMuestra);
                muestraFiltrada = filtrarPasoAltoMuestra(muestraFiltrada) * signo;

                if (muestraFiltrada > threshold && OldSample <= threshold) {
                        nciclosRespiracion++;
                        // Calculate periodo
                        T_period = tnew_index - told_index;

                        // Calculamos muestrasTranscurridas desde el anterior flanco
                        // positivo
                        muestrasTranscuridas = tnew_index - told_index;
```

```
                        told_index = tnew_index;
                        Indcont[M_av4_index] = T_period;
                        // Filtro de media movil de 4 coeficientes
                        M_av4_index = (M_av4_index + 1) % 4;
                        // Buffer circular, calculamos el indice para el siguiente valor
                        difCalculado = 0;
                        // tiene que haber cinco flancos detectados, igual a cuatro
                        // periodos, para implementar el filtro de media movil
                        if (nciclosRespiracion >= 5) {
                                T_period = 0;
                                for (n = 0; n < 4; n++) {

                                        T_period = T_period + Indcont[n];
                                }
                                T_period = T_period / 4;

                        }
                        for (n = 1; n < 11; n++) {
                                Indt[n - 1] = Indt[n];
                        }
                        Indt[10] = T_period;
                        // tiene que haber quince flancos, igual a 14 periodos, para
                        // calcular las 10 absolutas diferencias
                        if (nciclosRespiracion >= 15) {
                                for (n = 0; n < 10; n++)// Calculamos el indice como la
media de
                                                                                // las últimas
10 absolutas diferencias
                                {
                                        difCalculado = difCalculado
                                                        + Math.abs(Indt[n] - Indt[n + 1]);
                                }
                                difCalculado = difCalculado / 10;
                                // Aplicamos el factor de normalizacion previamente
calculado
                                difCalculado = difCalculado / factorNormalizacion;
                                calcularEstadoAviso(muestrasTranscuridas, difCalculado);

                        }
                }
                OldSample = muestraFiltrada;
                return muestraFiltrada;
        }

        /**
         * Decide a partir de los valores del ks del último minuto si se ha de dar
         * alarma o no
         *
         * @param t_period
         *            longitud en muestras del periodo respiratorio
         * @param difCalculado
         *            valor del indice ks calculado
         */
        private void calcularEstadoAviso(int t_period, float difCalculado) {
                mListaTiempoKS.add(new TiempoKS(t_period, difCalculado));
                int sumaTiempo = 0;
                int contadorAvisoDormido = 0;
                int numeroCiclosAviso = mListaTiempoKS.size();
                // Recorremos la lista de valores de ks para contar el tiempo acumulado
                // y el numero de ciclos que están por encima del umbral de decision
                for (int i = 0; i < numeroCiclosAviso; i++) {
                        sumaTiempo += mListaTiempoKS.get(i).getTiempo();
                        if (mListaTiempoKS.get(i).getKs() > umbralFase1)
                                contadorAvisoDormido++;
                }
                if (sumaTiempo >= 300 * 40) { // Si el tiempo acumulado es mayor de un
                                                                                // minuto
decidimos el estado de alarma
                                                                                // por mayoría
```

```
                        bEstadoAlarma = (contadorAvisoDormido > numeroCiclosAviso / 2);
                }
                mControladorLogica.EscribirArchivoDecisionAlarma(sumaTiempo,
        bEstadoAlarma);

                while (sumaTiempo >= 300 * 40) {// Eliminamos los ultimos elementos para
                                                                        // deslizar la
ventana de un minuto
                        sumaTiempo -= mListaTiempoKS.get(0).getTiempo();
                        mListaTiempoKS.remove(0);
                }

        }

        /**
         *
         * @param muestra
         *            valor de la muestra recibida
         * @return valor de la muestra filtrada
         */
        public float ProcesarMuestra(int muestra) {

                float muestraFiltrada;
                if (N_samples < RETARDO) { // Fase de inicilizacion de los filtros
                        muestraFiltrada = iniciarFiltrado(muestra);
                        N_samples++;
                } else if (N_samples < (N_SAMPLES_INIT_WINDOW + RETARDO)
                                && !bMinimoAlcanzado) {// Fase de calculo de los parametros
del
                                                                        // algoritmo
                        muestraFiltrada = CalcularThreshold(muestra);

                } else
                        muestraFiltrada = CalcularDif(muestra);// Fase de calculo de Ks

                return muestraFiltrada;
        }

        private float iniciarFiltrado(int muestra) {
                float muestraFiltrada = filtrarPasoBajoMuestra(muestra);
                muestraFiltrada = filtrarPasoAltoMuestra(muestraFiltrada);
                return muestraFiltrada;
        }

        /*
         * public boolean EstadoAlarma() { return bEstadoAlarma; //return difActual
         * >= umbralFase1; }
         */

        public int getNumeroCiclos() {
                return nciclosRespiracion;
        }

        public float getThreshold() {
                return threshold;
        }

        public float getFactorNormalizacion() {

                return factorNormalizacion;
        }

        public boolean estadoIncializacion() {

                return tnew_index == 0;
        }

        public int getTiempoReferenciaInicial() {

                return nMuestraReferenciaInicio;
        }

        public int getTiempoReferenciaFinal() {
```

```
            return nMuestraReferenciaFinal;
        }

        public float getHtrIndex() {

            return htr;
        }

        public int getPasosPorUmbral() {

            return nPasosUmbralNormalizacion;
        }

        public boolean minimoAlcanzado() {

            return bMinimoAlcanzado;
        }

}
```

Contenido del fichero ArchivadorResultados.java

```
package logica;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

import android.os.Environment;

public class ArchivadorResultados {

	private String pathArchivoVariables;
	private boolean tieneSD;
	private String pathArchivoInicializacion;
	private String pathArchivoUmbral;
	private String pathArchivoDecision;

	public ArchivadorResultados() {
		// String newPath = Environment.getDataDirectory().getAbsolutePath();
		// Create a new subfolder under the current active folder
		// Create the subfolder

		pathArchivoVariables = null;
		tieneSD = compobarAlmacenamientoExterno();
		if (tieneSD)
			getPath();
	}

	private boolean compobarAlmacenamientoExterno() {
		boolean mExternalStorageWriteable = false;
		String state = Environment.getExternalStorageState();
		if (Environment.MEDIA_MOUNTED.equals(state)) {
			mExternalStorageWriteable = true;
		} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
			mExternalStorageWriteable = false;
		} else {
			mExternalStorageWriteable = false;
		}
		return mExternalStorageWriteable;
	}

	private void getPath() {
		File folder = null;
		Date date = new Date();
		String pattern = "yyyy_MM_dd_HH_mm";
		SimpleDateFormat format = new SimpleDateFormat(pattern);
		CharSequence simpleDate = format.format(date);

		String filenameVariables = "TestKS_HMI_Variables "
				+ simpleDate.toString() + ".txt";
		String filenameInicializacion = "TestKS_HMI_Inicializacion "
				+ simpleDate.toString() + ".txt";
		String filenamePasosUmbral = "TestKS_HMI_Pasos_Umbral "
				+ simpleDate.toString() + ".txt";
		String filenameDecisionAlarma = "TestKS_HMI_Decision_Alarma "
				+ simpleDate.toString() + ".txt";

		folder = new File(Environment.getExternalStorageDirectory(), "/Ks_HMI");
		if (!folder.isDirectory())
```

```
            folder.mkdir();
        pathArchivoVariables = folder.getPath();
        pathArchivoInicializacion = folder.getPath();
        pathArchivoUmbral = folder.getPath();
        pathArchivoDecision = folder.getPath();

        File fileVariables = new File(pathArchivoVariables, filenameVariables);
        if (fileVariables.exists()) {
            fileVariables.delete();
            try {
                fileVariables.createNewFile();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        pathArchivoVariables = fileVariables.getPath();

        File fileInicializacion = new File(pathArchivoInicializacion,
                filenameInicializacion);
        pathArchivoInicializacion = fileInicializacion.getPath();
        if (fileInicializacion.exists()) {
            fileInicializacion.delete();
            try {
                fileInicializacion.createNewFile();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        File fileUmbral = new File(pathArchivoUmbral, filenamePasosUmbral);
        pathArchivoUmbral = fileUmbral.getPath();
        if (fileUmbral.exists()) {
            fileUmbral.delete();
            try {
                fileUmbral.createNewFile();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        File fileDecision = new File(pathArchivoDecision,
                filenameDecisionAlarma);
        pathArchivoDecision = fileDecision.getPath();
        if (fileDecision.exists()) {
            fileDecision.delete();
            try {
                fileDecision.createNewFile();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

}

public void EscribirArchivoVariables(int[] lNumeroMuestra,
            int[] lValoresMuestra, float[] lValoresFiltrados,
```

```
                        float[] lValoresIndice, int nValoresGuardados, int[]
lNumeroCiclos) {
                // http://www.webprogramacion.com/171/csharp/crear-un-fichero-de-texto-
en-un-pocket-pc.aspx
                int nValoresArchivados = 0;
                String s;
                FileWriter fw;
                if (tieneSD) {
                        try {
                                fw = new FileWriter(pathArchivoVariables, true);
                                while (nValoresArchivados < nValoresGuardados) {

                                        s =
String.valueOf(lNumeroMuestra[nValoresArchivados])
                                                        + " "
                                                        +
String.valueOf(lValoresMuestra[nValoresArchivados])
                                                        + " "
                                                        +
String.valueOf(lValoresFiltrados[nValoresArchivados])
                                                        + " "
                                                        +
String.valueOf(lValoresIndice[nValoresArchivados])
                                                        + " "
                                                        +
String.valueOf(lNumeroCiclos[nValoresArchivados])
                                                        + "\n";
                                        fw.write(s);
                                        nValoresArchivados++;
                                }
                                fw.flush();
                                fw.close();
                        } catch (FileNotFoundException e) {
                                e.printStackTrace();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
        }

        public void EscribirArchivoIncializacion(int[] lNumeroMuestra,
                        int nValoresGuardados, float[] lValorThreshold,
                        float[] lFactorNormalizacion, int[] lTiempoReferenciaInicial,
                        int[] lTiempoReferenciaFinal, int[] lPasosPorUmbral, float[] lhtr)
{
                int nValoresArchivados = 0;
                String s;
                FileWriter fw;
                if (tieneSD) {
                        try {
                                fw = new FileWriter(pathArchivoInicializacion, true);
                                while (nValoresArchivados < nValoresGuardados) {

