Universitat Oberta de Catalunya

uoc.edu

Control de acceso NFC mediante autorización biométrica con smartphone

Antonio Ortega Pérez
Máster Universitario de Ingeniería de Telecomunicación
Sistemas de Comunicación

Consultor: Raúl Parada Medina
Profesor: Carlos Monzo Sánchez

10/06/2018

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Control de acceso NFC mediante autorización biométrica con smartphone*
Nombre del autor:	*Antonio Ortega Pérez*
Nombre del consultor/a:	*Raúl Parada Medina*
Nombre del PRA:	*Carlos Monzo Sánchez*
Fecha de entrega (mm/aaaa):	06/2018
Titulación:	*Máster Universitario de Ingeniería de Telecomunicación*
Área del Trabajo Final:	*Sistemas de Comunicación*
Idioma del trabajo:	*Castellano*
Palabras clave	*NFC, biometría, autorización, Arduino, smartphone, Android, smartwatch*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Los sistemas de control de acceso se utilizan para permitir el acceso a lugares, información, o sistemas informáticos. Éstos controles se basan en tres procesos: autenticación, autorización, y trazabilidad.

En dichos procesos se pueden utilizar medidas biométricas (las relacionadas con características humanas), orientadas principalmente a la autenticación (identificación) de las personas. En este Trabajo se pretende plantear un sistema de acceso físico con autorización bajo criterios biométricos, dejando de lado la autenticación.

Para ello se plantea hacer uso de un *smartphone* Android con tecnología NFC (*Near-Field Communication*) de comunicación de corta distancia, y un sistema con microcontrolador que gobierne el elemento que permite el acceso. El *smartphone* captará la medida biométrica, y enviará al microcontrolador la medida o el resultado de la evaluación de autorización mediante NFC.

El desarrollo de ese sistema permitirá alcanzar los objetivos planteados en cuanto a estudio de la tecnología NFC y su uso para transmitir información entre teléfonos inteligentes y hardware basado en microcontroladores, así como explorar diferentes sensores biométricos que se puedan utilizar con un *smartphone*.

Abstract (in English, 250 words or less):

Access control systems are used to allow access to premises, information, or computer systems. These control systems are based on three processes: authentication, authorization, and traceability.

In these processes, biometric measurements (those related to human characteristics) may be used, mainly oriented to the authentication (identification) of people. This Thesis intention is to propose a physical access system using biometric criteria for authorization, leaving aside the authentication.

For that purpose, the use of an NFC-capable (Near-Field Communication) Android smartphone is presented, along with a microcontroller-based system governing the component which physically allows the access. The smartphone will acquire the biometric measurement and will send the value or assessment to the microcontroller using NFC.

The development of this system will allow to reach the objectives regarding the study of NFC technology and its use for information transmission between smartphones and microcontroller boards, as well as to explore different biometric sensors that can be used with a smartphone.

Índice

1. Introducción 1
1.1 Contexto y justificación del Trabajo 1
1.2 Objetivos del Trabajo 2
1.3 Enfoque y método seguido 2
1.4 Planificación del Trabajo 3
1.5 Breve sumario de productos obtenidos 3
1.6 Breve descripción de los otros capítulos de la memoria 4
2. Estado del arte 5
2.1 Tecnología NFC para la comunicación entre dispositivos 5
2.2 Sensores biométricos y fisiológicos para *smartphone* 5
2.3 Sistemas de control de acceso 8
3. Descripción del sistema propuesto 9
3.1 Descripción general de la arquitectura 9
3.2 Elección del sensor para el prototipo 10
3.3 Elección del microcontrolador, lector NFC y accionador 11
3.4 Elección del *smartphone* 13
4. Descripción de aplicaciones Android 14
4.1 Android y desarrollo de aplicaciones 14
4.2 Ciclo de vida de las Actividades 16
4.3 Gestores de eventos y lectura de sensores 17
4.4 Comunicación con Wear OS – Data Layer 17
4.5 Comunicación NFC – Host-based Card Emulation 18
4.6 Arquitectura de las aplicaciones Android del sistema a implementar 20
5. Descripción de NFC y el código del lector 23
5.1 Estándares NFC para tarjetas inteligentes 23
5.2 Protocolo de emulación de tarjetas ISO/IEC 7816-4 y Android 24
5.3 Arduino, I2C, y Módulo PN532 24
6. Prueba práctica del sistema 28
6.1 Escenario 28
6.2 Pruebas y resultados 29
6.3 Coste del sistema 32
7. Conclusiones 33
8. Glosario 35
9. Bibliografía 37
10. Anexos 38

Índice de figuras

Figura 1. Diagrama de Gantt con la planificación del Trabajo 3
Figura 2. Envíos mundiales de *smartphones* con NFC 5
Figura 3. *Smartphone* VIVO con lector de huella dactilar.................................. 6
Figura 4. Sensor de pulsioximetría del Samsung Galaxy S6............................. 6
Figura 5. SMUFS-BT PRO (escáner huella dactilar) 7
Figura 6. BACtrack Mobile Pro (alcoholímetro) 7
Figura 7. MindWave Mobile (EEG)... 7
Figura 8. NFC Forum Product Showcase – My Lock. 8
Figura 9. Arquitectura a alto nivel... 9
Figura 10. Posible utilidad práctica del sistema propuesto................................ 9
Figura 11. Sensor de gases MQ-3 con salida analógica 10
Figura 12. Sensor de ritmo cardíaco del Ticwatch E 11
Figura 13. Placa de desarrollo Arduino UNO 12
Figura 14. Módulo PN532 NFC 12
Figura 15. Esquemático de conexiones entre Arduino y módulo NFC 13
Figura 16. Ciclo de vida de una Actividad 16
Figura 17. Pila de protocolos HCE de Android 19
Figura 18. Arquitectura de las aplicaciones Android del sistema 20
Figura 19. Mensaje en pantalla si la precisión del sensor no es ALTA 21
Figura 20. Interfaz gráfica de la aplicación del *smartphone* 22
Figura 21. Estructura de un paquete de comando PN532 25
Figura 22. Diagrama de flujo del programa Arduino.. 26
Figura 23. Respuesta al comando `InListPassiveTarget` con tarjeta tipo A..... 26
Figura 24. Estructura del comando `SELECT` y su respuesta 27
Figura 25. Arquitectura del prototipo 28
Figura 26. Escenario completo... 28
Figura 27. Mensajes en consola del PC tras el arranque de Arduino............... 29
Figura 28. Pantallas mostradas al iniciar las aplicaciones 29
Figura 29. Rangos de frecuencia cardíaca.. 30
Figura 30. Mensajes tras detección y valor superior al umbral......................... 30
Figura 31. Mensajes tras detección y valor caducado...................................... 31
Figura 32. Mensajes con valor inferior al umbral.. 31
Figura 33. Mensajes al acercar un *smartphone* sin la aplicación 31
Figura 34. Mensajes al acercar el *smartphone* con error 31

1. Introducción

En este capítulo se expone la justificación que motiva la realización del presente Trabajo, los objetivos que se persiguen, el enfoque planteado para su ejecución, así como la planificación prevista y un resumen del producto obtenido.

1.1 Contexto y justificación del Trabajo

En los campos de la seguridad física y de la seguridad de la información se utilizan sistemas de control de accesos para permitir o no el acceso a lugares, a información, o a sistemas informáticos. Estos sistemas de control de acceso, en particular los de tipo acceso físico, abarcan desde una sencilla cerradura hasta modernos y complejos sistemas electrónicos basados en criptografía [1]. Basta con acercarse a cualquier oficina para observar cómo existe algún tipo de control de acceso electrónico, por ejemplo, mediante una tarjeta.

En un sentido general, los sistemas de control de acceso se basan en tres procesos: autenticación (confirmación de identidad), autorización (verificación de derechos), y trazabilidad/contabilización. Estos procesos se abrevian en inglés por sus siglas AAA (*Authentication, Authorization and Accounting*) [2].

La biometría hace referencia a las medidas y cálculos relacionados con características humanas, sean características físicas (como la huella dactilar) o conductuales (como la escritura). Esto tipo de medidas pueden ser utilizadas en los citados procesos AAA, sobre todo cuando se trata de acceso a recursos especialmente sensibles.

Si bien el foco principal de la biometría en estos sistemas se sitúa en la autenticación (identificación) de las personas, con este Trabajo se pretende plantear un sistema de autorización bajo criterios biométricos sencillos.

Los *smartphones*, o teléfonos inteligentes, se han convertido en un dispositivo indispensable para las necesidades de comunicación personales de la sociedad, de tal manera que se puede afirmar que cada individuo dispone de un *smartphone* y que lo lleva consigo a todos lados. Aprovechando dicha situación, en la actualidad incluso las empresas están comenzando a aplicar políticas de tipo BYOD (*bring your own device*) que consisten en que los empleados lleven sus propios dispositivos a su lugar de trabajo para el acceso a los recursos [3].

NFC (*Near-Field Communication*) es un conjunto de protocolos de comunicación que permiten a dos dispositivos electrónicos establecer una comunicación cuando se encuentran en un rango de unos 4 cm entre ellos. El uso de esta tecnología se está extendiendo de tal manera que en la actualidad los *smartphones* la incorporan y se puede utilizar para realizar pagos, emular tarjetas de proximidad para sistemas de control de acceso, o simplemente intercambiar pequeños fragmentos de información.

Por todo ello, con este Trabajo se pretenden explorar los campos descritos y realizar un sistema de autorización de acceso, donde la comunicación se realice mediante NFC entre un *smartphone* Android y el elemento de control microcontrolador, y la autorización se base en algún umbral de un parámetro biométrico medido por el *smartphone*, como podría ser la concentración de alcohol en sangre, la frecuencia cardíaca, o el estado de concentración del individuo.

1.2 Objetivos del Trabajo

Con este Trabajo se pretende alcanzar los siguientes objetivos:

· Familiarizarse con la tecnología NFC, conociendo los estándares en los que se basa, los distintos tipos de funcionamiento, y sus diferencias con otras tecnologías de comunicación similares.

· Ser capaz de transmitir información mediante tecnología NFC entre una aplicación Android en un *smartphone* y un microcontrolador con un hardware especializado.

· Explorar diferentes sistemas de adquisición de medidas biométricas mediante un *smartphone* Android.

· Diseñar y crear un prototipo de sistema de autorización basado en alguna medida biométrica, utilizando los elementos anteriormente descritos: tecnología NFC para la comunicación, *smartphone* y aplicación Android, microcontrolador y hardware NFC, sensor biométrico.

1.3 Enfoque y método seguido

La estrategia consistirá, tras un análisis del estado del arte de las diferentes tecnologías, en estudiar el funcionamiento de éstas, ya que este conocimiento será necesario para el desarrollo de un prototipo.

Tras su estudio, se desarrollarán pequeños programas en C (ejecutable en un microcontrolador como el de la plataforma Arduino) y Java (ejecutable en Android) que permitan poner en práctica lo aprendido, y conduzcan hacia poder implementar el hardware y el software del sistema prototipo final.

Con todo ello se podrán obtener unas conclusiones sobre el Trabajo en cuanto a idoneidad de las tecnologías escogidas, utilidad del producto desarrollado, y se sentarán las bases de posibles futuros trabajos o líneas de investigación.

1.4 Planificación del Trabajo

El trabajo se estructura en 5 entregables de evaluación continua, que se corresponden a diferentes subtareas necesarias para su realización.

En la figura 1 se observa un diagrama de Gantt con la planificación a alto nivel:

			FEBRERO			MARZO				ABRIL				MAYO				JUNIO				
				S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24	S25	S26
Actividad	**Inicio**	**Fin**	**Duración**																			
PEC1	22/02/2018	07/03/2018	2 sem.																			
Búsqueda de tema			1 sem.																			
Redacción objetivos			1 sem.																			
Entrega PEC1		07/03/2018				X																
PEC2	08/03/2018	19/03/2018	2 sem.																			
Búsqueda de información			2 sem.																			
Redacción del estado del arte			2 sem.																			
Entrega PEC2		19/03/2018					X															
PEC3	20/03/2018	20/05/2018	9 sem.																			
Planteamiento del sistema			1 sem.																			
Búsqueda del sensor adecuado			1 sem.																			
Lectura de estándares NFC			3 sem.																			
Búsqueda de módulo NFC			1 sem.																			
Compra del wearable y módulo			2 sem.																			
Familiarización con Wear OS			3 sem.																			
Familiarización con módulo NFC			2 sem.																			
Desarrollos Android			6 sem.																			
Desarrollo Arduino			4 sem.																			
Prueba completa			2 sem.																			
Redacción PEC3			9 sem.																			
Entrega PEC3		20/05/2018														X						
PEC4	21/05/2018	10/06/2018	4 sem.																			
Revisión correcciones PEC3			1 sem.																			
Redacción memoria final			3 sem.																			
Entrega PEC4		10/06/2018																	X			
PEC5	11/06/2018	22/06/2018	2 sem.																			
Preparación presentación			1 sem.																			
Grabación vídeos/audio			1 sem.																			
Entrega PEC5		22/06/2018																			X	

Figura 1. Diagrama de Gantt con la planificación del Trabajo.

Las subtareas correspondientes a cada entregable vendrán indicadas a partir de los enunciados de las diferentes PECs.

Para la PEC2, la idea es realizar una recopilación del estado del arte de las diferentes tecnologías, mientras que en la PEC3 se desarrollará la mayor parte de carga de trabajo. Finalmente, la PEC4 será la entrega de la memoria final, así como en la PEC5 se realizará la presentación del Trabajo.

1.5 Breve sumario de productos obtenidos

Con la ejecución de este Trabajo se ha desarrollado un prototipo de sistema de control de acceso mediante NFC, donde el lector conectado al actuador (un diodo LED por simplicidad) se basa en la plataforma Arduino UNO de prototipado con microcontrolador y en un circuito integrado PN532 que implementa los protocolos NFC.

La información que recibe el lector es una medida de frecuencia cardíaca (pulsaciones por minuto, tres dígitos numéricos) presentada por un *smartphone* Android, y que éste recibe de un *smartwatch*/*wearable* que integra un sensor óptico para la adquisición de dicha medida. El microcontrolador toma la decisión de permitir o no el acceso si la medida supera un cierto umbral previamente configurado en el código fuente del lector.

1.6 Breve descripción de los otros capítulos de la memoria

En el capítulo 2 se expone un análisis del estado del arte en el que se investiga la penetración de las tecnologías inalámbricas Bluetooth y NFC en los *smartphones*. También se realiza una búsqueda breve de diferentes sensores biométricos que pueden trabajar con ellos, bien integrados o bien como periférico externo. Finalmente se comenta el uso de NFC para el control de acceso, buscando productos ya existentes.

En el capítulo 3 se muestra la propuesta presentada en este Trabajo, donde se analizará su arquitectura, y se justificará la elección de los distintos elementos que la componen.