                                        s =
String.valueOf(lNumeroMuestra[nValoresArchivados])
                                                        + " "
                                                        +
String.valueOf(lValorThreshold[nValoresArchivados])
                                                        + " "
                                                        +
String.valueOf(lFactorNormalizacion[nValoresArchivados])
                                                        + " "
                                                        +
String.valueOf(lTiempoReferenciaInicial[nValoresArchivados])
```

```
                                                    + " "
                                                    +
String.valueOf(lTiempoReferenciaFinal[nValoresArchivados])
                                                    + " "
                                                    +
String.valueOf(lPasosPorUmbral[nValoresArchivados])
                                                    + " " +
String.valueOf(lhtr[nValoresArchivados])
                                                    + "\n";
                                        fw.write(s);
                                        nValoresArchivados++;
                                }
                                fw.flush();
                                fw.close();
                        } catch (FileNotFoundException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }

                }
        }

        public void EscribirArchivoPasosUmbral(float[] vTPeriodo,
                        int nPasosUmbralNormalizacion) {
                String s;
                FileWriter fw;
                if (tieneSD) {
                        try {
                                fw = new FileWriter(pathArchivoUmbral, true);
                                for (int j = 1; j < nPasosUmbralNormalizacion; j++) {
                                        s = String.valueOf(vTPeriodo[j]) + "\n";
                                        fw.write(s);
                                }
                                fw.flush();
                                fw.close();
                        } catch (FileNotFoundException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }

                }

        }

        public void EscribirArchivoDecisionAlarma(
                        int sumaTiempo, boolean bEstadoAlarma) {
                String s;
                FileWriter fw;
                if (tieneSD) {
                        try {
                                fw = new FileWriter(pathArchivoDecision, true);
                                s=String.valueOf(sumaTiempo)+ " "+
String.valueOf(bEstadoAlarma)+ "\n";
                                fw.write(s);
```

```
                        fw.flush();
                        fw.close();
                    } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }

                }

            }
        }
```

Contenido del fichero AvisadorSomnolencia.java

```
package logica;

import java.io.IOException;

import upc.KS_HMI.R;

import android.content.Context;
import android.media.MediaPlayer;
import android.telephony.ServiceState;
import android.telephony.SmsManager;
import android.telephony.TelephonyManager;

public class AvisadorSomnolencia {
	String numTlf;
	MediaPlayer player;
	Context appContext;
	// ThreadStart thrStart;
	Boolean AvisoOn;
	Boolean bTelefonoActivo;
	TelephonyManager telephonyManager;

	public AvisadorSomnolencia(String number, Context myContext,
			TelephonyManager telephonyManager2) {
		// thrStart = new ThreadStart(this.playSound);
		appContext = myContext;
		player = MediaPlayer.create(appContext, R.raw.beep_4);
		player.setLooping(true);
		AvisoOn = false;
		numTlf = number;
		telephonyManager = telephonyManager2;
		bTelefonoActivo = comprobarTelefono();
	}

	private boolean comprobarTelefono() {
		Boolean telefonoActivo = false;
		ServiceState myServiceState = new ServiceState();
		if (telephonyManager.getSimState() == TelephonyManager.SIM_STATE_READY
				&& myServiceState.getState() ==
ServiceState.STATE_IN_SERVICE)
			telefonoActivo = true;
		return telefonoActivo;
	}

	private void playAviso() {

		if (!AvisoOn) {
			AvisoOn = true;
			try {
				player.start();
			} catch (IllegalStateException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}

	}

	private void EnviarSMS() {
```

```
                if (bTelefonoActivo) {
                        SmsManager sms = SmsManager.getDefault();
                        sms.sendTextMessage(numTlf, null, "Te estas durmiendo", null,
null);
                }
        }

        public void PararAviso() {
                if (AvisoOn) {
                        player.stop();
                        player = MediaPlayer.create(appContext, R.raw.beep_4);
                        player.setLooping(true);
                        try {
                                player.prepare();
                        } catch (IllegalStateException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                        AvisoOn = false;
                }

        }

        public void Avisar() {
                this.playAviso();
                this.EnviarSMS();
                // LanzarAviso();

        }
}
```

Anexo C. Código fuente de la aplicación Windows Mobile

Contenido del fichero VistaInicial.cs

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.Media;
using System.IO;
using System.Threading;
using Microsoft.WindowsCE.Forms;
using Gestion;
namespace Ks_HMI
{
    namespace Vista
    {
        public partial class VistaInicial : Form
        {

            VistaPrincipal miform;
            AnalizadorDispositivo ntf;
            //Thread soundThread;
            //SoundPlayer player;
            public VistaInicial()
            {
                InitializeComponent();
                ntf = new AnalizadorDispositivo();
                SystemSettings.ScreenOrientation = ScreenOrientation.Angle0;
                iniciarCombo();
            }
            private void iniciarCombo()
            {
                Cursor.Current = Cursors.Default;
                string[] puertosDisponibles = SerialPort.GetPortNames();

                for (int i = 0; i < puertosDisponibles.Length; i++)
                {
                    comboBox1.Items.Add(puertosDisponibles[i]);
                }
            }
            private void Form1_FormClosing(Object sender)
            {
                miform.Close();
            }
            private void menuContinuarClick(object sender, EventArgs e)
            {
                String text = "Compruebe que los dispositivos están emparejados \n" +
                    " Inicio > Configuración > Ficha Conexiones > Bluetooth > " +
                    "Admnistrador de Bluetooth \n";
                String msgEstadoSistema = null;
                stbInicio.Text = "Espere, comprobando dispositivo ...";
                this.menuContinuar.Enabled = false;
                this.Refresh();
                ntf.ComprobarEstadoSistema(ref msgEstadoSistema);

                if (msgEstadoSistema != null) text += msgEstadoSistema;
                DialogResult result = MessageBox.Show(text, "Importante",
                      MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation,
                      MessageBoxDefaultButton.Button1);
```

```
                if (result == DialogResult.OK)
                {
                    stbInicio.Text = "Espere, iniciando sistema ...";
                    this.Refresh();
                    miform = new VistaPrincipal(this);
                    miform.InciarTimerSleep(ntf.ShortestTimeoutInterval());
                    string nomPuerto = obtenerNombrePuerto();
                    if (!miform.IniciarSistema(nomPuerto))
                    {
                        this.TratarError();
                        stbInicio.Text = "Pulse continuar para empezar";
                        this.menuContinuar.Enabled = true;
                    }
                }
                else
                {
                    this.Refresh();
                    stbInicio.Text = "Pulse continuar para empezar";
                    this.menuContinuar.Enabled = true;
                }
            }

            private string obtenerNombrePuerto()
            {

                if (comboBox1.SelectedItem == null)
                    comboBox1.SelectedIndex = 0;
                return (string)comboBox1.SelectedItem;
            }
            public void TratarError()
            {
                miform.Close();
                this.Show();
                String text = "Error al abrir el puerto. Compruebe que el nombre del
puerto es correcto";
                DialogResult resultError = MessageBox.Show(text, "Error Crítico",
                    MessageBoxButtons.OK, MessageBoxIcon.Hand,
                    MessageBoxDefaultButton.Button2);
                switch (resultError)
                {
                    case DialogResult.OK:
                        // Inicializamos Vista
                        this.Refresh();
                        break;
                }

            }
            private void menuSalirClick(object sender, EventArgs e)
            {
                // avsSueno.ParararAviso();

                String text = "¿Estás seguro de que quieres salir de la aplicación?";
                DialogResult result = MessageBox.Show(text, "Importante",
                    MessageBoxButtons.YesNo, MessageBoxIcon.Question,
                    MessageBoxDefaultButton.Button2);
                switch (result)
                {
                    case DialogResult.Yes:
                        this.Close();
                        break;
                    default:
                        break;
                }

            }
        }

    }
}
```

Contenido del fichero VistaPrincipal.cs

```
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;
using Microsoft.WindowsCE.Forms;
using System.Threading;
using System.Runtime.InteropServices;
using Gestion;
using Ks_HMI.Logica;

namespace Ks_HMI
{
    namespace Vista
    {
        class PosicionImagenesEstado
        {
            public int btmX, btmY, imgWitdh, imgHeight;
            public PosicionImagenesEstado()
            {

            }
            public PosicionImagenesEstado(int img1X, int img1Y, int imgWitdh,
                int imgHeigth)
            {
                this.btmX = img1X;
                this.btmY = img1Y;
                this.imgWitdh = imgWitdh;
                this.imgHeight = imgHeigth;
            }
        }
        public partial class VistaPrincipal : Form
        {
            VistaInicial ant_form;
            int estadoPintado;
            PosicionImagenesEstado pie;
            ControladorLogica interfazProc;
            // AnalizadorDispositivo ntf;
            Bitmap btmSemaforoVerde;
            Bitmap btmSemaforoGris;
            Bitmap btmSemaforoRojo;
            [DllImport("coredll.dll")]
            static extern void SystemIdleTimerReset();
            # region Constructor

            public VistaPrincipal(VistaInicial ant_form)
            {
                this.ant_form = ant_form;
                InitializeComponent();
                SystemSettings.ScreenOrientation = ScreenOrientation.Angle0;
                PosicionarElementosGraficos();
                //Relaciones de Aspecto
                //las primeras muestras lo pinte
                Rectangle resolucion = Screen.PrimaryScreen.WorkingArea;
                if (resolucion.Height <= 480 || resolucion.Width <= 240)
                {
                    btmSemaforoVerde = Properties.Resources.EstadoAtento;
                    btmSemaforoGris = Properties.Resources.EstadoInicializacion;
                    btmSemaforoRojo = Properties.Resources.EstadoDormido;
                }
                else
                {
                    btmSemaforoVerde = Properties.Resources.EstadoAtentoBig;
                    btmSemaforoGris = Properties.Resources.EstadoInicializacionBig;
                    btmSemaforoRojo = Properties.Resources.EstadoDormidoBig;
                }
                CalcularRelacionesAspecto();
                estadoPintado = -1;
```

```
            interfazProc = new ControladorLogica(this);

        }

        private void PosicionarElementosGraficos()
        {
            //The datatype of the Location property is a Point, so you would assign
a new Point to it.
            int alturaForm = panel2.Height;
            Point aux = new Point();
            aux.X = 0;
            aux.Y = alturaForm - stbEstadoSistema.Height - 1;
            stbEstadoSistema.Location = aux;

            aux.Y = stbEstadoSistema.Location.Y - lbltxtMuestras.Height - 1;
            lbltxtMuestras.Location = aux;

            aux.X = lbltxtMuestras.Width + 1;
            aux.Y = lbltxtMuestras.Location.Y;
            lbNumeroMuestras.Location = aux;

            aux.X = 0;
            aux.Y = lbltxtMuestras.Location.Y - lbltxtDif.Height - 1;
            lbltxtDif.Location = aux;

            aux.X = lbltxtDif.Width + 1;
            aux.Y = lbltxtDif.Location.Y;
            lblValorDif.Location = aux;
        }


        public bool IniciarSistema(string nombrePuerto)
        {
            bool ok = interfazProc.AbrirConexion(nombrePuerto);
            if (ok)
            {
                ant_form.Refresh();
                this.Show();
                ant_form.Hide();
                this.ActualizarStbProcesando();
                interfazProc.IniciarConexionProcesar();
            }
            return ok;
        }
        private void SalirForm()
        {
            this.Close();
            ant_form.Close();
        }

        # endregion

        #region Delegados y Eventos

        public delegate void CallbackSinParametros();
        private void BarraEstadoMostrandoEstado()
        {
            this.stbEstadoSistema.Text = "Mostrando estado somnolencia actual";
        }
        public delegate void StringCallback(string msg);
        private void MostrarMsgConfirmacion(string msg)
        {
            DialogResult result = MessageBox.Show(msg, "Error Crítico",
                   MessageBoxButtons.OK, MessageBoxIcon.Exclamation,
                   MessageBoxDefaultButton.Button2);
            switch (result)
            {
                case DialogResult.OK:
                    SalirAplicacion();
                    break;
            }
        }
        public delegate void TwoStringCallback(string text1, string text2);
```

```
            private void SetContadoresFlash(string textDif, string textMuestras)
            {
                // Set the textbox text.
                lblValorDif.Text = textDif;
                lbNumeroMuestras.Text = textMuestras;
                Graphics grpCuadro = this.panel2.CreateGraphics();
                grpCuadro.Clear(SystemColors.Window);
                grpCuadro.Dispose();
            }
            public delegate void TwoStringBitmapCallback(string text1, string text2,
Bitmap btm);
            private void SetContadoresSemaforos(string textDif, string textMuestras,
Bitmap btm)
            {
                // Set the textbox text.
                lblValorDif.Text = textDif;
                lbNumeroMuestras.Text = textMuestras;
                Graphics grpCuadro = this.panel2.CreateGraphics();
                grpCuadro.DrawImage(btm, pie.btmX, pie.btmY);
                grpCuadro.Dispose();
            }
            public void LimpiarGrafico()
            {
                Graphics grpCuadro = this.panel2.CreateGraphics();
                grpCuadro.Clear(SystemColors.Window);
                grpCuadro.Dispose();
            }
            #endregion

            #region Metodos
            private void CalcularRelacionesAspecto()
            {
                if (pie == null) pie = new PosicionImagenesEstado();
                pie.imgWitdh = btmSemaforoVerde.Width;
                pie.imgHeight = btmSemaforoVerde.Height;
                pie.btmX = (int)(Screen.PrimaryScreen.WorkingArea.Width - pie.imgWitdh)
/ 2;
                pie.btmY = (int)(Screen.PrimaryScreen.WorkingArea.Height - pie.imgHeight
- stbEstadoSistema.Height - lbNumeroMuestras.Height * 2) / 2;
            }
            private void menuSalir_Click(object sender, EventArgs e)
            {
                String text = "¿Estás seguro de que quieres salir de la aplicación?";
                DialogResult result = MessageBox.Show(text, "Importante",
                    MessageBoxButtons.YesNo, MessageBoxIcon.Question,
                    MessageBoxDefaultButton.Button2);
                switch (result)
                {
                    case DialogResult.Yes:
                        // Inicializamos Vista
                        this.Refresh();
                        tmrPreventSleep.Enabled = false;
                        interfazProc.ParararSistema();
                        SalirForm();
                        break;
                    default:
                        this.Refresh();
                        break;
                }
            }
            # endregion


            public void ActualizarVistaPinta(float difCalculado, int estadoCalculado,
int muestrasRecibidas)
            {
                if (estadoCalculado != estadoPintado || muestrasRecibidas % 20 != 0)
                {
                    estadoPintado = estadoCalculado;
```

```
                    ActualizarContadoresSemaforo(difCalculado, estadoCalculado,
muestrasRecibidas);
                }
                else
                {
                    ActualizarContadoresFlash(difCalculado, muestrasRecibidas);
                }
            }
            private void ActualizarContadoresSemaforo(float difCalculado, int
estadoCalculado, int muestrasRecibidas)
            {
                if (!this.IsDisposed)
                {
                    switch (estadoCalculado)
                    {
                        case -1:
                            this.Invoke(new TwoStringBitmapCallback(
                                    this.SetContadoresSemaforos), new object[] {
difCalculado.ToString(), muestrasRecibidas.ToString(), btmSemaforoGris });
                            break;

                        case 0:
                            this.Invoke(new TwoStringBitmapCallback(
                                    this.SetContadoresSemaforos), new object[] {
difCalculado.ToString(), muestrasRecibidas.ToString(), btmSemaforoVerde });
                            break;
                        case 1:
                            this.Invoke(new TwoStringBitmapCallback(
                                    this.SetContadoresSemaforos), new object[] {
difCalculado.ToString(), muestrasRecibidas.ToString(), btmSemaforoRojo });
                            break;
                        default:
                            break;
                    }
                }
            }
            private void ActualizarContadoresFlash(float difCalculado, int
muestrasRecibidas)
            {
                this.Invoke(new TwoStringCallback(this.SetContadoresFlash),
                      new object[] { difCalculado.ToString(),
muestrasRecibidas.ToString() });
            }

            private void ActualizarStbProcesando()
            {
                this.Invoke(new CallbackSinParametros(this.BarraEstadoMostrandoEstado));
            }
            public void MostrarMensaje(string msg)
            {
                DialogResult result = MessageBox.Show(msg, "Importante",
                      MessageBoxButtons.OK, MessageBoxIcon.Exclamation,
                      MessageBoxDefaultButton.Button1);
                this.Refresh();
            }
            public void AvisarErrorInterfaz()
            {
                string msg = "Error de comunicación. Compruebe la conexión. " +
                 "La aplicación se cerrará automaticamente";
                this.Invoke(new StringCallback(this.MostrarMsgConfirmacion), new
Object[] { msg });
            }
            public void AvisarErrorFormato()
            {
                string msg = "Error de Formato. Compruebe la conexión. " +
                 "La aplicación se cerrará automaticamente";
                this.Invoke(new StringCallback(this.MostrarMsgConfirmacion),
                    new Object[] { msg });
            }
            public void AvisarErrorMemoria()
            {
                string msg = "Memoria Insuficiente. " +
                 "La aplicación se cerrará automaticamente";
```

```
                this.Invoke(new StringCallback(this.MostrarMsgConfirmacion),
                    new Object[] { msg });
            }

            public void AvisarErrorFatal()
            {
                string msg = "Error Fatal. " +
                 "La aplicación se cerrará automaticamente";
                this.Invoke(new StringCallback(this.MostrarMsgConfirmacion),
                    new Object[] { msg });
            }
            public void AvisarMuestrasPerdidas()
            {
                string msg = "Muestras perdidas. " +
                  "La aplicación se cerrará automaticamente";
                this.Invoke(new StringCallback(this.MostrarMsgConfirmacion),
                    new Object[] { msg });
            }
            public void SalirAplicacion()
            {
                if (!this.IsDisposed) this.Invoke(new
CallbackSinParametros(this.SalirForm));
            }
            private void resetTimer_Tick(object sender, EventArgs e)
            {
                SystemIdleTimerReset();
            }
            public void InciarTimerSleep(int interval)
            {
                tmrPreventSleep.Interval = interval;
                tmrPreventSleep.Enabled = true;
                this.tmrPreventSleep.Tick += new System.EventHandler(resetTimer_Tick);
            }



        }
    }
}
```