En el capítulo 4 se describen los conceptos básicos de las aplicaciones Android, para poder presentar el diseño propuesto para las aplicaciones que ejecutarán el *smartphone* y el *wearable* (elemento con el sensor escogido).

En el capítulo 5 se describen en la medida de lo posible los estándares de NFC y tarjetas inteligentes que se seguirán para el diseño de la comunicación entre smartphone y lector NFC basado en microcontrolador, así como la programación de este microcontrolador bajo la plataforma Arduino.

En el capítulo 6 se mostrarán las pruebas completas realizadas para demostrar el correcto funcionamiento del sistema implementado.

Finalmente, en el capítulo 7 se exponen las conclusiones del Trabajo, junto con un análisis crítico y futuras líneas de trabajo derivadas de él.

2. Estado del arte

En este capítulo se analiza el uso actual de tecnologías inalámbricas para la comunicación, la existencia de sensores biométricos y fisiológicos para su uso con *smartphones*, y el uso de NFC en sistemas de control de accesos.

2.1 Tecnología NFC para la comunicación entre dispositivos

A la hora de plantear la transmisión de información a bajas tasas de transferencia entre un dispositivo tal como un *smartphone* y otros elementos electrónicos, se pueden valorar diferentes tecnologías inalámbricas, encontrándose entre ellas los infrarrojos (protocolos IrDA), Bluetooth, y NFC.

Estudios comparativos entre las citadas tecnologías que se han realizado, concluyen que la tecnología NFC es la más adecuada a la hora de implantarla en sistemas empotrados y sistemas de autenticación, gracias a su bajo coste y bajo consumo, y cuando el rango de transmisión se permite que sea de escasos centímetros [4].

En cuanto a penetración, Bluetooth es la tecnología más extendida, pero la cantidad de *smartphones* con NFC incorporado no para de crecer, estimándose una fabricación superior a un 80% para el año 2019, como se puede ver en la predicción de la figura 2.

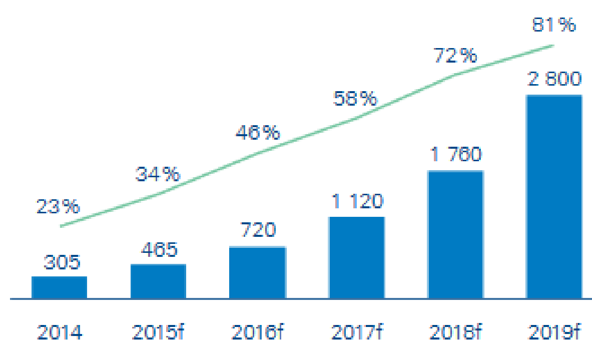

Figura 2. Envíos mundiales de smartphones con NFC en millones (penetración en %). 'Mobile payment space taking hold across the globe'. Consultancy.uk (2015)

Si se trata de transferir información entre el *smartphone* y otros dispositivos a distancias de decenas de centímetros como en las redes de área personal (PAN), o a tasas de transferencia más elevadas, Bluetooth sería la tecnología más adecuada.

2.2 Sensores biométricos y fisiológicos para *smartphone*

Actualmente existen diversos sensores comerciales que pueden medir parámetros biométricos o fisiológicos y que están ideados para enviar las medidas a un *smartphone* mediante tecnología Bluetooth.

Asimismo, en el caso de la medida biométrica de la huella dactilar, o en el caso de la medida fisiológica de la frecuencia cardíaca, existen *smartphones* que ya integran sensores para tales mediciones.

Por ejemplo, se ha presentado recientemente en la feria CES (*Consumer Electronics Show*) de las Vegas el *smartphone* Vivo que incluye un sensor de huella dactilar integrado en la propia pantalla del *smartphone* [5], que se puede observar en la figura 3.

Figura 3. Smartphone VIVO con lector de huella dactilar integrado en pantalla.
'CES 2018: Vivo smartphone fingerprint scanner'. Moneycontrol.com (2018)

Los *smartphones* de Samsung como el Galaxy S6 incorporan en la parte trasera un sensor de saturación de oxígeno (SpO_2) y frecuencia cardíaca, que realmente consiste en un LED emisor de luz infrarroja, el LED utilizado como flash de la cámara, y un sensor fotodetector. En la figura 4 se muestra este sensor.

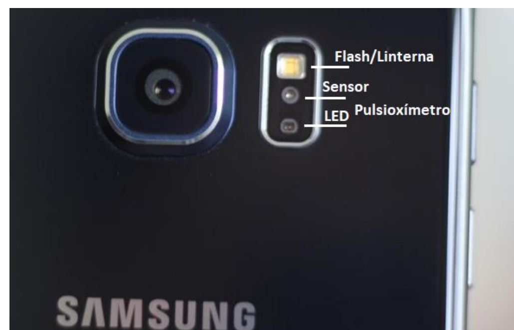

Figura 4. Sensor de pulsioximetría del Samsung Galaxy S6.

No obstante, es cierto que la precisión de este tipo de sensores está bajo discusión [6].

Volviendo a los sensores externos que ya existen en el mercado, podemos encontrar, por ejemplo, los siguientes:

· Biométrico: Huella dactilar:

- SMUFS-BT RAW (www.smufsbio.com)

  Este dispositivo es un escáner de huella portátil con Bluetooth 2.1 y USB. La imagen en crudo (escala de grises se envía al *smartphone*).

Figura 5. SMUFS-BT PRO (escáner de huella dactilar Bluetooth).
Fuente: www.smufsbio.com

- Fisiológico: Alcohol en aire espirado:
  - BACtrack Mobile Pro (www.bactrack.com)

  Este dispositivo es un alcoholímetro Bluetooth que mide la concentración de alcohol en sangre (*Blood Alcohol Content*) compatible con *smartphones* iPhone, Samsung Galaxy, y Google Nexus y Pixel.

Figura 6. BACtrack Mobile Pro (alcoholímetro Bluetooth).
Fuente: www.bactrack.com

- Fisiológico: Electroencefalograma:
  - MindWave Mobile (neurosky.com)

  Este dispositivo es un sensor de potencia espectral EEG (ondas alfa, beta, etc.) para medir niveles de atención y meditación.

Figura 7. MindWave Mobile (EEG Bluetooth).
Fuente: www.neurosky.com

2.3 Sistemas de control de acceso

El uso de NFC para el control de accesos está ampliamente extendido, desde las sencillas etiquetas (tags) en formato tarjeta o llavero, con un identificador único simple en texto plano, hasta complejos sistemas con *smartphone* y emulación de tarjetas criptográficas.

Incluso en la web del NFC Forum (nfc-forum.org), organismo de estandarización de especificaciones NFC, se puede consultar un listado de productos para control de accesos, donde se encuentra el producto "My Lock" para la apertura de armarios domésticos con el *smartphone*. En la figura 8 se muestra una captura de pantalla de la web del NFC Forum donde se muestra dicho producto.

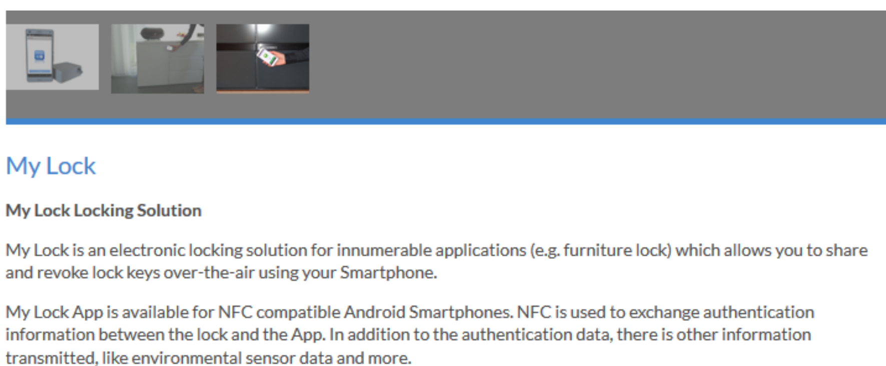

Figura 8. NFC Forum Product Showcase – My Lock.

El control de accesos con NFC ya ha sido objeto de Trabajos Fin de Grado anteriores, pero recientes, como el de Miguel Viñas (alumno de la UOC) [7], pero tan apenas describe la tecnología y sólo hace uso de la lectura del número de serie de una etiqueta pasiva.

Es ahí donde este Trabajo pretende dar un paso adelante, intentando que sea un *smartphone* el que genere la información que el sistema lector de NFC que controla algún tipo de cerradura necesita, y aportando como novedad la utilización de algún sensor biométrico o fisiológico similar a los analizados en los puntos anteriores, como fuente de los datos que sirvan para decidir si se debe autorizar la apertura o no.

3. Descripción del sistema propuesto

En este capítulo se expone la arquitectura del sistema propuesto, así como la elección de los elementos que compondrán el prototipo a desarrollar.

3.1 Descripción general de la arquitectura

En la figura 9 se puede observar la arquitectura a alto nivel del sistema propuesto en el Trabajo.

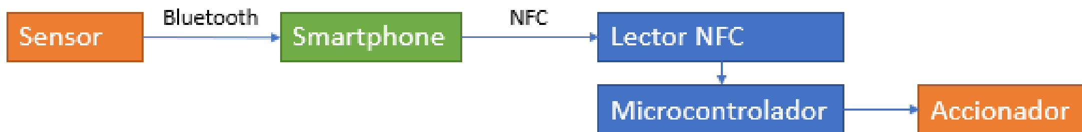

Figura 9. Arquitectura a alto nivel.

Un sensor captará la variable biométrica o fisiológica del individuo. En las siguientes secciones del Trabajo se expondrá el sensor escogido

El sensor enviará mediante comunicación Bluetooth los valores a una aplicación Android ejecutándose en un *smartphone*.

El *smartphone* podrá trasladar el valor de dicha medida mediante comunicación NFC a un microcontrolador, y éste tomará la decisión de permitir o no el acceso al recurso según el valor de la medida y cierto umbral, actuando sobre un accionador.

Pensando en una posible utilidad práctica, el sensor sería de tipo alcoholímetro como el BACtrack presentado en el estado del arte, y el accionador sería un bloqueo del sistema de arranque de un vehículo, donde, para permitir el arranque, la medida debería ser de 0 mg de alcohol. De hecho, según un estudio de la Universidad de Michigan [8], la instalación de sistemas de este tipo en los vehículos podría evitar un alto porcentaje de accidentes de tráfico donde el alcohol es la principal causa. En la figura 10 se observa la arquitectura de este sistema descrito como esta posible utilidad práctica.

Figura 10. Posible utilidad práctica del sistema propuesto.

3.2 Elección del sensor para el prototipo

A pesar del interés en un sensor de tipo alcoholímetro, para la realización del prototipo se ha optado por otro tipo de sensor por los motivos que se exponen a continuación.

La adquisición de un BACtrack implicaría un pedido internacional donde no se asegura el plazo de entrega. Además, aunque la comunicación es mediante Bluetooth, no existe información sobre el formato o protocolo en el que este dispositivo envía la medida al *smartphone* (en principio está diseñado para el uso con una aplicación propietaria).

Otra opción sería la fabricación propia de un sistema sensor mediante algún sensor de gases, como el conocido MQ-3 (que se observa en la figura 11). Pero esto implicaría el montaje de un circuito adicional con un microcontrolador y un módulo de comunicación Bluetooth, junto con el desarrollo del software del microcontrolador, ya que el sensor MQ-3 simplemente entrega un valor de tensión en su pin de salida. Ello también requiere de algún tipo de proceso de calibración.

Figura 11. Sensor de gases MQ-3 con salida analógica.
Fuente: sparkfun.com

Por todo ello (cuestiones prácticas, y plazo de realización de este Trabajo) se ha optado por la utilización de un sensor de frecuencia cardíaca de tipo fotoplestimográfico (óptico), como el presentado en el estudio del arte, pero ya desarrollado e integrado en un reloj inteligente o *smartwatch*, de los también denominados *wearables* (dispositivos electrónicos que se incorporan en alguna parte del cuerpo, como ropa, relojes, complementos, etc.).

La elección para el desarrollo rápido de un prototipo es muy apropiada ya que existen *wearables* de este tipo que ejecutan un sistema operativo Android denominado Wear OS [9], con el que la compatibilidad y comunicación Bluetooth con el *smartphone* está garantizada si éste ejecuta una versión de Android 4.4 o superior.

No obstante, habrá que desarrollar una aplicación Android para el *wearable*, que se encargue de la lectura del sensor y de la comunicación con el *smartphone*, como se expondrá en secciones posteriores.

Aunque este sensor se aleja de la utilidad práctica planteada en la sección anterior, la arquitectura general del sistema no se ve alterada, por lo que a efectos de desarrollo del prototipo es útil cualquier tipo de sensor.

El *smartwatch* elegido es el Mobvoi Ticwatch E (www.mobvoi.com). En la figura 12 se puede observar una fotografía del mismo y de su sensor.

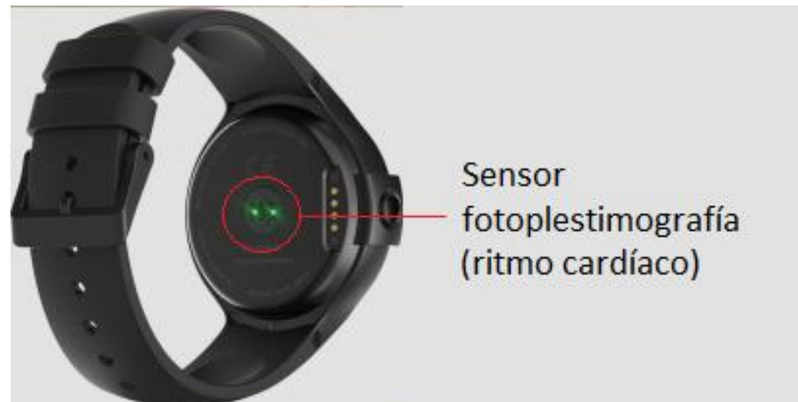

Figura 12. Sensor de ritmo cardíaco del Ticwatch E.

El sensor dispone de una fuente de luz verde (LED) que se emite sobre la piel de la muñeca para iluminar los vasos subcutáneos, y éstos reflejan parte de dicha luz (dependiendo de la cantidad de hematíes que contienen). Esta luz reflejada incide en un fotosensor que la convierte en un voltaje equivalente, y el sensor calcula el ciclo cardíaco midiendo el intervalo que existe entre cada pico de voltaje.