Contenido del fichero LectorPuertoSerie.cs

```
using System;
using System.Threading;
using System.Globalization;
using System.IO.Ports;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using Gestion;
using Ks_HMI.Logica;
namespace Ks_HMI
{

    namespace Comunicacion
    {
        public class LectorPuertoSerie : IDisposable
        {
            private SerialPort serialPort;
            private ControladorLogica interfazProc;
            private Thread myThread;
            private bool estaEsperando;
            private bool estaConectado;
            private int nTramaAnterior;

            public LectorPuertoSerie(ControladorLogica interfazProc)
            {
                serialPort = new SerialPort();
                this.interfazProc = interfazProc;
                estaConectado = false;
                nTramaAnterior = -1;
                myThread = new Thread(new ThreadStart(this.spDataReceivedBanda));
                //myThread = new Thread(new ThreadStart(this.spDataReceivedTimeout));
            }

            # region CambiarEvento
            //Este meotodo es llamado desde la interfaz poe si esta bloqueado
            //por el timeout
            public void ParararLectura()
            {
                // bContinuarLectura = false;
                //serialPort.DataReceived -= serialPort_DataReceived;
                if (estaEsperando) this.Close();
            }
            # endregion

            public bool ConfigurarAbrirPuerto(string nomPuerto)
            {// Instancia y lo abre
                bool bAbierto = false;
                if (!serialPort.IsOpen)
                {
                    try
                    {
                        serialPort.Handshake = Handshake.None;
                        serialPort.Parity = Parity.None;
                        serialPort.StopBits = StopBits.One;
                        serialPort.ReadTimeout = 10 * 1000;
                        serialPort.PortName = nomPuerto;
                        serialPort.ReadBufferSize = 4096;
                        serialPort.Open();

                        bAbierto = true;
                        estaConectado = bAbierto;
                        serialPort.ErrorReceived += serialPort_ErrorReceived;
                    }
                    catch (Exception)
                    {
                        //error emparejamiento
                        bAbierto = false;
                    }
                }
                return bAbierto;
            }
```

```
        public void IniciarLectura()
        {
            myThread.Priority = ThreadPriority.AboveNormal;
            myThread.Start();
        }

        private void spDataReceivedTimeout()
        {
            string[] datosLeidos;
            byte[] buffer;
            int bufferSize = serialPort.ReadBufferSize;
            int bytesLeidos;
            int MuestrasLeidas;
            string readmessage;
            int bytesNoLeidos;
            buffer = new byte[bufferSize];
            estaEsperando = false;
            try
            {
                while (serialPort.IsOpen && estaConectado)
                {
                    bytesNoLeidos = serialPort.BytesToRead;
                    if (bytesNoLeidos == 0)
                    {
                        estaEsperando = true;
                        //bytesLeidos = serialPort.Read(buffer, 0, bufferSize);
                        readmessage = serialPort.ReadLine();
                        estaEsperando = false;
                        interfazProc.tratarMuestraObtenida(int.Parse(readmessage));
                        interfazProc.MostrarResultado(0);
                    }
                    else
                    {
                        if (bytesNoLeidos <= bufferSize)
                        {
                            bytesLeidos = serialPort.Read(buffer, 0, bytesNoLeidos);
                        }
                        else
                        {
                            bufferSize = bytesNoLeidos;
                            buffer = null;
                            buffer = new byte[bufferSize];
                            bytesLeidos = serialPort.Read(buffer, 0, bytesNoLeidos);
                        }
                        readmessage = ASCIIEncoding.ASCII.GetString(buffer, 0,
bytesLeidos);
                        datosLeidos = readmessage.Split('\n');
                        MuestrasLeidas = datosLeidos.Length;
                        for (int i = 0; i < MuestrasLeidas; i++)
                        {
                            if (datosLeidos[i] != string.Empty)
                            {

interfazProc.tratarMuestraObtenida(int.Parse(datosLeidos[i]));
                            }
                        }
                        datosLeidos = null;
                        interfazProc.MostrarResultado(0);
                    }
                }
                SalirSistema();
            }
            catch (FormatException)
            {
                CerrarSistema();
                if (estaConectado) interfazProc.AvisarErrorFormato();

            }
            catch (TimeoutException)
```

```
                {
                    CerrarSistema();
                    if (estaConectado) interfazProc.AvisarErrorLectura();
                }
                catch (OutOfMemoryException)
                {
                    CerrarSistema();
                    if (estaConectado) interfazProc.AvisarErrorMemoria();

                }
                catch (Exception)
                {
                    CerrarSistema();
                    if (estaConectado) interfazProc.AvisarErrorFatal();
                }
                Thread.CurrentThread.Abort();
            }

            private void serialPort_ErrorReceived(object sender,
SerialErrorReceivedEventArgs e)
            {
                interfazProc.AvisarErrorLectura();
            }
            private void spDataReceivedBanda()
            {
                byte[] buffer;
                int bytesLeidos, bytesNoLeidos, bytesPorLeer;
                int resto = 0;
                int nMuestrasPerdidas = 0;
                int bufferSize = 4096;
                buffer = new byte[bufferSize];
                estaEsperando = false;
                Boolean error = false;

                try
                {
                    if (serialPort.IsOpen && estaConectado)
                    {
                        //Espero comando de aqusicion pero  no proceso esa info
                        estaEsperando = true;
                        EnviarSecuenciaIncializacion();
                        estaEsperando = false;
                    }
                    while (serialPort.IsOpen && estaConectado && !error)
                    {
                        estaEsperando = true;
                        // bytesLeidos = serialPort.Read(buffer, 0, bytesNoLeidos);
                        bytesNoLeidos = serialPort.BytesToRead;
                        if (bytesNoLeidos < 4)
                        {
                            bytesLeidos = serialPort.Read(buffer, 0, 4);
                            while (bytesLeidos < 4)
                            {
                                bytesPorLeer = 4 - bytesLeidos;
                                bytesLeidos += serialPort.Read(buffer, bytesLeidos,
                                        bytesPorLeer);
                            }
                        }
                        else if (bytesNoLeidos <= bufferSize)
                        {
                            resto = bytesNoLeidos % 4;
                            bytesLeidos = serialPort.Read(buffer, 0, bytesNoLeidos -
resto);
                        }
                        else
                        {
                            bytesLeidos = serialPort.Read(buffer, 0, bufferSize);

                        }
                        estaEsperando = false;
                        if ((bytesLeidos % 4) == 0)
                        {
```

```
                            nMuestrasPerdidas=ParsearDatosRecibidos(buffer, 0,
bytesLeidos);
                            interfazProc.MostrarResultado(nMuestrasPerdidas);
                        }
                        else
                        {
                            error = true;

                        }
                    }
                    if (error)
                    {
                        CerrarSistema();
                        interfazProc.AvisarErrorFormato();
                    }
                    else
                    {
                        SalirSistema();
                    }

                }
                catch (FormatException)
                {
                    CerrarSistema();
                    if (!estaConectado) interfazProc.AvisarErrorFormato();

                }
                catch (TimeoutException)
                {
                    CerrarSistema();
                    if (estaConectado) interfazProc.AvisarErrorLectura();
                }
                catch (OutOfMemoryException)
                {
                    CerrarSistema();
                    if (estaConectado) interfazProc.AvisarErrorMemoria();
                }
                catch (Exception)
                {
                    CerrarSistema();
                    if (estaConectado) interfazProc.AvisarErrorFatal();
                }
                Thread.CurrentThread.Abort();
            }

            private void EnviarSecuenciaIncializacion()
            {
                byte[] bcomandos = new byte[20];
                string comando = "V";
                System.Text.ASCIIEncoding codificador = new System.Text.ASCIIEncoding();
                bcomandos = codificador.GetBytes(comando);
                serialPort.Write(bcomandos, 0, 1);
                comando = serialPort.ReadLine();
                comando = "@START,0040,01,12;";
                bcomandos = codificador.GetBytes(comando);
                serialPort.Write(bcomandos, 0, 18);
            }
            private int comprobarNumeroTrama(int nTramaActual)
            {
                int difTrama = nTramaActual - nTramaAnterior;
                int muestrasPerdidas = 0;
                if (nTramaAnterior != -1
                        && !(nTramaActual == 0 && nTramaAnterior == 127))
                {
                    if (difTrama > 1)
                    {
                        muestrasPerdidas = difTrama - 1;
                    }
                    else if (difTrama < 0)
```

```
                    {
                        muestrasPerdidas = nTramaActual + 127 - nTramaAnterior;
                    }
                }
                return muestrasPerdidas;
            }
            private int ParsearDatosRecibidos(byte[] buffer, int offset, int count)
            {
                int x = 0, y = 0, bytesParseados=0, muestrasPerdidas = 0;
                int valorMuestra;
                while (bytesParseados < count)
                {
                    int nTrama = (int)(buffer[bytesParseados] & 0x7F);
                    x = (int)(buffer[bytesParseados + 1] & 0xFF);
                    y = (int)(buffer[bytesParseados + 2] & 0x0F);
                    valorMuestra = x + y * 256;
                    muestrasPerdidas += comprobarNumeroTrama(nTrama);
                    interfazProc.tratarMuestraObtenida(valorMuestra);
                    nTramaAnterior = nTrama;
                    bytesParseados += 4;
                }
                return muestrasPerdidas;
            }

            #region IDisposable Members
            protected void Dispose(bool disposing)
            {
                if (disposing)
                {
                    // Code to clean up managed resources
                    if (serialPort != null) serialPort.Dispose();
                }
                serialPort = null;
                // Code to clean up unmanaged resources
            }
            public void Dispose()
            {
                Dispose(true);
                GC.SuppressFinalize(this);
            }
            #endregion
            private void CerrarSistema()
            {
                Close();
                Dispose();
                interfazProc.CerrarControlador();
            }
            private void SalirSistema()
            {
                CerrarSistema();
                interfazProc.SalirAplicacion();
            }
            private void Close()
            {
                if (serialPort != null)
                {
                    if (serialPort.IsOpen)
                    {
                        System.Text.ASCIIEncoding codificador = new
System.Text.ASCIIEncoding();
                        serialPort.Write(codificador.GetBytes("R"), 0, 1);
                        serialPort.DiscardInBuffer();
                        serialPort.Close();
                        estaConectado = false;
                    }
                }
            }
        }
    }
}
```

Contenido del fichero ControladorLogica.cs

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using Ks_HMI.Vista;
using Ks_HMI.Comunicacion;
namespace Ks_HMI
{
    namespace Logica
    {
        public class ControladorLogica
        {
            LectorPuertoSerie lsp;
            ProcesadorRespiracion myResp;
            VistaPrincipal vst;
            AvisadorSomnolencia avisadorSomnolencia;
            ArchivadorResultados ArchivoResultado;
            private int[] lNumeroMuestra;
            private int[] lValoresMuestra;
            private float[] lValoresFiltrados;
            private float[] lValoresIndice;
            private int[] lNumeroCiclos;
            private float[] lValorThreshold;
            private float[] lFactorNormalizacion;
            private int[] lTiempoReferenciaInicial;
            private int[] lTiempoReferenciaFinal;
            private int[] lPasosPorUmbral;
            private float[] lHtr;
            int maxValoresGuardados = 100;
            int nValoresGuardados;
            private bool bAvisoMuestrasPerdidas;

            # region Constructores
            public ControladorLogica(VistaPrincipal vst)
            {
                this.vst = vst;
                ArchivoResultado = new ArchivadorResultados();
                if (lsp == null) lsp = new LectorPuertoSerie(this);
                if (myResp == null) myResp = new ProcesadorRespiracion(this);

                avisadorSomnolencia = new AvisadorSomnolencia("690107933");
                ArchivoResultado = new ArchivadorResultados();
                lNumeroMuestra = new int[maxValoresGuardados];
                lValoresMuestra = new int[maxValoresGuardados];
                lValoresFiltrados = new float[maxValoresGuardados];
                lValoresIndice = new float[maxValoresGuardados];
                lNumeroCiclos = new int[maxValoresGuardados];
                lValorThreshold = new float[maxValoresGuardados];
                lFactorNormalizacion = new float[maxValoresGuardados];
                lTiempoReferenciaInicial = new int[maxValoresGuardados];
                lTiempoReferenciaFinal = new int[maxValoresGuardados];
                lPasosPorUmbral = new int[maxValoresGuardados];
                lHtr = new float[maxValoresGuardados];
                nValoresGuardados = 0;
                bAvisoMuestrasPerdidas = false;
                //avisadorActivo = false;
            }
            #endregion

            #region Conexion
            //Comprueba que el nombre del Puerto es correcto llama la VistaIncial
            public bool AbrirConexion(string nombrePuerto)
            {
                bool puertoAbierto = lsp.ConfigurarAbrirPuerto(nombrePuerto);
                if (!puertoAbierto) lsp.Dispose();
                return puertoAbierto;
```

```
}
public void IniciarConexionProcesar()
{
    lsp.IniciarLectura();
}
# endregion