Las características más importantes del Ticwatch E son:

- Sistema Operativo: Wear OS 1.1 (basado en Android 8.0)
- Comunicaciones: Bluetooth v4.1 (Low Energy) y WiFi
- Sensores: Ritmo cardíaco, acelerómetro

3.3 Elección del microcontrolador, lector NFC y accionador

El microcontrolador escogido para el desarrollo del hardware del lector NFC y el control del acceso es el Atmel ATmega328P, ya que existe una plataforma de desarrollo de prototipos para este micro denominada Arduino [10], de la que existen en el mercado multitud de placas de circuito impreso con diversidad de componentes ya integrados como conector USB para su fácil conexión a un PC para cargar el código desarrollado, y existe un entorno de desarrollo de código libre (Arduino IDE) que mediante una librería específica de lenguaje C++, simplifica enormemente el desarrollo de código para dicho microcontrolador.

En concreto se ha escogido la placa Arduino UNO, que se puede ver en la figura 13 y que presenta las siguientes características:

- Microcontrolador: ATmega328P @ 16 MHz
- Tensión de alimentación: 5 V
- Flash / SRAM / EEPROM: 32 kB / 2 kB / 1 kB
- Comunicación: Serie TTL, SPI, TWI/I^{2}C

Figura 13. Placa de desarrollo Arduino UNO.

Para la comunicación NFC se ha escogido el circuito integrado controlador NXP PN532 [11] que soporta los siguientes modos de comunicación:

- ISO/IEC 14443A/MIFARE + Card emulation
- FeliCa + Card emulation
- ISO/IEC 14443B
- ISO/IEC 18092, ECMA 340 Peer-to-Peer

Más adelante en el Trabajo se profundizará sobre el modo de comunicación NFC escogido.

En el mercado se pueden adquirir placas de circuito o módulos que contienen al circuito PN532 junto con los componentes necesarios para su funcionamiento, así como una antena NFC ya integrada en la propia placa de circuito. Se ha escogido el módulo NFC MODULE v3 de elechouse.com, que se puede observar en la figura 14.

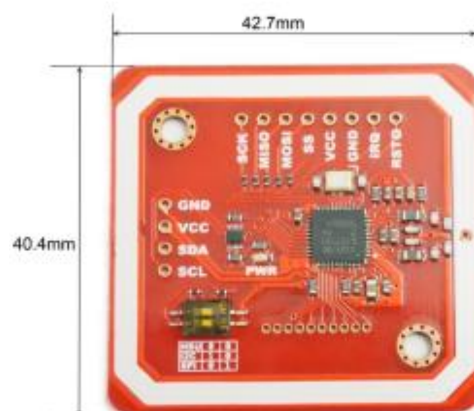

Figura 14. Módulo PN532 NFC (elechouse.com).

El integrado PN532 presenta una interfaz de comunicación I²C (*Inter-Integrated Circuit*, bus serie desarrollado por Philips-NXP) que será la utilizada para la comunicación con el microcontrolador de Arduino. El bus I2C presenta dos cables: SDA (datos) y SCL (reloj).

En cuanto al accionador, por cuestiones prácticas, será simplemente un diodo LED que ya contiene la propia placa Arduino UNO, pero podría ser

un elemento más avanzado como una cerradura eléctrica o el bloqueo de arranque de vehículo.

Por todo ello la conexión física entre Arduino y módulo NFC será la mostrada en el esquemático de la figura 15, donde se observa que la alimentación (VCC) del módulo provendrá de la propia placa de Arduino:

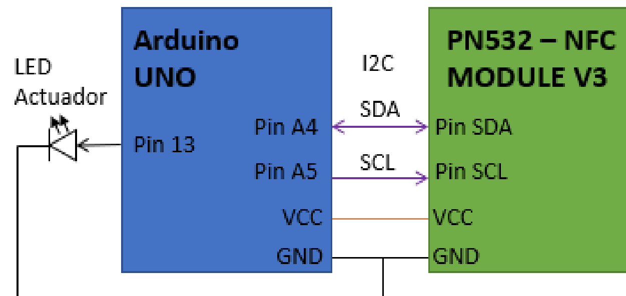

Figura 15. Esquemático de conexiones entre Arduino y módulo NFC.

3.4 Elección del *smartphone*

El *smartphone* escogido es el que el autor del Trabajo tiene a su disposición, pero serviría cualquier *smartphone* que ejecute Android 4.4 o superior, y que posea NFC y Bluetooth.

En concreto el utilizado es BQ Aquaris M5 ejecutando Android versión 7.1.5. El *wearable* se debe emparejar con el *smartphone* mediante la aplicación Wear OS disponible en Google Play Store.

4. Descripción de aplicaciones Android

En este capítulo se describen el sistema operativo Android y los conocimientos básicos necesarios para abordar el diseño y desarrollo de aplicaciones para el mismo. También se expone la comunicación Android con Wear OS y NFC, y finalmente la arquitectura *software* del sistema prototipo a implementar.

4.1 Android y desarrollo de aplicaciones

Android es un sistema operativo móvil desarrollado por Google, basado en software de código abierto y en una versión modificada del *kernel* de Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil como *smartphones* y tabletas.

Como ya ha quedado patente en la descripción general del sistema, el elemento central y que implementará los dos tipos de comunicación inalámbrica (Bluetooth y NFC) será el *smartphone*. Habrá que desarrollar una aplicación para éste que será la encargada de gestionar ambas comunicaciones.

Del mismo modo, el *wearable* también requiere del desarrollo de una aplicación Android que ejecutará para gestionar la lectura de la información del sensor y el envío de la misma al *smartphone* mediante Bluetooth.

Toda la documentación sobre Android necesaria para el desarrollo de aplicaciones está disponible en la página web Android Developers [11].

Las aplicaciones para Android se pueden escribir en los lenguajes Kotlin, Java o C++. Las herramientas del paquete de desarrollo (SDK) de Android compilan el código junto con ficheros de datos y recursos en un paquete Android (APK), que es un fichero con extensión `.apk` que utilizan los dispositivos Android para instalar la aplicación. Existe un IDE (entorno de desarrollo) propuesto por Google y que contiene todas las herramientas para la edición de código y compilación denominado Android Studio.

Las aplicaciones están formadas por componentes, que son puntos de entrada mediante los cuales el sistema o un usuario pueden acceder a la aplicación. Hay cuatro tipos diferentes de componentes, y cada uno de ellos sirve para una finalidad distinta y tiene un ciclo de vida distinto (que define cómo se crea y se destruye un componente):

- Actividades: son el punto de entrada de interacción con el usuario, y representan una única pantalla con una interfaz de usuario. Se implementan mediante la clase Java denominada `Activity`.
- Servicios: son puntos de entrada de propósito general utilizados para ejecutar procesos en segundo plano, y no proporcionan interfaz de usuario. Se implementan mediante la clase `Service`.

- Receptores de mensajes: habilitan al sistema para enviar eventos (alarmas, batería baja, pantalla apagada, …) a las aplicaciones fuera del flujo de uso normal, permitiendo a la aplicación responder a dichos eventos, incluso sin estar la aplicación en ejecución. Las aplicaciones también pueden generar eventos, y aunque estos receptores no disponen de interfaz gráfica por sí mismos, pueden crear notificaciones en la barra de tareas. Se implementan mediante la clase `BroadcastReceiver`, y cada evento se envía como un objeto de tipo Intención (`Intent`).
- Proveedores de contenido: gestionan un conjunto de datos de aplicación compartidos y que pueden estar almacenados en el sistema de ficheros, una base de datos SQLite, la web, o cualquier otra localización persistente a la que la aplicación pueda acceder. Se implementan mediante la clase `ContentProvider`, y deben implementar un conjunto de APIs para permitir a otras aplicaciones ejecutar transacciones.

Cuando el sistema inicia un componente, inicia el proceso de su aplicación (si es que no estaba ya en ejecución) y se instancian las clases necesarias para ese componente. Cualquier aplicación puede iniciar un componente de otra aplicación, pero dado que el sistema ejecuta cada aplicación en procesos diferentes con permisos de fichero que restringen el acceso a otras aplicaciones, los componentes se inician a través del sistema. Una aplicación envía al sistema un mensaje que especifica la intención de iniciar un componente concreto, y el sistema lo activa.

Tres de los cuatro tipos de componentes (actividades, servicios, y receptores de mensajes) se activan mediante un mensaje asíncrono denominado intención (*intent*). Las intenciones asocian componentes individuales entre sí en tiempo de ejecución. Estas intenciones se crean con objetos `Intent`, que definen un mensaje con el propósito de activar un componente en concreto (intención explícita), o un tipo de componente (intención implícita).

Para las actividades y servicios, una intención define la acción a ejecutar, junto con elementos que el componente que se inicia puede necesitar. Si la actividad iniciada debe devolver un resultado, éste se devuelve también en un objeto de intención. Para los receptores de mensajes las intenciones simplemente definen el mensaje que se anuncia. Esto es:

- Se puede iniciar una actividad o enviarle información nueva enviando un objeto de intención invocando al método `startActivity()` (o al método `startActivityForResult()` si se desea que la actividad devuelva un resultado).
- Se puede iniciar un servicio o enviarle instrucciones nuevas enviando un objeto de intención invocando al método `startService()`. Se puede crear una asociación con el servicio enviando un objeto de intención invocando al método `bindService()`. Con Android 5.0 o

superior, se puede usar la clase `JobScheduler` para programar acciones.

- Se puede iniciar un receptor enviando un objeto de intención invocando métodos Java de Android como `sendBroadcast()`, `sendOrderedBroadcast()` o `sendStickyBroadcast()`.

Los proveedores de contenido no se activan con intenciones, sino cuando son objeto de una petición mediante el método `query()` de un `ContentResolver`.

Toda aplicación debe disponer de un fichero manifiesto (`AndroidManifest.xml`), que describe información esencial sobre la aplicación, necesaria para las herramientas de compilación, el propio sistema operativo Android, y la plataforma de distribución Google Play Store. En este fichero se declaran los permisos que requiere la aplicación para acceder a partes del sistema protegidas, así como los componentes de la aplicación y sus filtros de intención (esto es, a qué intenciones de otras aplicaciones puede responder).

4.2 Ciclo de vida de las Actividades

Las actividades pasan por diferentes etapas en su ciclo de vida, y su clase proporciona una serie de retrollamadas (*callbacks*) que le permiten conocer que el estado ha cambiado, esto es, que el sistema está creando, deteniendo, retomando, o destruyendo el proceso en el que la actividad reside.

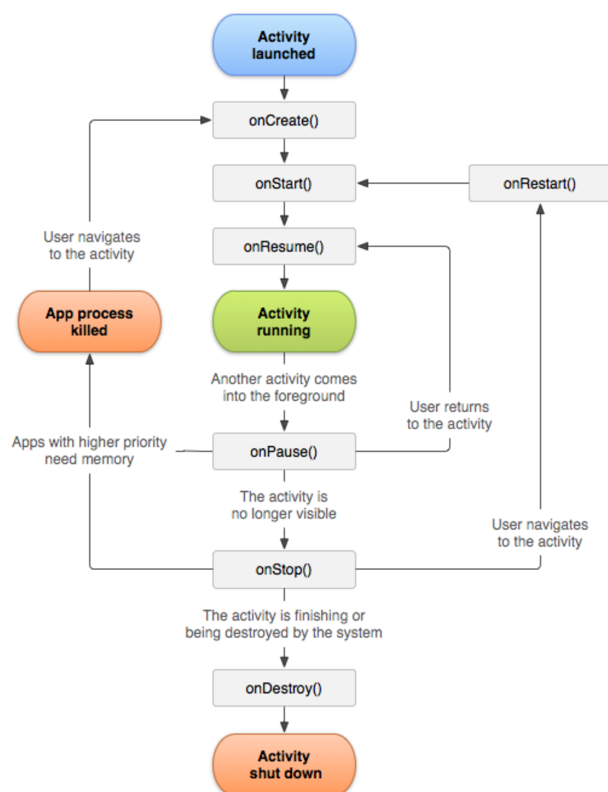

Figura 16. Ciclo de vida de una Actividad.
Fuente: Android Developers (developer.android.com)

En la figura 16 se puede observar un diagrama de dicho ciclo de vida con los métodos a los que se invoca en cada caso.

4.3 Gestores de eventos y lectura de sensores

Los gestores o *listeners* en Android se utilizan para implementar el código que se debe ejecutar cuando ocurre un evento, invocando retrollamadas asíncronas a ciertos métodos. Estos gestores se pueden utilizar para capturar la interacción con la interfaz de usuario

Los dispositivos que ejecutan Android suelen disponer de sensores que proporcionan datos en crudo con alta precisión. Se puede acceder a esos datos en crudo mediante el *framework* de sensores, que proporciona varias clases e interfaces Java para realizar diversas tareas relacionadas con los mismos, como por ejemplo:

· Determinar los sensores disponibles.

· Determinar las capacidades de un sensor (rango, resolución, …).

· Adquirir datos en crudo del sensor y definir la tasa de muestreo.

· Registrar y de-registrar gestores de eventos que monitorizan los cambios del sensor.

Para el último punto se requiere implementar dos retrollamadas que se exponen mediante la interfaz `SensorEventListener`:

- `onAccuracyChanged()`: el sistema la invoca cuando la precisión del sensor se ve modificada. La precisión puede ser `LOW`, `MEDIUM`, `HIGH` o `UNRELIABLE`, aunque no todos los sensores soportan los cuatro tipos.
- `onSensorChanged()`: el sistema lo invoca cuando el sensor informa de un nuevo valor, proporcionando un objeto `SensorEvent` que contiene el valor junto con su precisión y una marca de tiempo. Android permite especificar la frecuencia de muestreo (`DELAY`) entre los siguientes valores: `FASTEST`, `GAME`, `NORMAL`, `UI`. Pero estos valores sólo son orientativos, basados en el nivel de uso de la interfaz gráfica, y el sistema Android y el sensor deciden de manera autónoma cuándo reportar valores. El programador puede saber la diferencia de tiempo entre valores con la marca de tiempo.

4.4 Comunicación con Wear OS – Data Layer

Google proporciona una API denominada Data Layer (capa de datos) que proporciona un canal de comunicación para compartir información entre el *smartphone* y el *wearable*, sin que el desarrollador tenga que entrar en los detalles de bajo nivel de la comunicación Bluetooth.

Esta capa consiste en un conjunto de objetos que el sistema puede enviar y/o sincronizar, junto con unos gestores de eventos que pueden usar las aplicaciones:

- Data Items: proporcionan almacenamiento de datos con sincronización automática entre *smartphone* y *wearable*.
- Assets: objetos para el envío de datos binarios (como imágenes).
- Mensajes: envío de mensajes para comunicación unidireccional o de tipo petición/respuesta. Si los dos dispositivos están conectados, el sistema encola los mensajes para su entrega y devuelve un código de resultado satisfactorio. Si los dispositivos no están conectados, se devuelve un error.
- Canal: adecuados para el envío de entidades grandes (como vídeos) desde el *smartphone* hacia el *wearable*.
- `WearableListenerService`: servicio que permite escuchar en segundo plano eventos que ocurran en la capa Data Layer.
- `OnDataChangedListener`: gestor de eventos que se puede implementar dentro de una actividad en primer plano.