#region Procesado y Guardar
public void tratarMuestraObtenida(int muestra)
{
    float muestraFiltrada = myResp.ProcesarMuestra(muestra);
    lNumeroMuestra[nValoresGuardados] = myResp.GetNumeroMuestras();
    lValoresIndice[nValoresGuardados] = myResp.getDif();
    lValoresMuestra[nValoresGuardados] = muestra;
    lValoresFiltrados[nValoresGuardados] = muestraFiltrada;
    lNumeroCiclos[nValoresGuardados] = myResp.getNumeroCiclos();
    if (myResp.estadoIncializacion())
    {
        lValorThreshold[nValoresGuardados] = myResp.getThreshold();
        lFactorNormalizacion[nValoresGuardados] = myResp
                .getFactorNormalizacion();
        lTiempoReferenciaInicial[nValoresGuardados] = myResp
                .getTiempoReferenciaInicial();
        lTiempoReferenciaFinal[nValoresGuardados] = myResp
                .getTiempoReferenciaFinal();
        lPasosPorUmbral[nValoresGuardados] = myResp.getPasosPorUmbral();
        lHtr[nValoresGuardados] = myResp.getHtrIndex();
        nValoresGuardados++;
        if (nValoresGuardados >= maxValoresGuardados
                || myResp.minimoAlcanzado())
        {
            ArchivoResultado.EscribirArchivoIncializacion(lNumeroMuestra,
                    lValoresMuestra, lValoresFiltrados, nValoresGuardados,
                    lValorThreshold, lFactorNormalizacion,
                    lTiempoReferenciaInicial, lTiempoReferenciaFinal,
                    lPasosPorUmbral, lHtr);

            ArchivoResultado.EscribirArchivoVariables(lNumeroMuestra,
                    lValoresMuestra, lValoresFiltrados, lValoresIndice,
                    nValoresGuardados, lNumeroCiclos);
            nValoresGuardados = 0;
        }
    }
    else
    {
        nValoresGuardados++;
        if (nValoresGuardados >= maxValoresGuardados)
        {
            ArchivoResultado.EscribirArchivoVariables(lNumeroMuestra,
                    lValoresMuestra, lValoresFiltrados, lValoresIndice,
                    nValoresGuardados, lNumeroCiclos);
            nValoresGuardados = 0;
        }

    }
}
# endregion

#region Vista y Avisar
public void MostrarResultado(int nMuestrasPerdidas)
{

    if (nMuestrasPerdidas == 0)
    {
        vst.ActualizarVistaPinta(myResp.getDif(),
                myResp.consularEstadoAlarma(), myResp.GetNumeroMuestras());

        if (myResp.consularEstadoAlarma() == 1)
        {
            avisadorSomnolencia.Avisar();
        }
        else
```

```
                    avisadorSomnolencia.PararAviso();
                }
                else if (!bAvisoMuestrasPerdidas && nMuestrasPerdidas > 0)
                {
                    bAvisoMuestrasPerdidas = true;
                    vst.AvisarMuestrasPerdidas();

                    avisadorSomnolencia.PararAviso();
                }
            }
            # endregion

            #region Cerrar
            public void CerrarControlador()
            {
                myResp = null;
                avisadorSomnolencia.PararAviso();
            }
            public void PararSistema()
            {
                lsp.PararLectura();
            }
            public void AvisarErrorLectura()
            {
                //avisadorSomnolencia.PararAviso();
                vst.AvisarErrorInterfaz();
            }
            public void AvisarErrorMemoria()
            {
                vst.AvisarErrorMemoria();
            }
            public void AvisarErrorFormato()
            {
                vst.AvisarErrorFormato();
            }
            public void AvisarErrorFatal()
            {
                vst.AvisarErrorFatal();
            }
            # endregion

            internal void SalirAplicacion()
            {
                vst.SalirAplicacion();
            }
            public void EscribirArchivoPasosUmbral(float[] vTPeriodo,
            int nPasosUmbralNormalizacion)
            {
                ArchivoResultado.EscribirArchivoPasosUmbral(vTPeriodo,
                        nPasosUmbralNormalizacion);

            }
            public void EscribirArchivoDecisionAlarma(
           List<TiempoKS> mListaTiempoKS, int sumaTiempo,
            bool bEstadoAlarma)
            {
                ArchivoResultado.EscribirArchivoDecisionAlarma(mListaTiempoKS,
                        sumaTiempo, bEstadoAlarma);