Para el caso de los Mensajes, utilizando el cliente `MessageClient` se adjuntan al mensaje los siguientes elementos:

- Una carga (*payload*) arbitraria y opcional.
- Una ruta (*path*) que identifica de manera única la acción del mensaje.

4.5 Comunicación NFC – Host-based Card Emulation

Los dispositivos Android que ofrecen funcionalidades NFC soportan los tres modos diferentes de operación que NFC contempla:

- Lectura/escritura de etiquetas pasivas.
- Modo P2P, para el intercambio de información con otro dispositivo (en Android se denomina modo Android Beam).
- Emulación de tarjetas, que permite el acceso por parte de un lector NFC externo a información de una tarjeta inteligente emulada.

Ese último modo es el que se utilizará en este Trabajo. Aunque en algunos casos la tarjeta se emula mediante un chip independiente (el denominado elemento seguro, conocido por sus siglas SE en inglés), Android 4.4 introdujo la emulación de tarjeta sin necesidad de SE. Esta emulación se denomina "*host-based card emulation*" (HCE), y permite que cualquier aplicación Android emule una tarjeta y se comunique directamente con un lector NFC basándose en la especificación ISO-DEP (Data Exchange Protocol) del NFC-Forum (basada a su vez en ISO/IEC 14443-4). Esto permite a Android procesar paquetes APDUs (*Application Protocol Data Units*) según se define en la especificación ISO/IEC 7816-4. En la figura 17 se puede observar la pila de protocolos de esta emulación HCE.

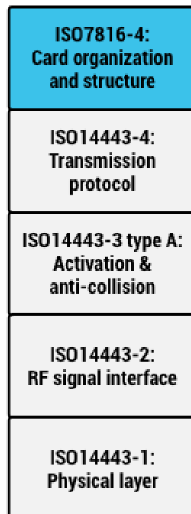

Figura 17. Pila de protocolos HCE de Android.
Fuente: Android Developers (developer.android.com)

HCE de Android se basa en los componentes de tipo “Servicios HCE”, que pueden ejecutarse en segundo plano sin interfaz de usuario. Al acercar el dispositivo a un lector NFC, se inicia el servicio adecuado según un Identificador de Aplicación (AID), descritos también en la especificación ISO/IEC 7816-4.

En su estructura más simple, los servicios HCE de Android se pueden implementar con la clase `HostApduService`, que presenta dos métodos a sobrescribir:

- `byte[] processCommandApdu()`: se invoca cuando el lector NFC envía un paquete APDU. Estos paquetes APDUs se intercambian en modo semidúplex. El lector NFC envía un APDU con un comando, y espera recibir un APDU como respuesta. Este método es invocado por el hilo de ejecución principal de la aplicación, por lo que el código de la aplicación no debe provocar bloqueos. Si la respuesta se va a demorar, es mejor devolver ‘null’ y ejecutar el trabajo necesario en otro hilo, usando el método `sendResponseApdu()` para devolver la respuesta una vez se tenga.
- `onDeactivated()`: se invoca, con un argumento que indica el motivo, cuando Android deja de retransmitir nuevos APDUs del lector al servicio. Esto ocurriría, o bien cuando el enlace NFC se interrumpiese, o cuando el lector enviase otro APDU de tipo “`SELECT AID`” con otro AID distinto (esto es, cuando el lector desea cambiar de Aplicación según define ISO/IEC 7816-4).

Se debe intentar limitar el tamaño y la cantidad de APDUs para asegurar que los usuarios sólo tienen que mantener el dispositivo cerca del lector NFC durante poco tiempo. Un límite razonable es 1 KiB de información, que normalmente se intercambia en 300 ms.

Según ISO/IEC 7816-4 los paquetes APDU de tipo comando y respuesta tienen la siguiente estructura:

- Comando:
  - Cabecera obligatoria de 4 bytes:
    - `CLA`: indica la clase de comando (p.ej. propietario)
    - `INS`: indica la instrucción a procesar (p.ej. escribir datos)
    - `P1`, `P2`: indican opciones para la instrucción (p.ej. un offset)
  - Cuerpo de longitud variable

- Respuesta:
  - Cuerpo de longitud variable
  - Cola obligatoria de 2 bytes:
    - `SW1`, `SW2`: indican el estado tras la ejecución del comando.

Estos comandos y respuestas pueden seguir algún protocolo existente, o ser completamente arbitrarios (esto es, protocolo propietario entre lector y tarjeta).

Además de declarar el servicio en el fichero de manifiesto, la declaración del servicio debe contener un filtro de intención para este servicio `HOST_APDU`.

4.6 Arquitectura de las aplicaciones Android del sistema a implementar

En la figura 18 se puede observar la arquitectura de componentes de las dos aplicaciones Android que se desarrollarán.

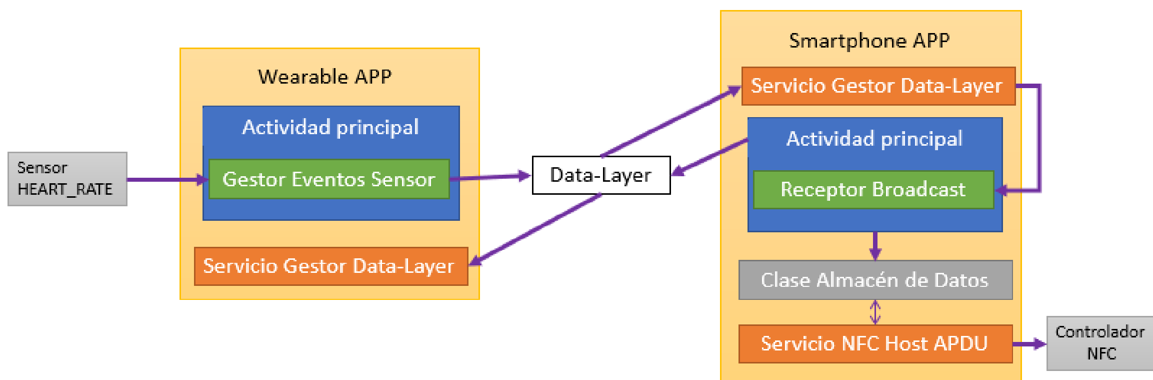

Figura 18. Arquitectura de las aplicaciones Android del sistema.

En el *wearable*, la aplicación instalará un servicio Gestor en segundo plano que escuchará mensajes que reciba la capa Data-Layer, de modo que cuando se reciba un mensaje con una ruta concreta (por ejemplo `/start/MainActivity`), automáticamente se abrirá la Actividad principal. De este modo el *smartphone* puede lanzar la actividad del *wearable* de manera automática.

La actividad principal iniciará el sensor de frecuencia cardíaca y registra un Gestor de eventos para el sensor, con una periodicidad de lectura de tipo `DELAY_NORMAL`. Este gestor recibirá el valor de precisión del sensor y el valor de la medida, y hará que la Actividad muestre en pantalla el valor de la medida.

El sensor utilizado es capaz de diferenciar entre tres niveles de precisión (BAJA, MEDIA, ALTA). No existe información sobre cómo decide el sensor la precisión de la medida, pero probablemente se apoye en el acelerómetro que incorpora el *wearable* (ya que se puede comprobar que si se agita la muñeca mientras se está tomando la medida, la precisión se ve reducida).

Si la precisión de la medida no es BAJA, el Gestor enviará el valor de la medida (pulsaciones por minuto) a la Data-Layer para compartirla con el *smartphone*, mediante la ruta `/pulsaciones`. Con precisiones BAJA o MEDIA, la Actividad mostrará en la pantalla del *wearable* un mensaje al usuario para que intente mantenerse quieto para que la precisión del sensor se incremente, como se observa en la figura 19.

Figura 19. Mensaje que se muestra en pantalla si la precisión del sensor no es ALTA

En cuanto a la aplicación del *smartphone*, al ejecutarla se lanzará su Actividad principal, que como se ha indicado anteriormente, enviará un mensaje a la Data-Layer para que se inicie la actividad principal de la aplicación del *wearable*, que se abrirá si no estaba ya en ejecución.

La aplicación instalará un servicio Gestor en segundo plano que escuchará mensajes que reciba la capa Data-Layer, de modo que cuando se reciba un mensaje con la ruta `/pulsaciones`, el servicio extraerá el valor recibido y lo retransmitirá como un evento Broadcast de manera local.

Ahí es donde entra en acción el Receptor de mensajes *broadcast* de la Actividad principal, que recibirá el mensaje y almacenará el valor de la medida junto con la fecha y hora de recepción de la misma mediante una clase auxiliar (en la figura 18, la clase Almacén de Datos). Esta clase Almacén guardará la información mediante una interfaz que ofrece Android denominada `SharedPreferences`, que es un almacén de tipo clave-valor al que pueden acceder las aplicaciones.

En la figura 20 se puede ver la interfaz gráfica de la aplicación del *smartphone*, tras recibir una medida de pulsación.

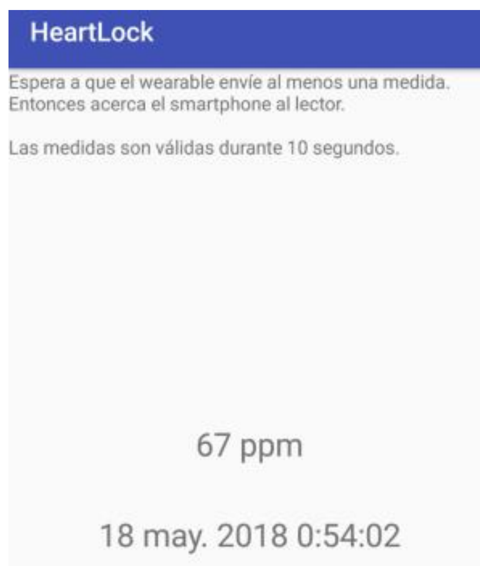

Figura 20. Interfaz gráfica de la aplicación del smartphone

La aplicación también instala un servicio `HostApduService` para la comunicación NFC. Este Servicio se invoca cuando el controlador NFC del *smartphone* recibe un APDU con un comando de tipo SELECT AID (selección de aplicación según ISO/IEC 7816-4) y el código de aplicación AID coincide con el (o los) definido(s) en el filtro declarado en el fichero Manifiesto de la aplicación. Esto es, al acercar el *smartphone* al lector NFC, el lector deberá emitir un comando APDU de selección de aplicación con un AID concreto para que la aplicación responda. El *smartphone* vibrará para hacer saber al usuario que se está estableciendo comunicación NFC.

Los identificadores AID pueden ser de hasta 16 bytes, y según el estándar el primer *nibble* (medio byte) indica la categoría de la aplicación, que puede ser Internacional (A), Nacional (D), Estándar (E), o Propietaria (F). Excepto los de clase propietaria, el resto se deben registrar según los procedimientos establecidos en ISO/IEC 7816-5 para garantizar que son únicos. En este caso será una aplicación propietaria, y se ha escogido un AID arbitrario de 7 bytes: **F**0394148148100.

Si el AID seleccionado por el lector es el que el Servicio espera, el Servicio sabrá que el lector le está solicitando la medida del sensor almacenada en el Almacén de la aplicación. La medida se enviará como una cadena de 3 caracteres ASCII con el valor numérico de las pulsaciones por minuto (por ejemplo: 068).

Como un pequeño mecanismo de verificación de que la información del sensor es reciente, se programará que el Servicio sólo envíe la información real de la medida al lector NFC si la fecha y hora de la medida registrada en el Almacén no es más antigua de 10 segundos. Esto es, desde la adquisición de la medida de pulsaciones, se dispone de 10 segundos para acercar el *smartphone* al lector NFC. Si la medida caduca, el *smartphone* mostrará una notificación en la pantalla y el dato enviado serán tres caracteres ‘X’ (XXX).

5. Descripción de NFC y el código del lector

En este capítulo se describe el protocolo de emulación de tarjetas utilizado, así como la manera en la que se va a implementar en el microcontrolador de Arduino.

5.1 Estándares NFC para tarjetas inteligentes

Aunque en todas las secciones anteriores del Trabajo ya se ha nombrado NFC en varias ocasiones, en esta sección se presenta una definición formal. NFC es un tipo de tecnología de identificación por radiofrecuencia (RFID, sus siglas en inglés) que utiliza la inducción electromagnética mediante dos antenas de espira para el intercambio de información, y que opera en la banda de radiofrecuencia HF de 13,356 MHz (banda ISM), con tasas de transferencia de entre 106 y 424 kbps.

La tecnología NFC está definida principalmente en el estándar ISO/IEC 18092, donde se definen los tres modos de funcionamiento citados anteriormente: lectura/escritura de etiquetas, P2P, y emulación de tarjetas.

Para la emulación de tarjetas, NFC debe satisfacer el estándar ISO/IEC 14443 (*Identification cards -- Contactless integrated circuit cards -- Proximity cards*). Este estándar 14443 se compone de cuatro partes:

- 14443-1: Características físicas
- 14443-2: Interfaz de señal y potencia RF
- 14443-3: Inicialización y anticolisión
- 14443-4: Protocolo de transmisión

El estándar 14443 también define dos tipos de tarjeta: A y B. La diferencia entre los dos tipos son los métodos de modulación y esquemas de codificación, así como los protocolos de inicialización, pero el tipo A es el más ampliamente utilizado.

En la figura 17 observábamos que la emulación HCE de Android se sustenta sobre estos estándares 14443 en su pila de protocolos.

La asociación industrial NFC Forum [12] se encarga de promover el uso de la tecnología NFC, impulsando su implementación y estandarización asegurando la interoperabilidad entre dispositivos y servicios. Sus normas, que se apoyan sobre ISO/IEC 18092 y ISO/IEC 14443, principalmente se centran en la estandarización de un formato de intercambio de mensajes denominado NDEF (*NFC Data Exchange Format*) y su uso con los diversos tipos de etiquetas pasivas NFC existentes.

Volviendo a las tarjetas inteligentes, el estándar que define todas sus características es el ISO/IEC 7816. Es de especial importancia la parte 4 del estándar, donde se definen los comandos para el intercambio de información. De nuevo podemos recordar que en la figura 17 sobre la

emulación HCE de Android, este estándar 7816-4 es el que ocupa la capa superior de la pila de protocolos.

5.2 Protocolo de emulación de tarjetas ISO/IEC 7816-4 y Android

Para construir una infraestructura de lector compatible con los dispositivos HCE Android, se debe conocer qué parámetros de protocolo utilizan los dispositivos HCE durante las fases de anticolisión y activación. Como parte de la activación del protocolo `Nfc-A` (así se denomina la clase Android según las convenciones de nombres de Java) se produce el intercambio de múltiples tramas.