            }
        }

    }
}
```

Contenido del fichero ProcesadorRespiracion.cs

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Collections;

namespace Ks_HMI
{
    namespace Logica
    {
        public class TiempoKS
        {
            private int tiempo;
            private float ks;

            public TiempoKS(int tiempo, float ks)
            {
                this.tiempo = tiempo;
                this.ks = ks;
            }

            public int getTiempo()
            {
                return tiempo;
            }

            public void setTiempo(int tiempo)
            {
                this.tiempo = tiempo;
            }

            public float getKs()
            {
                return ks;
            }

            public void setKs(float ks)
            {
                this.ks = ks;
            }
        }

        class ProcesadorRespiracion
        {
            private int FS = 40;
            private int RETARDO;
            private int REF_WIN_LENGTH; // Longitud de la ventana de referencia en
            // muestras 40s *fs
            private int PCTIL = 60; // Percentil valor para determinar el threshold
            private int N_SAMPLES_INIT_WINDOW;
            private int tnew_index;
            private int told_index;
            private int M_av4_index;
            private int nciclosRespiracion;
            private int umbralFase1;
            private float threshold;
            private float OldSample;
            private float difCalculado;
            private float htr;
            private int index_buff;
            private int N_samples;
            private float htr_old;
            private float[] Resp_buffer;
            private float[] min_Resp_buffer;
            private float[] aux_Resp_buffer;
            private int[] Indt;
            private int[] Indcont;
            private bool bMinimoAlcanzado;
            // Coeficientes filtro paso bajo
            private float[] bCoeficientePasoBajo;
```

```
private float[] aCoeficientePasoBajo;
private float xn1Bajo = 0, xn2Bajo = 0, yn1Bajo = 0, yn2Bajo = 0;
// Coeficientes filtro paso alto
private float[] bCoeficientePasoAlto;
private float[] aCoeficientePasoAlto;
private float xn1Alto = 0, xn2Alto = 0, yn1Alto = 0, yn2Alto = 0;
private int signo;
private float factorNormalizacion;
private int nMuestraReferenciaFinal;
private int nMuestraReferenciaInicio;
private int nPasosUmbralNormalizacion;
private List<TiempoKS> mListaTiempoKS;
private ControladorLogica mControladorLogica;
private bool bEstadoAlarma;

public ProcesadorRespiracion(ControladorLogica mControladorLogica)
{
    tnew_index = 0;
    told_index = 0;
    M_av4_index = 0;
    difCalculado = 0;
    threshold = 0;
    umbralFase1 = 6;
    nciclosRespiracion = 0;
    RETARDO = 20 * FS;
    REF_WIN_LENGTH = 40 * FS;
    N_SAMPLES_INIT_WINDOW = 900 * FS;
    index_buff = 0;
    N_samples = 0;
    htr_old = float.MaxValue;
    Resp_buffer = new float[REF_WIN_LENGTH];
    min_Resp_buffer = new float[REF_WIN_LENGTH];
    aux_Resp_buffer = new float[REF_WIN_LENGTH];
    Indt = new int[11];
    Indcont = new int[4];
    mListaTiempoKS = new List<TiempoKS>();
    this.mControladorLogica = mControladorLogica;
    bMinimoAlcanzado = false;
    bEstadoAlarma = false;
    // [b,a]=butter(2,1/(40/2),'low')
    bCoeficientePasoBajo = new float[3];
    // bCoeficientePasoBajo[0] = 0.0055427172102806566;
    bCoeficientePasoBajo[0] = 0.0055427172F;

    // bCoeficientePasoBajo[1] = 0.011085434420561313;
    bCoeficientePasoBajo[1] = 0.0110854344F;

    // bCoeficientePasoBajo[2] = 0.0055427172102806566;
    bCoeficientePasoBajo[2] = 0.0055427172F;


    aCoeficientePasoBajo = new float[3];
    aCoeficientePasoBajo[0] = 1;
    // aCoeficientePasoBajo[1] = -1.778631777824585;
    aCoeficientePasoBajo[1] = -1.7786317778F;
    // aCoeficientePasoBajo[1] = -1.77863178;

    // aCoeficientePasoBajo[2] = 0.80080264666570777;
    aCoeficientePasoBajo[2] = 0.8008026467F;

    // > [b,a]=butter(2,0.1/(40/2),'high')
    bCoeficientePasoAlto = new float[3];
    // bCoeficientePasoAlto[0] = 0.98895424806714005;
    bCoeficientePasoAlto[0] = 0.9889542481F;

    // bCoeficientePasoAlto[1] = -1.9779084961342801;
    bCoeficientePasoAlto[1] = -1.9779084961F;
```

```
        // bCoeficientePasoAlto[2] = 0.98895424806714005;
        bCoeficientePasoAlto[2] = 0.9889542481F;

        aCoeficientePasoAlto = new float[3];
        aCoeficientePasoAlto[0] = 1;
        // aCoeficientePasoAlto[1] = -1.9777864837767638;
        aCoeficientePasoAlto[1] = -1.9777864838F;
        //

        // aCoeficientePasoAlto[2] = 0.97803050849179607;
        aCoeficientePasoAlto[2] = 0.9780305085F;


        signo = 1;

    }

    /**
     * Elimina la media de los valores del array de entrada
     *
     * @param aux_Resp_buffer
     *            array de muestras de respiracion
     */
    private void remove_mean(float[] aux_Resp_buffer)
    {
        int k;
        float media = 0.0F;
        for (k = 0; k < REF_WIN_LENGTH; k++)
            media = media + aux_Resp_buffer[k];
        media = media / REF_WIN_LENGTH;

        for (k = 0; k < REF_WIN_LENGTH; k++)
            aux_Resp_buffer[k] = aux_Resp_buffer[k] - media;
    }

    private void copy_buffer(float[] resp_buffer, float[] aux_Resp_buffer)
    {
        int k;
        for (k = 0; k < REF_WIN_LENGTH; k++)
            aux_Resp_buffer[k] = resp_buffer[k];
    }

    /**
     * Calcula el indice de heteroestacidad
     *
     * @param resp_buffer
     *            array de muestras respiratorias
     * @return indice de heteroestacidad
     */
    private float htrstest(float[] resp_buffer)
    {
        int j, k;
        float square_sum = 0, cumsum = 0;
        float htrs_index = 0.0F, d_index = 0.0F;
        this.copy_buffer(resp_buffer, aux_Resp_buffer);
        this.remove_mean(aux_Resp_buffer); // Elimina la media en los datos
        for (k = 0; k < REF_WIN_LENGTH; k++) // Calcula la suma de los cuadrados
        // del array de entrada
        {
            square_sum = square_sum + aux_Resp_buffer[k] * aux_Resp_buffer[k];
        }

        for (j = 0; j < REF_WIN_LENGTH; j++) // Substract mean value and
        // calculate square sum
        {
            // Calculamos la suma acumulada de los cuadrados para cada indice
            // del array
            cumsum = cumsum + (aux_Resp_buffer[j] * aux_Resp_buffer[j]);
            d_index = (cumsum * REF_WIN_LENGTH) / square_sum - j;
            if (Math.Abs(d_index) > htrs_index)
            {
                htrs_index = Math.Abs(d_index);
            }
```

```
    }

    htrs_index = htrs_index / (REF_WIN_LENGTH);
    return htrs_index;
}

void insertionSort(float[] aux_Resp_buffer, int array_size)
{
    int i, j;
    float index;
    for (i = 1; i < array_size; ++i)
    {
        index = aux_Resp_buffer[i];
        for (j = i; j > 0 && aux_Resp_buffer[j - 1] > index; j--)
            aux_Resp_buffer[j] = aux_Resp_buffer[j - 1];

        aux_Resp_buffer[j] = index;
    }
}

private float percentil(float[] R_buff, int pctl)
{
    int k;
    float resul;
    for (k = 0; k < REF_WIN_LENGTH; k++)
        aux_Resp_buffer[k] = R_buff[k]; // Make a temporal copy of the array
    // to sort
    insertionSort(aux_Resp_buffer, REF_WIN_LENGTH); // Sort array
    k = (pctl * REF_WIN_LENGTH / 100);
    resul = aux_Resp_buffer[k];

    return resul; // Return amplitude value that has %pctl of array samples
    // below
}

void shift_buffer(float[] resp_buffer, float muestra)
{
    int k;
    for (k = 0; k < (REF_WIN_LENGTH - 1); k++)
        resp_buffer[k] = resp_buffer[k + 1];
    resp_buffer[k] = muestra;
}

/**
 * Devuelve el estado de la alarma
 *
 * @return -1 inicialización, 0 estado despierto, 1 estado dormido
 */
public int consularEstadoAlarma()
{

    if (N_samples < (N_SAMPLES_INIT_WINDOW + RETARDO) && !bMinimoAlcanzado)
        return -1;
    else if (!bEstadoAlarma)
        return 0;
    else
        return 1;
}

public void setFrecuencia(int p)
{
    FS = p;
}

public void setVentana(int p)
{
    REF_WIN_LENGTH = p;
}
```

```
public void setPercentil(int p)
{
    PCTIL = p;
}

public void setUmbralFase1(int p)
{
    umbralFase1 = p;
}

public int GetNumeroMuestras()
{

    return tnew_index + N_samples;
}

public float getDif()
{
    return difCalculado;
}

private float filtrarPasoBajoMuestra(int nuevaMuestra)
{
    float muestraFiltrada = nuevaMuestra * bCoeficientePasoBajo[0]
            + xn1Bajo * bCoeficientePasoBajo[1] + xn2Bajo
            * bCoeficientePasoBajo[2] - aCoeficientePasoBajo[1] * yn1Bajo
            - aCoeficientePasoBajo[2] * yn2Bajo;
    xn2Bajo = xn1Bajo;
    xn1Bajo = nuevaMuestra;
    yn2Bajo = yn1Bajo;
    yn1Bajo = muestraFiltrada;

    return muestraFiltrada;
}

private float filtrarPasoAltoMuestra(float nuevaMuestra)
{
    float muestraFiltrada = nuevaMuestra * bCoeficientePasoAlto[0]
            + xn1Alto * bCoeficientePasoAlto[1] + xn2Alto
            * bCoeficientePasoAlto[2] - aCoeficientePasoAlto[1] * yn1Alto
            - aCoeficientePasoAlto[2] * yn2Alto;
    xn2Alto = xn1Alto;
    xn1Alto = nuevaMuestra;
    yn2Alto = yn1Alto;
    yn1Alto = muestraFiltrada;

    return muestraFiltrada;
}

private void GirarSeñal(float[] resp_buffer)
{
    int i;
    float minimaMuestra = resp_buffer[0];
    float maximaMuestra = resp_buffer[0];
    // Buscamos el valor maximo y minimo del array de entrada
    for (i = 1; i < REF_WIN_LENGTH; i++)
    {
        if (resp_buffer[i] > maximaMuestra)
            maximaMuestra = resp_buffer[i];
        else if (resp_buffer[i] < minimaMuestra)
            minimaMuestra = resp_buffer[i];
    }
    if (maximaMuestra < Math.Abs(minimaMuestra))
    {
        signo = -1;
        // Giramos el array de entrada
        for (i = 0; i < REF_WIN_LENGTH; i++)
        {
            resp_buffer[i] = resp_buffer[i] * signo;
        }
    }

}
```

```
private float CalcularThreshold(int NuevaMuestra)
{

    float muestraFiltrada = filtrarPasoBajoMuestra(NuevaMuestra);
    muestraFiltrada = filtrarPasoAltoMuestra(muestraFiltrada);
    N_samples++;
    if (index_buff < REF_WIN_LENGTH) // Si el buffer no esta lleno
    // insertamos una muestra nueva
    {
        Resp_buffer[index_buff] = muestraFiltrada;
        index_buff++;
    }
    else // Si el buffer esta lleno añadimos la muestra
    {
        shift_buffer(Resp_buffer, muestraFiltrada);
        htr = htrstest(Resp_buffer); // Calculamos indice de heteroestacidad

        if (htr < 0.03)
            bMinimoAlcanzado = true;
        if (htr < htr_old)
        {
            htr_old = htr;
            copy_buffer(Resp_buffer, min_Resp_buffer);
            nMuestraReferenciaFinal = N_samples;
            nMuestraReferenciaInicio = nMuestraReferenciaFinal
                    - REF_WIN_LENGTH;
        }
    }

    if (N_samples == N_SAMPLES_INIT_WINDOW || bMinimoAlcanzado)
    {
        GirarSeñal(min_Resp_buffer);
        threshold = percentil(min_Resp_buffer, PCTIL);
        factorNormalizacion = CalcularFactorNormalizacion(min_Resp_buffer);
    }

    return muestraFiltrada;
}

private float CalcularFactorNormalizacion(float[] resp_buffer2) {
   int tAnterior = 0, tActual = 0, M_av4_indice = 0;
   float periodo = 0;
   int []indCont = new int[4];
   float []vTPeriodo = new float[REF_WIN_LENGTH];
   float difPasosUmbral = 0.0F;
   nPasosUmbralNormalizacion = 0;
   // Detectamos flancos positivos dentro de la ventana
   for (tActual = 1; tActual < REF_WIN_LENGTH; tActual++) {
          if (resp_buffer2[tActual] > threshold
                        && resp_buffer2[tActual - 1] <= threshold) {
                periodo = tActual - tAnterior; // Periodo
                tAnterior = tActual;
                if (nPasosUmbralNormalizacion > 0) {
                       // Filtro de media móvil de 4 Periodos
                       indCont[M_av4_indice] = (int) periodo;
                       M_av4_indice = (M_av4_indice + 1) % 4;
                       if (nPasosUmbralNormalizacion >= 4) {
                              periodo = 0;
                              for (int n = 0; n < 4; n++)
                                     periodo = periodo + indCont[n];

                              periodo = periodo / 4;
                       }
                       vTPeriodo[nPasosUmbralNormalizacion] = periodo;
                }
                nPasosUmbralNormalizacion++;
          }
   }
```

```
            mControladorLogica.EscribirArchivoPasosUmbral(vTPeriodo,
                            nPasosUmbralNormalizacion);
            // Calculamos el factor de normalizacion en la ventana de 40s como la
            // media del valor absoluto de la diferencias de los periodos
            for (int j = 1; j < nPasosUmbralNormalizacion - 1; j++) {
                    difPasosUmbral = difPasosUmbral
                                    + Math.Abs(vTPeriodo[j] - vTPeriodo[j + 1]);
            }
            return difPasosUmbral / (nPasosUmbralNormalizacion - 2);
    }


        private float CalcularDif(int NuevaMuestra)
        {

            int T_period, n, muestrasTranscuridas;
            float muestraFiltrada;
            tnew_index++;
            muestraFiltrada = filtrarPasoBajoMuestra(NuevaMuestra);
            muestraFiltrada = filtrarPasoAltoMuestra(muestraFiltrada) * signo;
            if (muestraFiltrada > threshold && OldSample <= threshold)
            {
                nciclosRespiracion++;
                // Calculate periodo
                T_period = tnew_index - told_index;

                // Calculamos muestrasTranscurridas desde el anterior flanco
                // positivo
                muestrasTranscuridas = tnew_index - told_index;
                told_index = tnew_index;
                Indcont[M_av4_index] = T_period;
                // Filtro de media movil de 4 coeficientes
                M_av4_index = (M_av4_index + 1) % 4;
                // Buffer circular, calculamos el indice para el siguiente valor
                difCalculado = 0;
                // tiene que haber cinco flancos detectados, igual a cuatro
                // periodos, para implementar el filtro de media movil
                if (nciclosRespiracion >= 5)
                {
                    T_period = 0;
                    for (n = 0; n < 4; n++)
                    {

                        T_period = T_period + Indcont[n];
                    }
                    T_period = T_period / 4;

                }
                for (n = 1; n < 11; n++)
                {
                    Indt[n - 1] = Indt[n];
                }
                Indt[10] = T_period;
                // tiene que haber quince flancos, igual a 14 periodos, para
                // calcular las 10 absolutas diferencias
                if (nciclosRespiracion >= 15)
                {
                    for (n = 0; n < 10; n++)// Calculamos el indice como la media de
                    // las últimas 10 absolutas diferencias
                    {
                        difCalculado = difCalculado
                                + Math.Abs(Indt[n] - Indt[n + 1]);
                    }
                    difCalculado = difCalculado / 10;
                    // Aplicamos el factor de normalizacion previamente calculado
                    difCalculado = difCalculado / factorNormalizacion;
                    calcularEstadoAviso(muestrasTranscuridas, difCalculado);

                }
            }
            OldSample = muestraFiltrada;
```

```
    return muestraFiltrada;
}

/**
 * Decide a partir de los valores del ks del último minuto si se ha de dar
 * alarma o no
 *
 * @param t_period
 *            longitud en muestras del periodo respiratorio
 * @param difCalculado
 *            valor del indice ks calculado
 */
private void calcularEstadoAviso(int t_period, float difCalculado)
{
    mListaTiempoKS.Add(new TiempoKS(t_period, difCalculado));
    int sumaTiempo = 0;
    int contadorAvisoDormido = 0;
    int numeroCiclosAviso = mListaTiempoKS.Count;
    // Recorremos la lista de valores de ks para contar el tiempo acumulado
    // y el numero de ciclos que están por encima del umbral de decision
    for (int i = 0; i < numeroCiclosAviso; i++)
    {
        sumaTiempo += mListaTiempoKS[i].getTiempo();
        if (mListaTiempoKS[i].getKs() > umbralFase1)
            contadorAvisoDormido++;
    }
    if (sumaTiempo >= 300 * 40)
    { // Si el tiempo acumulado es mayor de un
        // minuto decidimos el estado de alarma
        // por mayoría
        bEstadoAlarma = (contadorAvisoDormido > numeroCiclosAviso / 2);
    }
    mControladorLogica.EscribirArchivoDecisionAlarma(mListaTiempoKS,
    sumaTiempo, bEstadoAlarma);
    while (sumaTiempo >= 300 * 40)
    {// Eliminamos los ultimos elementos para
        // deslizar la ventana de un minuto
        sumaTiempo -= mListaTiempoKS[0].getTiempo();
        mListaTiempoKS.RemoveAt(0);
    }

}

/**
 *
 * @param muestra valor de la muestra recibida
 * @return valor de la muestra filtrada
 */
public float ProcesarMuestra(int muestra)
{

    float muestraFiltrada;
    if (N_samples < RETARDO)
    { // Fase de inicilizacion de los filtros
        muestraFiltrada = iniciarFiltrado(muestra);
        N_samples++;
    }
    else if (N_samples < (N_SAMPLES_INIT_WINDOW + RETARDO)
            && !bMinimoAlcanzado)
    {// Fase de calculo de los parametros del
        // algoritmo
        muestraFiltrada = CalcularThreshold(muestra);

    }
    else
        muestraFiltrada = CalcularDif(muestra);// Fase de calculo de Ks

    return muestraFiltrada;
}
```

```
            private float iniciarFiltrado(int muestra)
            {
                float muestraFiltrada = filtrarPasoBajoMuestra(muestra);
                muestraFiltrada = filtrarPasoAltoMuestra(muestraFiltrada);
                return muestraFiltrada;
            }

            public int getNumeroCiclos()
            {
                return nciclosRespiracion;
            }

            public float getThreshold()
            {
                return threshold;
            }

            public float getFactorNormalizacion()
            {

                return factorNormalizacion;
            }

            public bool estadoIncializacion()
            {

                return tnew_index == 0;
            }

            public int getTiempoReferenciaInicial()
            {

                return nMuestraReferenciaInicio;
            }

            public int getTiempoReferenciaFinal()
            {

                return nMuestraReferenciaFinal;
            }

            public float getHtrIndex()
            {

                return htr;
            }

            public int getPasosPorUmbral()
            {

                return nPasosUmbralNormalizacion;
            }

            public bool minimoAlcanzado()
            {

                return bMinimoAlcanzado;
            }
        }
    }
}
```