En la primera parte del intercambio el dispositivo HCE presenta su identificador único (UID) de 4 bytes; hay que tener en cuenta que los dispositivos HCE deberían enviar un UID generado aleatoriamente cada vez que se acerquen a un lector. Por ello, los lectores NFC para tarjetas emuladas no deben depender de este valor UID como forma de autenticación o identificación, como sí se suele hacer con las etiquetas NFC del tipo más sencillas, donde se confía en que el UID es único, escrito de fábrica, y no modificable.

Después el lector NFC podrá seleccionar al dispositivo HCE enviando un comando denominado `SEL_REQ`, y la respuesta `SEL_RES` del dispositivo HCE tendrá al menos su sexto bit activo (0x20) para indicar que soporta ISO-DEP. Otros bits también pueden estar activos si el dispositivo quiere indicar que soporta otros protocolos como por ejemplo NFC-DEP (P2P), por lo que los lectores que quieran interactuar con dispositivos HCE deben comprobar explícitamente el sexto bit.

Tras la activación del protocolo `Nfc-A`, el lector inicia la activación del protocolo ISO-DEP, enviando para ello un comando `RATS` (*Request for Answer To Select*). La respuesta `RATS`, la `ATS`, la genera el controlador NFC y no es configurable por parte de los Servicios HCE de Android. No obstante, las implementaciones HCE deben cumplir los requerimientos del NFC Forum para la respuesta ATS, por lo que los lectores NFC pueden confiar en que dichos parámetros estarán ajustados conforme a los requerimientos.

5.3 Arduino, I²C, y Módulo PN532

Para la programación de código C++ que se ejecutará en el microcontrolador de la placa Arduino, existe un entorno de código libre denominado Arduino IDE, que ya contiene las librerías necesarias de protocolo I²C (librería `Wire.c`) para poner en marcha el desarrollo.

Los programas de Arduino se estructuran en base a dos funciones:

- `setup()`: se ejecuta una única vez tras el arranque en frío del microcontrolador o tras su reseteo. Idónea para realizar operaciones de configuración de comunicaciones.

- `loop()`: se ejecuta cíclicamente tras la función `setup`. Aquí irá el algoritmo principal del programa, que podrá invocar a otras funciones o procedimientos definidos fuera de estas dos funciones.

En el protocolo de comunicación I²C un dispositivo actúa como Maestro (en este caso el Arduino) y diversos periféricos que actúan como esclavos, que atienden peticiones de lectura o escritura del maestro. Cada uno de estos periféricos posee una dirección de tamaño 1 byte predeterminada por el fabricante, y el maestro dirige las peticiones a esta dirección. En el caso del módulo PN532, la dirección según se describe en el manual del circuito integrado [12] es 0x48.

Arduino por tanto podrá invocar cinco métodos que proporciona la librería Wire.c para hacer uso de esta comunicación:

- `Wire.beginTransmission(dirección)`: para indicar al esclavo que se va a comenzar la transmisión de información.
- `Wire.write(bytes)`: para enviar al esclavo bytes de información.
- `Wire.endTransmission()`: para indicar al esclavo que se ha finalizado la transmisión de información.
- `Wire.requestFrom(dirección, cantidad_bytes)`: para solicitar al esclavo el envío de una cantidad de bytes de información.
- `Wire.read()`: para obtener del esclavo los bytes de información que se le habían solicitado.

De este modo Arduino enviará al integrado PN532 comandos y leerá sus respuestas. El manual del integrado PN532 describe en su sección "6.2 Host controller communication protocol" el formato y el protocolo de los comandos y respuestas que soporta. El formato de los paquetes de comando se puede observar en la figura 21:

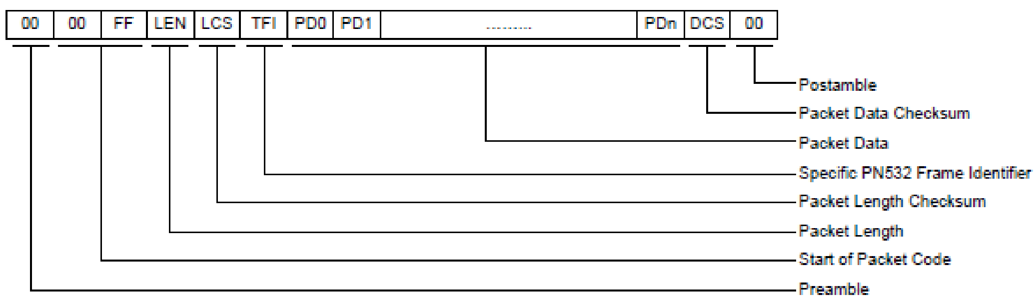

Figura 21. Estructura de un paquete de comando para el integrado PN532
Fuente: PN532 User Manual [12]

En el código del Arduino se van a definir una serie de funciones que se encargarán de realizar las tareas de generación de los paquetes para la comunicación con el PN532, siguiendo las indicaciones del manual del integrado, y que el código del programa principal (u otras funciones más específicas) invocarán cuando las requieran. Éstas serán:

- `writeCommand()`: para el envío de un comando al PN532.

- `readAckFrame()`: para la confirmación de recepción.
- `getResponseLength()`: para la obtención del tamaño de la respuesta.
- `readResponse()`: para la obtención de los bytes de la respuesta.

El diagrama de flujo de las funciones `setup` y `loop` a implementar en el Arduino se muestra en la figura 22:

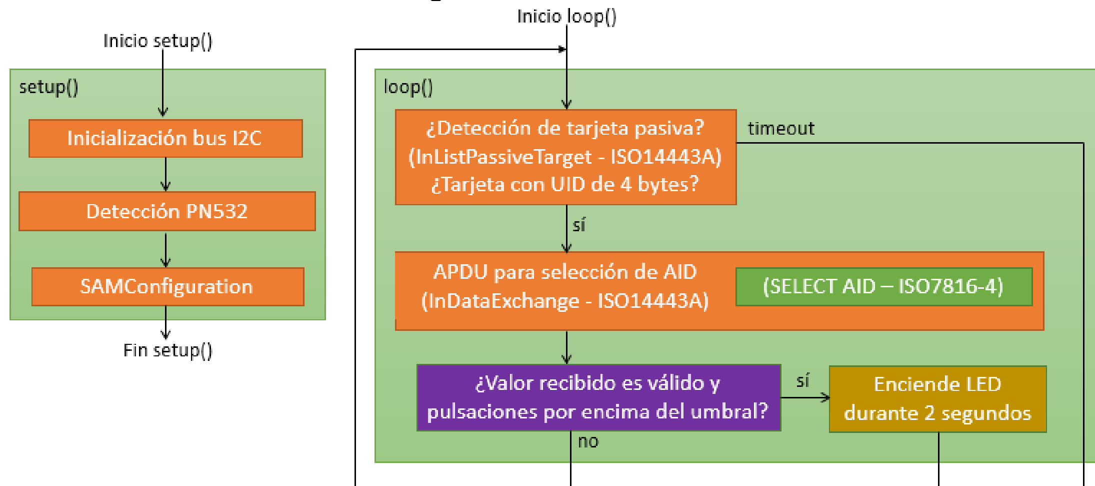

Figura 22. Diagrama de flujo del programa a ejecutar en el microcontrolador con Arduino

Cuando se ejecute la función `setup`, lo primero que se hará será inicializar el bus I²C para que el dispositivo PN532 se active (*wakeup*). Tras ello se lanzará un comando al PN532 para detectar su presencia y comprobar su versión de firmware.

Después se enviará el comando `SAMConfiguration` al PN532 para seleccionar el camino de flujo de datos (*data flow path*), tal y como indica su manual de instrucciones. Éste se configurará con el Modo Normal, ya que no se va a utilizar ningún SAM (*Security Access Module*).

Adicionalmente y por motivos de depuración, se establecerá una conexión serie para enviar por USB a un puerto COM del PC mensajes de texto en diferentes puntos del programa. Esto es muy conveniente para vigilar durante el desarrollo que todo está funcionando como se espera.

En la función iterativa `loop` se lanzará al PN532 el comando de detección de tarjeta pasiva (`InListPassiveTarget`) indicando que lo que se espera es una tarjeta ISO-14443 de tipo A. Este comando espera durante un *timeout* de 1 segundo a que se aproxime una tarjeta. En caso de detectar una, el comando devolverá su identificador UID junto con otros parámetros que se pueden observar en la figura 23. En dicha figura el UID se corresponde con el NFCID.

– **106 kbps Type A:**

Tg	SENS_RES[11] (2 bytes)	SEL_RES (1 byte)	NFCIDLength (1 byte)	NFCID1[] (NFCIDLength bytes)	[ATS[]] (ATSLength bytes[12])

Figura 23. Respuesta al comando InListPassiveTarget al detectar una tarjeta tipo A. Fuente: PN532 User Manual [12]

El UID que envía Android es de 4 bytes, aleatorio como ya se ha indicado anteriormente.

Posteriormente el microcontrolador enviará al PN532 un comando de tipo `InDataExchange` que contendrá la instrucción "`SELECT AID by name`" según define ISO/IEC 7816-4. El AID enviado será el mencionado anteriormente y configurado en la aplicación Android: **F**0394148148100. En la figura 24 se puede ver la tabla que contiene el documento del estándar donde indica la estructura de la instrucción `SELECT`.

Table 38 — SELECT command-response pair

CLA INS P1 P2	As defined in 5.1.1 'A4' See Table 39 See Table 40
L_c field	Absent for encoding $N_c = 0$, present for encoding $N_c > 0$
Data field	Absent or file identifier or path or DF name (according to P1)
L_e field	Absent for encoding $N_e = 0$, present for encoding $N_e > 0$

Data field	Absent or file control information (according to P2)
SW1-SW2	See Tables 5 and 6 when relevant, e.g., '6283', '6284', '6A80', '6A81', '6A82', '6A86', '6A87'

Figura 24. Estructura del comando SELECT y su respuesta.
Fuente: ISO/IEC 7816-4 [13]

El primer byte es el byte de clase (`CLAss byte`) que indica la clase del comando, que en este caso es 0x00 (*Interindustry*). El byte P1 se debe establecer a 0x04 para indicar que se selecciona el DF (*Dedicated File*) de la tarjeta por nombre, esto es, por identificador de aplicación AID.

La respuesta que devuelva el *smartphone* contendrá los bytes de estado `SW1-SW2`. El valor de estos bytes para una respuesta satisfactoria será 0x9000, y así tendrá que enviarlos nuestra aplicación Android tras los tres bytes de caracteres ASCII con la medida de pulsaciones. Si lo recibido son tres caracteres 'X', Arduino sabrá que la medida no es válida pues estará caducada.

Otro posible valor devuelto en `SW1-SW2` por parte de los *smartphones* Android puede ser 0x6A82 si no existe ninguna aplicación que maneje el AID seleccionado.

Una vez Arduino ya posee el dato de la medida, lo comparará con un umbral preconfigurado para tomar la decisión de si activar o no al actuador, en este caso, iluminar durante 2 segundos el LED que incorpora la placa Arduino.

Tras todo ello, la función *loop* se repite indefinidamente.

6. Prueba práctica del sistema

En este capítulo se describe el escenario final del prototipo, y se ejecutan las pruebas para comprobar el correcto funcionamiento del sistema completo. También se analiza el coste total del prototipo.

6.1 Escenario

Para el desarrollo del sistema completo se han ido haciendo pruebas unitarias y las modificaciones del código necesarias hasta lograr el comportamiento definido en las secciones anteriores, apoyándose en la impresión por pantalla o consola serie de mensajes de depuración y el contenido de los bytes en crudo de algunos comandos o mensajes.

Una vez implementados todos los desarrollos, que se adjuntan en los Anexos de este Trabajo, se puede poner a prueba la solución completa. En la figura 25 podemos observar la arquitectura del prototipo.

Figura 25. Arquitectura del prototipo.

En la figura 26, se muestra una fotografía de todos los elementos reales del escenario completo del prototipo.

Figura 26. Escenario completo (Arduino UNO, PN532, wearable Wear OS, smartphone Android).

Por motivos de depuración, mediante el puerto USB la placa Arduino recibirá alimentación y enviará mensajes por terminal serie a un PC.

6.2 Pruebas y resultados

Se parte de la situación en la que tanto *wearable* como *smartphone* tienen el Bluetooth activado, están emparejados mediante la aplicación Wear OS, y tiene cada uno su aplicación instalada desde el IDE Android Studio. El *wearable* está colocado en la muñeca, y el *smartphone* también tiene activado el NFC.

Se observa en la consola serie del PC que el Arduino ha detectado correctamente al PN532 y ha iniciado la ejecución de su código, como se puede ver en la figura 27.

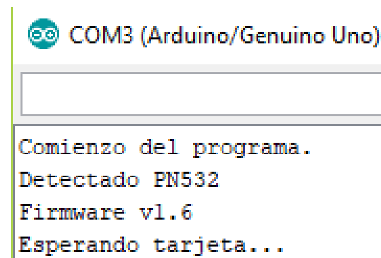

Figura 27. Mensajes en consola del PC tras el arranque de Arduino.

Al abrir la aplicación en el *smartphone* (se ha llamado 'HeartLock' a la misma), se comprueba que efectivamente se abre de manera automática la aplicación en el *wearable*, y que ésta comienza a tomar la medida del pulso cardíaco. En la figura 28 se observan las pantallas que se muestran tanto en la *app* móvil como en el *wearable*.

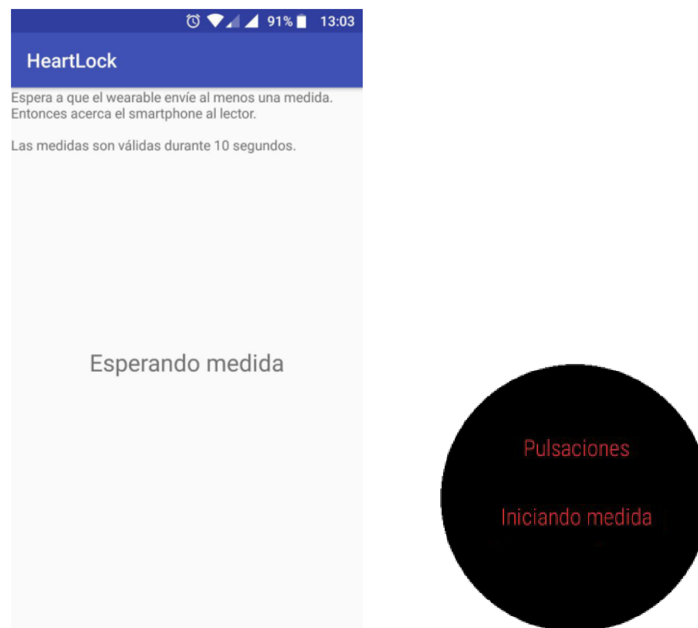

Figura 28. Pantallas mostradas al iniciar las aplicaciones.

En el momento en que el sensor considera que la medida tiene precisión `ALTA` o `MEDIA`, se observa como efectivamente en la pantalla del smartphone aparece la medida en tiempo real junto con la fecha y hora de la adquisición, como se mostraba en las figuras 19 y 20 respectivamente.