Contenido del fichero ArchivadorResultados.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
namespace Ks_HMI
{
    namespace Logica
    {
        class ArchivadorResultados
        {
            private string mydocpath;
            private bool tieneSD;
            private String pathArchivoInicializacion;
            private String pathArchivoUmbral;
            private String pathArchivoDecision;
            private String pathArchivoVariables;

            public ArchivadorResultados()
            {
                mydocpath = null;
                tieneSD = compobarAlmacenamientoExterno();
                getPath();
            }

            private bool compobarAlmacenamientoExterno()
            {
                //storage cards are directories with the temporary attribute
                System.IO.FileAttributes attrStorageCard =
System.IO.FileAttributes.Directory | System.IO.FileAttributes.Temporary;
                DirectoryInfo rootDir = new DirectoryInfo(@"\");
                foreach (DirectoryInfo di in rootDir.GetDirectories())
                {
                    //if directory and temporary
                    if ((di.Attributes & attrStorageCard) == attrStorageCard)
                    {
                        //add to collection of storage cards
                        if (di.Name.Contains("Card") || di.Name.Contains("Tarj"))
                        {
                            mydocpath = di.FullName;
                            return true;
                        }
                    }
                }
                return false;
            }

            private void getPath()
            {
                string date = DateTime.Now.ToString("yyyy_MM_dd_HH_mm");
                string filenameInicializacion = "\\TestKS_HMI_Inicializacion " + date +
".txt";
                string filenameVariables = "\\TestKS_HMI_Variables " + date + ".txt";
                string filenamePasosUmbral = "\\TestKS_HMI_Pasos_Umbral " + date +
".txt";
                string filenameDecisionAlarma = "\\TestKS_HMI_Decision_Alarma " + date +
".txt";

                if (!tieneSD) mydocpath =
Environment.GetFolderPath(Environment.SpecialFolder.Personal);
                mydocpath = Path.Combine(mydocpath, "KS_HMI");
                Directory.CreateDirectory(mydocpath);

                pathArchivoInicializacion = mydocpath + filenameInicializacion;
                pathArchivoVariables = mydocpath + filenameVariables;
                pathArchivoUmbral = mydocpath + filenamePasosUmbral;
                pathArchivoDecision = mydocpath + filenameDecisionAlarma;
```

```
                if (File.Exists(pathArchivoInicializacion))
                {
                    System.IO.File.Delete(pathArchivoInicializacion);
                }
                if (File.Exists(pathArchivoVariables))
                {
                    System.IO.File.Delete(pathArchivoVariables);
                }
                if (File.Exists(pathArchivoInicializacion))
                {
                    System.IO.File.Delete(pathArchivoInicializacion);
                }
                if (File.Exists(pathArchivoUmbral))
                {
                    System.IO.File.Delete(pathArchivoUmbral);
                }
                if (File.Exists(pathArchivoDecision))
                {
                    System.IO.File.Delete(pathArchivoDecision);
                }
                // Create the new, empty data file.
            }

            public void EscribirArchivoVariables(int[] lNumeroMuestra, int[]
lValoresMuestra,
                float[] lValoresFiltrados, float[] lValoresIndice, int
nValoresGuardados,
                int[] lNumeroCiclos)
            {
                StreamWriter w = File.AppendText(pathArchivoVariables);
                int nValoresArchivados = 0;
                string s;
                while (nValoresArchivados < nValoresGuardados)
                {
                    s = lNumeroMuestra[nValoresArchivados].ToString() + " " +
                        lValoresMuestra[nValoresArchivados].ToString() + " " +
                        lValoresFiltrados[nValoresArchivados].ToString() + " " +
                        lValoresIndice[nValoresArchivados].ToString() + " " +
                        lNumeroCiclos[nValoresArchivados].ToString() + "\n";
                    w.Write(s);
                    nValoresArchivados++;
                }
                w.Flush();
                w.Close();
                w.Dispose();
            }
            public void EscribirArchivoIncializacion(int[] lNumeroMuestra, int[]
lValoresMuestra,
                float[] lValoresFiltrados, int nValoresGuardados, float[]
lValorThreshold,
                float[] lFactorNormalizacion, int[] lTiempoReferenciaInicial,
                int[] lTiempoReferenciaFinal, int[] lPasosPorUmbral, float[] lhtr)
            {
                StreamWriter w = File.AppendText(pathArchivoInicializacion);
                int nValoresArchivados = 0;
                string s;
                while (nValoresArchivados < nValoresGuardados)
                {
                    s = lNumeroMuestra[nValoresArchivados].ToString() + " " +
                        lValoresMuestra[nValoresArchivados].ToString() + " " +
                        lValoresFiltrados[nValoresArchivados].ToString() + " " +
                        lValorThreshold[nValoresArchivados].ToString() + " " +
                        lFactorNormalizacion[nValoresArchivados].ToString() + " " +
                        lTiempoReferenciaInicial[nValoresArchivados].ToString() + " " +
                        lTiempoReferenciaFinal[nValoresArchivados].ToString() + " " +
                        lPasosPorUmbral[nValoresArchivados].ToString() + " " +
                        lhtr[nValoresArchivados].ToString() + "\n";
                    w.Write(s);
                    nValoresArchivados++;
                }
                w.Flush();
                w.Close();
```

```
            w.Dispose();
        }

        public void EscribirArchivoPasosUmbral(float[] vTPeriodo,
        int nPasosUmbralNormalizacion)
        {
            StreamWriter w = File.AppendText(pathArchivoUmbral);

            string s;

            for (int j = 1; j < nPasosUmbralNormalizacion; j++)
            {
                s = vTPeriodo[j].ToString() + "\n";
                w.Write(s);
            }


            w.Flush();
            w.Close();
            w.Dispose();
        }
        public void EscribirArchivoDecisionAlarma(
        List<TiempoKS> mListaTiempoKS,
        int sumaTiempo, bool bEstadoAlarma)
        {
            StreamWriter w = File.AppendText(pathArchivoDecision);

            string s;

            for (int j = 0; j < mListaTiempoKS.Count; j++)
            {
                s = mListaTiempoKS[j].getTiempo().ToString() + " "
                        + mListaTiempoKS[j].getKs().ToString()
                        + "\n";
                w.Write(s);
            }
            s = sumaTiempo.ToString() + " "
                    + bEstadoAlarma.ToString() + "\n";
            w.Write(s);

            w.Flush();
            w.Close();
            w.Dispose();
        }



    }
}
}
```

Contenido del fichero AvisadorSomnolencia.cs

```
using System;
using System.Text;
using System.Media;
using System.Threading;
using Microsoft.WindowsMobile.Status;
using System.IO;
using Microsoft.WindowsMobile.PocketOutlook;
using System.Runtime.InteropServices;
namespace Ks_HMI
{

    namespace Logica
    {

        class AvisadorSomnolencia
        {
            string numTlf;
            SoundPlayer player;
            bool bTelefonoActivo;
            bool AvisoOn;
            public AvisadorSomnolencia(string number)
            {
                Stream audioStream = new MemoryStream(Properties.Resources.beep_6);
                player = new SoundPlayer(audioStream);
                player.Load();
                AvisoOn = false;
                numTlf = number;
                bTelefonoActivo = SystemState.PhoneRadioPresent &&
                    SystemState.PhoneRadioOff && !SystemState.PhoneNoSim;
            }
            private void playAviso()
            {
                if (!AvisoOn)
                {
                    AvisoOn = true;
                    player.PlayLooping();
                }