Para activar el actuador de acceso (el LED verde integrado en la placa del microcontrolador) la medida de pulsaciones tiene que ser superior a un umbral definido en la aplicación Arduino. Este umbral se ha establecido en 65 ppm (pulsaciones por minuto) para facilitar las pruebas, ya que es una medida que se sitúa en el centro de los valores considerados normales, como se observa en la figura 29. Por ello es sencillo situarse por encima o por debajo y comprobar el funcionamiento del sistema.

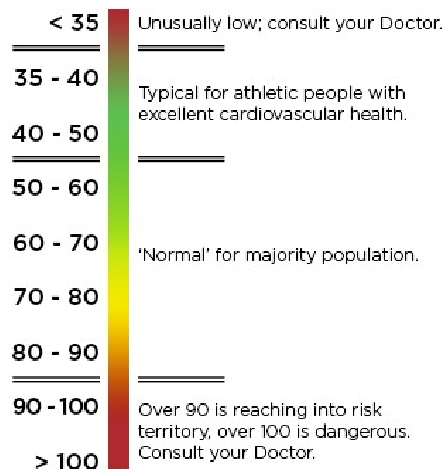

Figura 29. Rangos de frecuencia cardíaca.
Fuente: www.goiheart.com

Lógicamente en una aplicación real como la comentada en el capítulo 3 sobre un alcoholímetro y un control de arranque de vehículo, el umbral serían los 0,25 mg/l en aire espirado que establece la legislación española. Pero a todos los efectos, la arquitectura del sistema es idéntica entre esa aplicación real y el prototipo desarrollado, siendo indiferente el tipo de variable medida y sus unidades.

Cuando se tiene una medida superior al umbral, si se acerca el *smartphone* al módulo NFC se observa como el *smartphone* vibra, el LED del Arduino se enciende 2 segundos, y en la consola aparece un mensaje con el valor y la concesión de acceso. Esto se puede ver en la figura 30.

```
Tarjeta tipo ISO14443A detectada! UID [4 bytes]: 0x8 0x4 0x2E 0xD3

UID de 4 bytes detectado. ¡APDU SEL AID respondido!
Pulsaciones: 080 ppm
* Acceso PERMITIDO *
```

Figura 30. Mensajes en consola del PC tras detección del smartphone y un valor superior al umbral.

Si agitamos la muñeca para que la precisión del sensor se reduzca a `BAJA` y no se envíe medida al *smartphone*, podemos esperar algo más de 10 segundos para que, al acercarlo al lector NFC, éste informe de que la medida enviada está caducada. Esto se muestra en la figura 31.

```
Tarjeta tipo ISO14443A detectada! UID [4 bytes]: 0x8 0x55 0x1B 0x2A

UID de 4 bytes detectado. ¡APDU SEL AID respondido!
* Acceso DENEGADO: Medida caducada *
```

Figura 31. Mensajes en consola del PC tras detección del smartphone y un valor caducado (XXX)

También podemos observar en las figuras que el UID es distinto entre las diferentes aproximaciones del *smartphone* al lector, como era de esperar de un dispositivo Android.

Si acercamos el *smartphone* cuando la medida es inferior al umbral, podemos observar que el LED no se enciende, y que el mensaje en la consola es el esperado (figura 32).

```
Tarjeta tipo ISO14443A detectada! UID [4 bytes]: 0x8 0xD3 0xE1 0xC7

UID de 4 bytes detectado. ¡APDU SEL AID respondido!
Pulsaciones: 063 ppm
* Acceso DENEGADO: Medida por debajo de 65 ppm *
```

Figura 32. Mensajes en consola del PC tras detección y valor inferior al umbral

Si se aproxima al lector un *smartphone* con NFC activado pero que no tiene la aplicación instalada, se puede observar cómo el lector detecta que el *smartphone* no responde a la selección de la AID de nuestra aplicación, como se muestra en la figura 33.

```
Tarjeta tipo ISO14443A detectada! UID [4 bytes]: 0x1 0x2 0x3 0x4

UID de 4 bytes detectado. ¡APDU SEL AID respondido!
[AID diferente al de la aplicacion]
```

Figura 33. Mensajes al acercar un smartphone sin la aplicación instalada

Es curioso observar cómo ese otro *smartphone sin la aplicación instalada*, un Samsung Galaxy J5, envía como UID siempre el valor 0x01020304 en lugar de enviar cada vez un valor aleatorio.

El último caso que quedaría por probar sería un fallo en la comunicación durante la transferencia de los APDUs. Si se acerca el *smartphone* y se aleja rápidamente, se puede lograr esa situación, que es detectada por el Arduino como se muestra en la figura 34.

```
Tarjeta tipo ISO14443A detectada! UID [4 bytes]: 0x8 0x76 0xD4 0x24

UID de 4 bytes detectado. ¡APDU SEL AID respondido!
[Problema en la comunicacion]
```

Figura 34. Mensajes al acercar el smartphone, pero alejarlo inmediatamente

6.3 Coste del sistema

En la siguiente tabla se muestra el coste, a la fecha de realización del Trabajo, de los elementos que han sido necesarios para el desarrollo del prototipo:

Dispositivo	*Coste*
Arduino UNO (microcontrolador)	24 €
Módulo PN532 (NFC Module V3)	15 €
Ticwatch E (*smartwatch*)	160 €
BQ Aquaris M5 (*smartphone*)	150 €
TOTAL	349 €

7. Conclusiones

Este Trabajo se encuadra en el área de los Sistemas de Comunicación, donde lo que se perseguía por su propia definición era el diseño de un sistema de comunicación y el desarrollo de los bloques que lo forman. En ese sentido, se ha cumplido la expectativa a lo largo de su desarrollo, pues se ha implementado un sistema completo que transmite un dato de un extremo (sensor) a otro (microcontrolador) mediante elementos que se comunican utilizando tecnologías varias como Bluetooth, NFC, o bus I^2C.

No obstante, el primer y principal reto que ha supuesto este Trabajo ha sido la elección de tema, ya que encontrar un tema que sea objeto de interés, con utilidad práctica justificada, y que aporte innovación al área, puede llegar a ser complicado.

A pesar de ello, y teniendo en cuenta que el autor del Trabajo ya disponía de conocimientos sobre programación de microcontroladores en C y desarrollo de aplicaciones Android (adquiridos durante el estudio del Grado y Máster) se han cumplido correctamente los objetivos planteados al inicio, que podemos recapitular:

- Familiarizarse con la tecnología NFC y ser capaz de transmitir información mediante la misma: esta tecnología era desconocida para el autor del Trabajo, pero se ha logrado entender sus conceptos básicos, su funcionamiento práctico, y conocer los estándares que se utilizan actualmente.

- Explorar diferentes sistemas de adquisición de medidas biométricas mediante un *smartphone* Android: aunque aquí es donde menos se ha profundizado, se han conocido los dispositivos *wearables* que ejecutan Android, y el tremendo potencial que presentan por la sencillez de integración con los *smartphones*. De hecho, Wear OS, presentado en marzo de 2018, es una tecnología reciente y poco extendida todavía (menos de 4 años de existencia bajo la denominación de Android Wear), por lo que este es uno de los primeros Trabajos donde se aplica.

- Diseñar y crear un prototipo: este objetivo también se ha cumplido, pudiendo adquirir el hardware necesario, aprender su manejo, y lograr un funcionamiento correcto según las especificaciones sobre la arquitectura *hardware* y *software* del prototipo planteadas.

El seguimiento de la planificación ha sido correcto, logrando realizar a tiempo todas las tareas planificadas para cada PEC, aunque el autor del Trabajo hubiera deseado disponer de más tiempo, tanto para mejorar el sistema desarrollado (mejorar la interfaz gráfica, por ejemplo), como para poder reflejar en esta memoria todo el trabajo que ha sido realizado en paralelo.

Dicho trabajo no reflejado en esta memoria se refiere al aprendizaje y desarrollo de pequeños códigos de software que se han ido empleando para familiarizarse con las diferentes tecnologías utilizadas (por ejemplo, estudio de

los diferentes tipos de tarjetas NFC pasivas, y desarrollo de aplicaciones de lectura y escritura de las mismas, así como el estudio del funcionamiento de NFC en Android para la realización de pagos en comercios).

Por todo ello, el autor del Trabajo cree que con éste se presentan multitud de líneas de trabajo futuro y de mejora, como por ejemplo:

- Búsqueda de otras posibles aplicaciones prácticas para el sistema planteado, revisando el estado del arte. Entre ellas el uso en videojuegos.

- Rediseño de la comunicación NFC haciendo uso del modo P2P en lugar del modo de emulación de tarjetas, y justificación del uso de uno u otro.

- Añadir medidas de seguridad para evitar la suplantación de identidad (autenticación) no tratada en este Trabajo, y estudiar las debilidades del sistema planteado en cuanto a ataques de manipulación de la comunicación y la medida enviada (*tampering*). Por ejemplo, analizando en profundidad los estándares de tarjetas criptográficas.

- Mejora de las interfaces gráficas de la aplicación móvil y del *smartwatch*, añadiendo *feedback* hacia el usuario (ahora mismo sólo vibra el móvil cuando envía la medida), e incluso creando perfiles para distintos usuarios que la utilizasen.

8. Glosario

AAA (*Authentication, Authorization, Accounting*): familia de procesos para el control de acceso a algún recurso.

AID (*Application IDentifier*): identificador que se utiliza para identificar a una aplicación en una tarjeta inteligente.

APDU (*Application Protocol Data Unit*): unidad de comunicación entre un lector de tarjetas y una tarjeta inteligente, según ISO/IEC 7816-4.

API (*Application Programming Interface*): conjunto de definiciones de subrutinas, protocolos y herramientas para el desarrollo de aplicaciones *software*.

APK (*Android Package Kit*): formato de fichero utilizado por Android para la distribución e instalación de aplicaciones.

Arduino: plataforma *hardware* y *software* de Código abierto para el prototipado sobre microcontroladores, como por ejemplo Arduino UNO (Atmel ATmega328P)

Bluetooth: tecnología inalámbrica para el intercambio de información sobre distancias cortas.

BYOD (*Bring Your Own Device*): política consistente en permitir a los empleados traer sus dispositivos personales y usarlos en el lugar de trabajo para el acceso a información y aplicaciones de la empresa.

Data Layer [Wearable]: API parte de los servicios de Google Play que proporciona un canal de comunicaciones para las aplicaciones, permitiendo la comunicación entre un teléfono y un *wearable*.

Framework: marco de herramientas que facilitan el desarrollo de software.

HCE (*Host Card Emulation*): Emulación virtual por software de tarjetas inteligentes.

I^{2}C (*Inter-Integrated Circuit*): bus de comunicaciones serie síncrono, multil-maestro, multi-esclavo, utilizado para la conexión de periféricos de baja velocidad a microcontroladores.

LED (*Light-Emitting Diode*): semiconductor que emite luz cuando se activa.

NFC (*Near-Field Communications*): conjunto de protocolos que permiten a dos dispositivos electrónicos establecer una comunicación al aproximarse a corta distancia (4 cm).

P2P (*Peer-to-Peer*): uno de los modos de funcionamiento NFC para el intercambio de información entre dispositivos.

SCL (*Serial Clock Line*): línea de reloj del bus I^2C.

SDA (*Serial Data Line*): línea de datos del bus I^2C.

SDK (*Software Development Kit*): conjunto de herramientas de desarrollo de *software* para la creación de aplicaciones.

Smartcard: tarjeta con circuito integrado (inteligente) que permite la ejecución de cierta lógica programada.

UID (*Unique IDentifier*): identificador único de sólo lectura que poseen los chips NFC a modo de número de serie.

Wear OS: sistema operativo para dispositivos *wearables*, basado en Android, y anteriormente conocido como Android Wear.

Wearable: pequeño dispositivo computador que se lleva debajo, en, o encima de la ropa.

9. Bibliografía

[1] S. Micali, D. Engberg, P. Libin, L. Reyzin, A. Sinelnikov; “Physical access control”; U.S. Patent No. US7353396B2; 02-10-1995.

[2] L. Gommans, J. Vollbrecht, D. Spence; “RFC-2903: Generic AAA Architecture”. DOI 10.17487/RFC2903; 2000.

[3] L. Ellis, J. Saret, P. Weed; "BYOD: From company-issued to employee-owned devices”; McKinsey & Company, Inc.; 2012.

[4] A. Sharma, A. Ambekar, P. Dahale, P. Bendere, P. Halgaonkar; “Comparative Study of NFC, Bluetooth, RFID and IrDA”; International Journal of Computer Engineering and Applications, Vol XI. ISSN 2321-2469. May 2017.

[5] J. Dolcourt. “Vivo phone shows off first in-screen fingerprint scanner”; CNET web. Jan 2018. https://www.cnet.com/news/vivo-first-in-screen-fingerprint-scanner-for-phones-ces-2018/ . Último acceso: 16/03/2018.

[6] T. Coppetti, A. Brauchlin, S. Müggler, et al.; “Accuracy of smartphone apps for heart rate measurement”; European Journal of Preventive Cardiolog; doi: 10.1177/2047487317702044. May 2017.

[7] M. Viñas; “Control de acceso mediante NFC con Arduino”; Trabajo Fin de Grado de Tecnologías de Telecomunicación; UOC. Enero 2017.

[8] A. Arborr. “Thinking of drinking and driving? What if your car won’t let you?”; Michigan University News. Mar 2015. https://news.umich.edu/thinking-of-drinking-and-driving-what-if-your-car-won-t-let-you/ . Último acceso: 25/03/2018.

[9] Wear OS by Google. https://wearos.google.com/ . Último acceso: 10/05/2018.

[10] Arduino. http://www.arduino.cc . Último acceso: 03/05/2018.

[11] Android Developers. https://developer.android.com/ . Último acceso: 18/05/2018.

[12] NFC Forum. https://nfc-forum.org/ . Último acceso: 27/04/2018.

[12] “PN532 User Manual”. NXP Semiconductors. 2007. https://www.nxp.com/docs/en/user-guide/141520.pdf . Último acceso: 18/05/2018.

[13] “ISO/IEC 7816-4: ID cards - IC cards - Part 4: Organization, security and commands for interchange”. ISO/IEC. 2005.

10. Anexos

I. Código Arduino

```
// Codigo Arduino del lector NFC de la aplicacion HeartLock de Android
// TFM UOC - Antonio Ortega 2018
// Contiene funciones de la libreria Adafruit-PN532 disponible en:
// https://github.com/adafruit/Adafruit-PN532

// Liberia de comunicacion I2C
#include <Wire.h>

// Funciones utiles para depuracion por conexion serie con PC
//#define DEBUG
#ifdef DEBUG
#define DMSG(args...)       Serial.print(args)
#define DMSG_HEX(num)       Serial.print(' '); Serial.print((num>>4)&0x0F, HEX);
Serial.print(num&0x0F, HEX)
#define DMSG_INT(num)       Serial.print(' '); Serial.print(num)
#else
#define DMSG(args...)
#define DMSG_HEX(num)
#define DMSG_INT(num)
#endif

// Valores extraidos del manual del PN532
#define PN532_I2C_ADDRESS             (0x48 >> 1)
#define PN532_PREAMBLE                (0x00)
#define PN532_STARTCODE1              (0x00)
#define PN532_STARTCODE2              (0xFF)
#define PN532_POSTAMBLE               (0x00)
#define PN532_HOSTTOPN532             (0xD4)
#define PN532_PN532TOHOST             (0xD5)
#define PN532_ACK_WAIT_TIME           (10)
#define PN532_COMMAND_GETFIRMWAREVERSION    (0x02)
#define PN532_COMMAND_INDATAEXCHANGE        (0x40)
#define PN532_COMMAND_INLISTPASSIVETARGET   (0x4A)
#define PN532_MIFARE_ISO14443A              (0x00) // BrTy 106 kbps ISO14443A
#define PN532_COMMAND_SAMCONFIGURATION      (0x14)
#define PN532_INVALID_ACK             (-1)
#define PN532_TIMEOUT                 (-2)
#define PN532_INVALID_FRAME           (-3)
#define PN532_NO_SPACE                (-4)

// Buffers para almacenar los comandos y los bytes de los mensajes
uint8_t command = 0;
uint8_t pn532_packetbuffer[255];

// Funcion para el envio de un comando al PN532
// Pag. 28 del manual
int8_t writeCommand(const uint8_t *header, uint8_t hlen, const uint8_t *body = 0, uint8_t
blen = 0) {
  command = header[0];
  Wire.beginTransmission(PN532_I2C_ADDRESS);
  Wire.write(PN532_PREAMBLE);                // PREAMBLE (0x00)
  Wire.write(PN532_STARTCODE1);              // START CODE (0x00FF)
  Wire.write(PN532_STARTCODE2);
  uint8_t length = hlen + blen + 1;          // LENGTH (TFI+DATA)
  Wire.write(length);
  Wire.write(~length + 1);                   // LCS (LowB of LEN+LCS = 0x00)
  Wire.write(PN532_HOSTTOPN532);             // TFI (0xD4)
  uint8_t sum = PN532_HOSTTOPN532;
  DMSG("\nEnviado:");
  for (uint8_t i = 0; i < hlen; i++) {
    if (Wire.write(header[i])) {
      sum += header[i];
      DMSG_HEX(header[i]);
    } else {
      DMSG("\nDemasiado para enviar, paquete I2C max 32 bytes\n");
```

```
      return PN532_INVALID_FRAME;
    }
  }
  for (uint8_t i = 0; i < blen; i++) {
    if (Wire.write(body[i])) {
      sum += body[i];
      DMSG_HEX(body[i]);
    } else {
      DMSG("\nDemasiado para enviar, paquete I2C max 32 bytes\n");
      return PN532_INVALID_FRAME;
    }
  }
  uint8_t checksum = ~sum + 1;
  Wire.write(checksum);                          // DCS (LB of TFI+DATA+DCS = 0x00)
  Wire.write(PN532_POSTAMBLE);                   // POSTAMBLE (0x00)
  Wire.endTransmission();
  DMSG('\n');
  return readAckFrame();
}

// Funcion para la comprobacion de la recepcion correcta (ACK)
// Pag. 30 del manual
int8_t readAckFrame() {
  const uint8_t PN532_ACK[] = {0, 0, 0xFF, 0, 0xFF, 0};
  uint8_t ackBuf[sizeof(PN532_ACK)];
  DMSG(" Espera ACK en: "); DMSG(millis()); DMSG('\n');
  uint16_t time = 0;
  do {
    if (Wire.requestFrom(PN532_I2C_ADDRESS, sizeof(PN532_ACK) + 1)) {
      if (Wire.read() & 1) break;   // STATUS - Pag. 42 del manual
    }
    delay(1); time++;
    if (time > PN532_ACK_WAIT_TIME) {
      DMSG("Timeout esperando ACK\n");
      return PN532_TIMEOUT;
    }
  } while (1);
  DMSG(" ACK recibido en: "); DMSG(millis()); DMSG('\n');
  for (uint8_t i = 0; i < sizeof(PN532_ACK); i++) {
    ackBuf[i] = Wire.read();
  }
  if (memcmp(ackBuf, PN532_ACK, sizeof(PN532_ACK))) {
    DMSG("ACK invalido\n");
    return PN532_INVALID_ACK;
  }
  return 0;
}

// Funcion para la obtencion del tamaño de la respuesta del PN532
int16_t getResponseLength(uint8_t buf[], uint8_t len, uint16_t timeout) {
  const uint8_t PN532_NACK[] = {0, 0, 0xFF, 0xFF, 0, 0};
  uint16_t time = 0;
  do {
    if (Wire.requestFrom(PN532_I2C_ADDRESS, 6)) {
      if (Wire.read() & 1) break;   // STATUS - Pag. 42 del manual
    }
    delay(1); time++;
    if ((0 != timeout) && (time > timeout)) return -1;
  } while (1);
  if (0x00 != Wire.read() ||        // PREAMBLE
      0x00 != Wire.read() ||        // START CODE
      0xFF != Wire.read()) {
    return PN532_INVALID_FRAME;
  }
  uint8_t length = Wire.read();
  // Solicitar envio del Frame completo otra vez
  Wire.beginTransmission(PN532_I2C_ADDRESS);
  for (uint16_t i = 0; i < sizeof(PN532_NACK); ++i) {
    Wire.write(PN532_NACK[i]);
  }
  Wire.endTransmission();
  return length;
}
```

```
// Funcion para la lectura de respuestas del PN532
int16_t readResponse(uint8_t buf[], uint8_t len, uint16_t timeout = 1000) {
  uint16_t time = 0;
  uint8_t length;
  length = getResponseLength(buf, len, timeout);
  // [RDY] 00 00 FF LEN LCS (TFI PD0 ... PDn) DCS 00
  do {
    if (Wire.requestFrom(PN532_I2C_ADDRESS, 6 + length + 2)) {
      if (Wire.read() & 1) break;  // STATUS - Pag. 42 del manual
    }
    delay(1); time++;
    if ((0 != timeout) && (time > timeout)) return -1;
  } while (1);
  if (0x00 != Wire.read() ||       // PREAMBLE
      0x00 != Wire.read() ||       // START CODE
      0xFF != Wire.read()) {
    return PN532_INVALID_FRAME;
  }
  length = Wire.read();
  if (0 != (uint8_t)(length + Wire.read())) {
    return PN532_INVALID_FRAME;
  }
  uint8_t cmd = command + 1;
  if (PN532_PN532TOHOST != Wire.read() || (cmd) != Wire.read()) {
    return PN532_INVALID_FRAME;
  }
  length -= 2;
  if (length > len) {
    return PN532_NO_SPACE;
  }
  DMSG("Recibido:"); DMSG_HEX(cmd);
  uint8_t sum = PN532_PN532TOHOST + cmd;
  for (uint8_t i = 0; i < length; i++) {
    buf[i] = Wire.read();
    sum += buf[i];
    DMSG_HEX(buf[i]);
  }
  DMSG('\n');
  uint8_t checksum = Wire.read();
  if (0 != (uint8_t)(sum + checksum)) {
    DMSG("Checksum incorrecto\n");
    return PN532_INVALID_FRAME;
  }
  Wire.read();          // POSTAMBLE
  return length;
}

// Funcion para la deteccion del PN532 y obtencion de version de firmware
// Pag. 73 del manual
uint32_t getFirmwareVersion(void) {
  uint32_t response;
  pn532_packetbuffer[0] = PN532_COMMAND_GETFIRMWAREVERSION;
  if (writeCommand(pn532_packetbuffer, 1)) return 0;
  int16_t status = readResponse(pn532_packetbuffer, sizeof(pn532_packetbuffer));
  if (0 > status) return 0;
  response = pn532_packetbuffer[0]; response <<= 8;   // IC (0x32)
  response |= pn532_packetbuffer[1]; response <<= 8;  // Ver
  response |= pn532_packetbuffer[2]; response <<= 8;  // Rev
  response |= pn532_packetbuffer[3]; return response; // Support (ISO18092, 14443B, 14443A)
}

// Funcion para la configuracion del data flow path (Modo normal sin SAM)
// mediante el comando SAMConfig
bool SAMConfig(void) {
  pn532_packetbuffer[0] = PN532_COMMAND_SAMCONFIGURATION;
  pn532_packetbuffer[1] = 0x01; // Normal mode, SAM not used
  pn532_packetbuffer[2] = 0x14; // timeout 50ms * 20 = 1 second
  pn532_packetbuffer[3] = 0x01; // P70_IRQ pin driven by PN532
  DMSG("SAMConfig\n");
  if (writeCommand(pn532_packetbuffer, 4)) return false;
  return (0 < readResponse(pn532_packetbuffer, sizeof(pn532_packetbuffer)));
}

// Funcion para en envio de un APDU de seleccion de AID
```

```
uint32_t selectApdu(uint8_t *data) {
  pn532_packetbuffer[0] = PN532_COMMAND_INDATAEXCHANGE;
  pn532_packetbuffer[1] = 1;  // Tg, Tarjeta 1
  pn532_packetbuffer[2] = 0x00; // CLAss
  pn532_packetbuffer[3] = 0xA4; // INStruction, SELECT
  pn532_packetbuffer[4] = 0x04; // P1, select by name
  pn532_packetbuffer[5] = 0x00; // P2
  pn532_packetbuffer[6] = 0x07; // lenght of the application name
  memcpy (pn532_packetbuffer + 7, data, 7);
  writeCommand(pn532_packetbuffer, 14);
  return (0 < readResponse(pn532_packetbuffer, sizeof(pn532_packetbuffer)));
}

// Funcion para la deteccion de una tarjeta pasiva
bool readPassiveTargetID(uint8_t cardbaudrate, uint8_t *uid, uint8_t *uidLength, uint16_t
timeout = 1000) {
  pn532_packetbuffer[0] = PN532_COMMAND_INLISTPASSIVETARGET;
  pn532_packetbuffer[1] = 1;  // MaxTg 1 tarjeta (hasta 2)
  pn532_packetbuffer[2] = cardbaudrate;
  if (writeCommand(pn532_packetbuffer, 3)) return 0x0;  // command failed
  if (readResponse(pn532_packetbuffer, sizeof(pn532_packetbuffer), timeout) < 0) return 0x0;
  if (pn532_packetbuffer[0] != 1) return 0;  // Tags found = 1, [1] Tag Number
  uint16_t sens_res = pn532_packetbuffer[2]; // SENS_RES
  DMSG("ATQA:"); DMSG_HEX(sens_res);
  sens_res <<= 8; sens_res |= pn532_packetbuffer[3];
  DMSG_HEX(sens_res);
  DMSG(", SAK:"); DMSG_HEX(pn532_packetbuffer[4]); // SEL_RES
  DMSG("\n");
  *uidLength = pn532_packetbuffer[5];   // NFCID Length
  for (uint8_t i = 0; i < pn532_packetbuffer[5]; i++) {
    uid[i] = pn532_packetbuffer[6 + i]; // NFCID
  }
  return 1;
}

// Funcion que se ejecuta una vez en el arranque del programa
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);  // Actuador: LED integrado en Arduino
  Serial.begin(115200);          // Conexion serie con PC para debug
  Serial.println("Comienzo del programa.");
  Wire.begin(); delay(500); // Wakeup PN532
  uint32_t versionPN532 = getFirmwareVersion();
  if (! versionPN532) {
    // Se detiene el programa si no se detecta un PN532
    Serial.print(F("No se detecta PN532.")); while (1);
  }
  // Configuración del data flow path (Modo normal sin SAM)
  SAMConfig();
  // Debug por serie con PC
  Serial.print(F("Detectado PN5")); Serial.println((versionPN532>>24) & 0xFF, HEX);
  Serial.print(F("Firmware v")); Serial.print((versionPN532>>16) & 0xFF, DEC);
  Serial.print('.'); Serial.println((versionPN532>>8) & 0xFF, DEC);
  Serial.println("Esperando tarjeta...");
}

// Funcion que se ejecuta ciclicamente
void loop() {
  // UMBRAL de pulsaciones para la decision de permitir el acceso o no
  int Umbral = 65;
  boolean lecturaOK; boolean seleccionOK;
  uint8_t uidLength; uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer para el UID
  // AID de la aplicacion Android
  uint8_t AID[7] = { 0xF0, 0x39, 0x41, 0x48, 0x14, 0x81, 0x00 };

  // Deteccion de tarjeta pasiva de tipo ISO14443A
  lecturaOK = readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);
  if (lecturaOK) {
    // Debug por serie en PC de la deteccion
    Serial.print("\nTarjeta tipo ISO14443A detectada! ");
    Serial.print("UID ["); Serial.print(uidLength, DEC); Serial.print(" bytes]:");
    for (uint8_t i=0; i < uidLength; i++) {
      Serial.print(" 0x"); Serial.print(uid[i], HEX);
    }
    Serial.println("\n");
```

```
    // Si la longitud del UID es 4 bytes, es una tarjeta correcta
    if (uidLength == 4) {
      Serial.print("UID de 4 bytes detectado. ");
      // Envio de APDU con comando de seleccion de AID, con el AID de la aplicacion Android
      if (selectApdu(AID)) {
        // Si la aplicacion Android responde al APDU y envia un valor
        Serial.println("¡APDU SEL AID respondido!");
        // Si el primer byte de la respuesta (STATUS) no es 0, ha ocurrido algun fallo en la
comunicacion
        if (pn532_packetbuffer[0]==0x00) {
          // Si la respuesta al SELECT AID es 0x6A82, no se ha encontrado la aplicacion
          // correspondiente al AID enviado. La repuesta debe acabar con 0x9000.
          // La medida enviada por el smartphone son 3 bytes.
          if ( ((pn532_packetbuffer[1]!=0x6A) && (pn532_packetbuffer[2]!=0x82)) ||
               ((pn532_packetbuffer[4]!=0x90) && (pn532_packetbuffer[5]!=0x00)) ) {
            // Si la aplicacion envia como valor "XXX", la medida esta caducada
            // Solo es necesario comprobar que el primer byte no sea 'X' (0x58 ASCII)
            if (pn532_packetbuffer[1]!=0x58) {
           // Se genera la cadena de caracteres pulso con el valor en pulsaciones por minuto,
           // y se almacena el valor en una variable tipo entero. Las cadenas terminan con
caracter NULO
              char pulso[4];
              pulso[0] = pn532_packetbuffer[1]; pulso[1] = pn532_packetbuffer[2];
              pulso[2] = pn532_packetbuffer[3]; pulso[3] = '\0';
              int pulsaciones; pulsaciones = atoi(pulso);
              Serial.print("Pulsaciones: "); Serial.print(pulso); Serial.println(" ppm");
              // Comparacion del valor de pulso con respecto al Umbral,
              // en este caso las pulsaciones deben ser superiores al umbral para permitir el
acceso
              if (pulsaciones > Umbral) {
                // Acceso concedido.
                // Activacion del LED durante 2 segundos
                Serial.println("* Acceso PERMITIDO *");
                digitalWrite(LED_BUILTIN, HIGH); delay(2000); digitalWrite(LED_BUILTIN, LOW);
              } else {
                Serial.print("* Acceso DENEGADO: Medida por debajo de ");
Serial.print(Umbral);
                Serial.println(" ppm *");
              }
            } else {
              Serial.println("* Acceso DENEGADO: Medida caducada *");
            }
          } else {
            Serial.println("[AID diferente al de la aplicacion]");
          }
        } else {
          Serial.println("[Problema en la comunicacion]");
        }
      }
    // Pausamos 2 segundos entre lecturas consecutivas
    delay(2000);
    }
  } else {
      // El PN532 no ha detectado ninguna tarjeta durante 1 segundo
      DMSG("Timeout esperando tarjeta\n");
    }
}
```