            }

            public void PararAviso()
            {
                if (AvisoOn)
                {
                    player.Stop();
                    player.LoadAsync();
                    AvisoOn = false;
                }
            }

            private void EnviarSMS()
            {
                SmsMessage sms = new SmsMessage(numTlf,
                     "Te estás durmiendo");
                if (!SystemState.PhoneRadioOff && !SystemState.PhoneNoSim) sms.Send();

                MessagingApplication.DisplayComposeForm(sms);
            }

            public void Avisar()
            {
                this.playAviso();
                if (bTelefonoActivo)
                {
                    this.EnviarSMS();
                }
            }
        }
    }
}
```

Contenido del fichero AnalizadorDispositivo.cs

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using Microsoft.WindowsMobile.Status;
using System.Collections;
using Microsoft.WindowsCE.Forms;
using System.Runtime.InteropServices;
using Microsoft.Win32;
using Comunicacion;

namespace Gestion
{
    class AnalizadorDispositivo
    {
        private ArrayList stateList;
       //private VistaPrincipal vst;
        /*http://msdn.microsoft.com/en-us/library/aa446573.aspx*/
        [DllImport("coredll.dll")]
        private static extern int waveOutGetVolume(IntPtr hwo, ref int dwVolume);
        public enum Volumes : int
        {
            OFF = 0,
            LOW = 858993459,
            NORMAL = 1717986918,
            MEDIUM = -1717986919,
            HIGH = -858993460,
            VERY_HIGH = -1
        }
        public AnalizadorDispositivo()
        {

        }

        public void ComprobarEstadoSistema(ref String s)
        {
            s = null;
            if (SystemState.PhoneRadioOff && SystemState.PhoneRadioPresent)
            {
                MobileRadio.SetDeviceState(MobileRadio.RADIODEVTYPE.PHONE,
MobileRadio.RADIODEVSTATE.ON);
            }
            // if (!(bool) SystemState.GetValue(SystemProperty.BluetoothStatePowerOn)) {
            if (!(bool)SystemState.BluetoothStatePowerOn)
            {
                MobileRadio.SetDeviceState(MobileRadio.RADIODEVTYPE.BLUETOOTH,
MobileRadio.RADIODEVSTATE.ON);
                ///s = "Bluetooth Apagado. Para encender ir a Inicio > Configuración " +
                //  "> Ficha Conexiones > Bluetooth. \n";
                //bEstadoCorrecto = false;
            }
            int before = 0;
            waveOutGetVolume(IntPtr.Zero, ref before);
            if (before == (int)Volumes.OFF)
            {
                s += "El aviso acústico no funcionará hasta que se aumente el volumen "
+
                    "en la configuración de Sonidos y notificaciones.\n";
            }
            else if (!SystemState.PowerBatteryState.Equals(BatteryState.Charging) &&
(SystemState.PowerBatteryStrength.Equals(BatteryLevel.VeryLow) ||
SystemState.PowerBatteryStrength.Equals(BatteryLevel.Low)))
            {
                s = "Se recomienda conectar el dispositivo a la corriente \n" +
                    "El nivel de batería es: " +
SystemState.PowerBatteryStrength.ToString();
            }
```

```
        }
        // Look in the registry to see what the shortest timeout
        // period is. Note that Zero is a special value with respect
        // to timeouts. It indicates that a timeout will not occur.
        // As long as SystemIdleTimeerReset is called on intervals
        // that are shorter than the smallest non-zero timeout value
        // then the device will not sleep from idleness. This does
        // not prevent the device from sleeping due to the power
        // button being pressed.
        public int ShortestTimeoutInterval()
        {
            int retVal = 1000;
            RegistryKey key =

Registry.LocalMachine.OpenSubKey(@"\SYSTEM\CurrentControlSet\Control\Power");
            object oBatteryTimeout = key.GetValue("BattPowerOff");
            object oACTimeOut = key.GetValue("ExtPowerOff");
            object oScreenPowerOff = key.GetValue("ScreenPowerOff");
            if (oBatteryTimeout is int)
            {
                int v = (int)oBatteryTimeout;
                if (v > 0)
                    retVal = Math.Min(retVal, v);
            }
            if (oACTimeOut is int)
            {
                int v = (int)oACTimeOut;
                if (v > 0)
                    retVal = Math.Min(retVal, v);
            }
            if (oScreenPowerOff is int)
            {
                int v = (int)oScreenPowerOff;
                if (v > 0)
                    retVal = Math.Min(retVal, v);
            }
            return retVal * 500;
        }
    }
}
```

Contenido del fichero MobileRadio.cs

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
namespace Comunicacion
{
    class MobileRadio
    {// Cómo encender el teléfono en Windows Mobile con C#
        [DllImport("ossvcs.dll", EntryPoint = "#276", CharSet = CharSet.Unicode)]
        private static extern uint GetWirelessDevice(ref IntPtr pDevice, int pDevVal);

        [DllImport("ossvcs.dll", EntryPoint = "#273", CharSet = CharSet.Unicode)]
        private static extern uint ChangeRadioState(ref RDD pDevice, int dwState, int
saveAction);

        [DllImport("ossvcs.dll", EntryPoint = "#280", CharSet = CharSet.Unicode)]
        private static extern uint FreeDeviceList(IntPtr pDevice);

        [StructLayout(LayoutKind.Auto)]
        public struct RADIODEVSTATE
        {
            public const int ON = 1;
            public const int OFF = 0;
        }

        [StructLayout(LayoutKind.Auto, CharSet = CharSet.Unicode)]
        public struct RADIODEVTYPE
        {
            public const int PHONE = 2;
            public const int BLUETOOTH = 3;
        }

        [StructLayout(LayoutKind.Auto, CharSet = CharSet.Unicode)]
        struct SAVEACTION
        {
            public const int DONT_SAVE = 0;
            public const int PRE_SAVE = 1;
            public const int POST_SAVE = 2;
        }

        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
        struct RDD
        {
            [MarshalAs(UnmanagedType.LPTStr)]
            public string pszDeviceName;

            [MarshalAs(UnmanagedType.LPTStr)]
            public string pszDisplayName;

            public uint dwState;
            public uint dwDesired;
            public int DeviceType;
            public IntPtr pNext;
        }

        public static bool SetDeviceState(int dwDevice, int dwState)
        {
            IntPtr pDevice = new IntPtr(0);
            RDD device;
            uint result;

            //Get the first wireless device
            result = GetWirelessDevice(ref pDevice, 0);
            if (result != 0)
                return false;
```

```
            //If the first device has been found
            if (pDevice != null)
            {
                //While we're still looking at wireless devices
                while (pDevice != IntPtr.Zero)
                {
                    //Marshall the pointer into a C# structure
                    device =
(RDD)System.Runtime.InteropServices.Marshal.PtrToStructure(pDevice, typeof(RDD));

                    //If this device is the device we're looking for
                    if (device.DeviceType == dwDevice)
                    {
                        //Change the state of the radio
                        result = ChangeRadioState(ref device, dwState,
SAVEACTION.PRE_SAVE);
                    }

                    //Set the pointer to the next device in the linked list
                    pDevice = device.pNext;
                }

                //Free the list of devices
                FreeDeviceList(pDevice);
            }

            //Turning off radios doesn't correctly report the status, so return true
anyway.
            if (result == 0 || dwState == RADIODEVSTATE.OFF)
                return true;

            return false;
        }
    }
}
```

Contenido del fichero Program.cs

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Threading;
using Ks_HMI.Vista;


namespace Ks_HMI
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [MTAThread]
        static void Main()
        {
            Thread.CurrentThread.Priority = ThreadPriority.AboveNormal;
            Application.Run(new VistaInicial());
        }
    }
}
```

8. Bibliografía

[1] Rodríguez Ibáñez, N. Sistema multivariable de detección de somnolencia en conductores mediante el análisis de señales biomédicas. Proyecto de Tesis Doctoral. Universidad Politécnica de Catalunya, 2011.

[2]Microsoft Network Development http://msdn.microsoft.com/es-es/library/k1s94fta%28v=VS.90%29.aspx

[3] Página oficial del Open Handset Alliance resumen sobre android http://www.openhandsetalliance.com/android_overview.html

[4] Guía para desarrolladores de Android http://developer.android.com/guide/basics/what-is-android.html

[5] González Fernández,E. *Dispositivos móviles, Iphone*. Tesis de Máster, Universidad de Oviedo 2008.

[6] Web oficial para desarrolladores de Blackberry http://es.blackberry.com/developers/javaappdev/

[7] Piñera Sánchez, C.J. *Desarrollo de herramientas didácticas para el aprendizaje de Symbian OS*. Proyecto Final de Carrera, Universidad Politécnica de Cartagena, 2007.

[8] Windows Phone Development http://msdn.microsoft.com/en-us/library/ff402535%28v=VS.92%29.aspx

[9] Especificación del perfil puerto serie (SPP). https://www.bluetooth.org/Building/HowTechnologyWorks/ProfilesAndProtocols/SPP.htm

[10] Artículo de buenas prácticas de desarrollo en Android sobre diseño para respuesta. Web oficial de Android para desarrolladores http://developer.android.com/guide/practices/design/responsiveness.html

[11] García Eliseo, L.A. *Introducción a Windows Forms.* Notas de clase, Universidad Autónoma de Tamaulipas.

[12] Artículo Supporting Multiple Screens. Web oficial de Android para desarrolladores http://developer.android.com/guide/practices/screens_support.html#range

[13] Dr. Victor Matos, *Android Application's Life Cycle*. Notas de Clase, Cleveland State University 2010.

[14]Articulo sobre ResourceManager Visual Studio. Centro de desarrollo de Microsoft. http://msdn.microsoft.com/en-us/library/7k989cfy%28v=vs.80%29.aspx

[15] Application Resources SDK Android developers. Web oficial de Android para desarrolladores. http://developer.android.com/guide/topics/resources/index.html

[16] Pallás Areny, R. Sensores y acondicionadores de señal. Marcombo, 2003.

[17] Artículo sobre buenas prácticas de diseño para el rendimiento. Web oficial de Android para desarrolladores http://developer.android.com/guide/practices/design/performance.html#object_creation

[18] Página web contiene sonidos con licencia Creative Commons http://www.soundjay.com/beep-sounds-1.html

[19] Artículo sobre como encender el teléfono en Windows Mobile con C#. http://www.hinojosachapel.com/2010/02/como-encender-el-telefono-en-windows-mobile-con-c.aspx

[20] Artículo para prevenir el apagado automático. Centro de desarrollo de Microsoft. http://msdn.microsoft.com/en-us/library/bb158564.aspx