II. Código Android Wearable

II.I. Manifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.uoctfm.heartlock">
    <uses-feature android:name="android.hardware.type.watch" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.BODY_SENSORS" />
    <application android:allowBackup="true" android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name" android:supportsRtl="true"
        android:theme="@android:style/Theme.DeviceDefault">
        <uses-library android:name="com.google.android.wearable" android:required="true" />
        <meta-data android:name="com.google.android.wearable.standalone"
            android:value="true" />
        <activity android:name=".MainActivity" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".DataLayerListenerService" android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="com.google.android.gms.wearable.MESSAGE_RECEIVED" />
                <data android:host="*" android:pathPrefix="/start/MainActivity"
                    android:scheme="wear" />
            </intent-filter>
        </service>
    </application>
</manifest>
```

II.II. Clase: DataLayer Service

```
package com.uoctfm.heartlock;
import android.app.Service; import android.content.Intent; import android.os.IBinder;
import com.google.android.gms.wearable.MessageEvent;
import com.google.android.gms.wearable.WearableListenerService;

// Este servicio recibe mensajes del smartphone, y si el path es el adecuado,
// lanza la actividad principal de la aplicación
public class DataLayerListenerService extends WearableListenerService {
    public static final String START_ACTIVITY_PATH = "/start/MainActivity";
    @Override
    public void onMessageReceived(MessageEvent messageEvent) {
        super.onMessageReceived(messageEvent);
        if (messageEvent.getPath().equals(START_ACTIVITY_PATH)) {
            Intent intent = new Intent(this, MainActivity.class);
            // El flag SINGLE_TOP previene que se abra una nueva instancia de la app
            // si ya estaba en ejecución
            intent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
            startActivity(intent);
} } }
```

II.III. Clase: Main Activity

```
package com.uoctfm.heartlock;
import android.content.Context; import android.hardware.Sensor;
import android.hardware.SensorEvent; import android.hardware.SensorEventListener;
import android.hardware.SensorManager; import android.os.Bundle;
import android.support.wearable.activity.WearableActivity;
import android.view.WindowManager; import android.widget.TextView;
import com.google.android.gms.tasks.Task; import com.google.android.gms.tasks.Tasks;
import com.google.android.gms.wearable.Node;
import com.google.android.gms.wearable.Wearable; import java.util.List;
import java.util.concurrent.ExecutionException;
```

```java
public class MainActivity extends WearableActivity implements SensorEventListener {
    private SensorManager mSensorManager;
    private Sensor mHeart;
    private TextView pulso, precisionQuieto;
    private int accuracy = SensorManager.SENSOR_STATUS_ACCURACY_LOW;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        pulso = (TextView) findViewById(R.id.text);
        precisionQuieto = (TextView) findViewById(R.id.text2);
        // Pantalla siempre encendida
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        // Instancia del sensor de RITMO_CARDIACO.
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mHeart = mSensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE);
    }

    // El sensor presenta 3 tipos de precisión: LOW, MEDIUM, HIGH
    // Si la precisión no es HIGH, instamos al usuario a que permanezca quieto
    // para que la precisión se incremente
    @Override
    public final void onAccuracyChanged(Sensor sensor, int acc) {
        accuracy = acc;
        String msg = "";
        switch(accuracy) {
            case SensorManager.SENSOR_STATUS_ACCURACY_HIGH:
                msg="";
                break;
            case SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM:
                msg="¡Mantente quieto!";
                break;
            case SensorManager.SENSOR_STATUS_ACCURACY_LOW:
                msg="¡Mantente quieto!";
                break;
        }
        precisionQuieto.setText(msg);
    }
    // Método que se invoca cuando el valor del sensor cambia
    @Override
    public final void onSensorChanged(SensorEvent event) {
        // Lectura del valor
        float heart_rate_float = event.values[0];
        int heart_rate = Math.round(heart_rate_float);
        pulso.setText(Integer.toString(heart_rate)+" ppm");

        String datapath ="/pulsaciones";
       // Se envía el valor del sensor al smatphone si la precisión de la medida es HIGH
        if (accuracy!=SensorManager.SENSOR_STATUS_ACCURACY_LOW) {
            new SendMessage(datapath, Integer.toString(heart_rate)).start();
        }
    }
    @Override
    protected void onResume() {
        // Registro del Listener para el sensor
        super.onResume();
        mSensorManager.registerListener(this, mHeart, SensorManager.SENSOR_DELAY_NORMAL);
    }
    @Override
    protected void onPause() {
        // De-registro del Listener del sensor
        super.onPause();
        mSensorManager.unregisterListener(this);
    }
    // Thread para el envío del valor de pulsaciones al smartphone
    class SendMessage extends Thread {
        String path;
        String message;
        // Envío de información a la Data Layer
        SendMessage(String p, String m) {
            path = p;
            message = m;
        }
        public void run() {
```

```java
            // Obtención de los nodos (smartphones) conectados
            Task<List<Node>> nodeListTask =
                    Wearable.getNodeClient(getApplicationContext()).getConnectedNodes();
            try {
                List<Node> nodes = Tasks.await(nodeListTask);
                for (Node node : nodes) {
                    Task<Integer> sendMessageTask =
Wearable.getMessageClient(MainActivity.this).sendMessage(node.getId(), path, message.getBytes());
                    try {
                        // Envío del mensaje
                        Integer result = Tasks.await(sendMessageTask);
                    } catch (ExecutionException exception) {
                    } catch (InterruptedException exception) {
                    }
                }
            } catch (ExecutionException exception) {
            } catch (InterruptedException exception) {
} } } }
```

II.IV. Layout: Main Activity

```xml
<android.support.wear.widget.BoxInsetLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:background="@color/black"
    android:padding="@dimen/box_inset_layout_padding"
    tools:context=".MainActivity" tools:deviceIds="wear">
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="@dimen/box_inset_layout_padding"
        app:boxedEdges="all">
        <TextView
            android:id="@+id/text"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="@string/pulsaciones"
            android:textColor="@color/dark_red"
            android:textSize="18sp" />
        <TextView
            android:id="@+id/text2"
            android:layout_width="match_parent"
            android:layout_height="80dp"
            android:gravity="bottom|center"
            android:text="@string/iniciando"
            android:textColor="@color/dark_red"
            android:textSize="18sp" />
    </FrameLayout>
</android.support.wear.widget.BoxInsetLayout>
```

III. Código Android Smartphone

III.I. Manifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.uoctfm.heartlock">
    <uses-feature android:name="android.hardware.nfc.hce" android:required="true" />
    <uses-permission android:name="android.permission.NFC" />
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <application android:allowBackup="true" android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name" android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true" android:theme="@style/AppTheme">
        <activity android:name=".MainActivity" android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".MessageService" android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="com.google.android.gms.wearable.MESSAGE_RECEIVED" />
                <data android:host="*" android:pathPrefix="/pulsaciones"
                    android:scheme="wear" />
            </intent-filter>
        </service>
        <service android:name=".HApduService" android:enabled="true"
            android:exported="true"
            android:permission="android.permission.BIND_NFC_SERVICE">
            <intent-filter>
                <action android:name="android.nfc.cardemulation.action.HOST_APDU_SERVICE" />
            </intent-filter>
            <meta-data android:name="android.nfc.cardemulation.host_apdu_service"
                android:resource="@xml/aid_list" />
        </service>
    </application>
</manifest>
```

III.II. AID List

```
<!-- En este fichero se definen los AIDs que la aplicacion emula.
Los AIDs de tipo Vendor-specific deben comenzar con un byte "F", de acuerdo a la especificacion
ISO 7816. Se recomienda que los AIDs sean de 6 bytes de longitud (max 16 bytes). -->
<host-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/service_name"
    android:requireDeviceUnlock="false">
    <aid-group
        android:category="other"
        android:description="@string/card_title">
        <aid-filter android:name="F0394148148100" />
    </aid-group>
</host-apdu-service>
```

III.III. Clase: Message Service (Data Layer)

```
package com.uoctfm.heartlock;
import android.content.Intent;
import android.support.v4.content.LocalBroadcastManager;
import com.google.android.gms.wearable.MessageEvent;
import com.google.android.gms.wearable.WearableListenerService;

// Este servicio recibe mensajes del wearable, y si el path es el adecuado,
// retransmite el mensaje localmente (en el smartphone)
public class MessageService extends WearableListenerService {
    @Override
    public void onMessageReceived(MessageEvent messageEvent) {
```

```
        if (messageEvent.getPath().equals("/pulsaciones")) {
            final String message = new String(messageEvent.getData());
            Intent messageIntent = new Intent();
            messageIntent.setAction(Intent.ACTION_SEND);
            messageIntent.putExtra("message", message);
            //Broadcast local del mensaje recibido por Data Layer
            LocalBroadcastManager.getInstance(this).sendBroadcast(messageIntent);
        } else {
            super.onMessageReceived(messageEvent);
} } }
```

III.IV. Clase: Main Activity

```
package com.uoctfm.heartlock;
import android.content.BroadcastReceiver; import android.content.ComponentName;
import android.content.Context; import android.content.Intent;
import android.content.IntentFilter; import android.icu.text.DateFormat;
import android.nfc.NfcAdapter; import android.nfc.cardemulation.CardEmulation;
import android.os.Bundle; import android.support.v4.content.LocalBroadcastManager;
import android.support.v7.app.AppCompatActivity; import android.view.WindowManager;
import android.widget.TextView; import android.widget.Toast;
import com.google.android.gms.tasks.Task; import com.google.android.gms.tasks.Tasks;
import com.google.android.gms.wearable.Node;
import com.google.android.gms.wearable.Wearable; import java.util.Date;
import java.util.List; import java.util.concurrent.ExecutionException;

public class MainActivity extends AppCompatActivity {
    TextView textview, textview2;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textview = findViewById(R.id.textView);
        textview2 = findViewById(R.id.textView2);
        // Pantalla siempre encendida
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        // Registro de un receptor de mensajes
        IntentFilter messageFilter = new IntentFilter(Intent.ACTION_SEND);
        Receiver messageReceiver = new Receiver();
     LocalBroadcastManager.getInstance(this).registerReceiver(messageReceiver, messageFilter);
        // Envío de mensaje al wearable para que abra la aplicación
        new SendMessage("/start/MainActivity", "").start();
    }
    @Override
    protected void onResume() {
ComponentName hceComponentName = new ComponentName(getApplicationContext(), HApduService.class);
CardEmulation.getInstance(NfcAdapter.getDefaultAdapter(getApplicationContext())).setPreferredSe
rvice(MainActivity.this, hceComponentName);
        super.onResume();
    }
    @Override
    protected void onPause() {
CardEmulation.getInstance(NfcAdapter.getDefaultAdapter(getApplicationContext())).unsetPreferred
Service(MainActivity.this);
        super.onPause();
    }
    // Receptor de mensajes Broadcast
    public class Receiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            // Recepción de pulsaciones desde el wearable
            String message = intent.getStringExtra("message");
            Date date = new Date();
            Long milliseconds = date.getTime();
         String currentDateTimeString = DateFormat.getDateTimeInstance().format(milliseconds);
            // Almacenamiento de pulsaciones y fecha de la medida
            AlmacenAutorizacion.SetData(MainActivity.this, message, milliseconds);
            textview.setText(message + " ppm");
            textview2.setText(currentDateTimeString);
        }
    }
```

```
    class SendMessage extends Thread {
        String path;
        String message;
        // Envío de información a la Data Layer
        SendMessage(String p, String m) {
            path = p;
            message = m;
        }
        public void run() {
            // Obtención de los nodos (smartphones) conectados
            Task<List<Node>> nodeListTask =
                    Wearable.getNodeClient(getApplicationContext()).getConnectedNodes();
            try {
                List<Node> nodes = Tasks.await(nodeListTask);
                for (Node node : nodes) {
                    Task<Integer> sendMessageTask =
Wearable.getMessageClient(MainActivity.this).sendMessage(node.getId(), path,
message.getBytes());
                    try {
                        // Envío del mensaje
                        Integer result = Tasks.await(sendMessageTask);
                    } catch (ExecutionException exception) {
                    } catch (InterruptedException exception) {
                    }
                }
            } catch (ExecutionException exception) {
            } catch (InterruptedException exception) {
} } } }
```

III.V. Clase: Almacen Autorización

```
package com.uoctfm.heartlock;
import android.content.Context; import android.content.SharedPreferences;
import android.preference.PreferenceManager;

public class AlmacenAutorizacion {
    private static String sPulso = null;
    private static Long sFecha = null;
    private static final Object sLock = new Object();
    public static void SetData(Context c, String pulso, Long fecha) {
        synchronized (sLock) {
            SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(c);
            prefs.edit().putString("PULSO", pulso).commit();
            prefs.edit().putLong("FECHA", fecha).commit();
            sPulso = pulso;
            sFecha = fecha;
        }
    }
    public static String GetPulso(Context c) {
        synchronized (sLock) {
            if (sPulso == null) {
                SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(c);
                String pulso = prefs.getString("PULSO", "0");
                sPulso = pulso;
            }
            return sPulso;
        }
    }
    public static Long GetFecha(Context c) {
        synchronized (sLock) {
            if (sFecha == null) {
                SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(c);
                Long fecha = prefs.getLong("FECHA",0);
                sFecha = fecha;
            }
            return sFecha;
        }
} }
```

III.VI. Clase: HostAPDU Service

```
package com.uoctfm.heartlock;
import android.content.Context; import android.icu.text.DateFormat;
import android.nfc.cardemulation.HostApduService; import android.os.Bundle;
import android.os.VibrationEffect; import android.os.Vibrator; import android.widget.Toast;
import java.util.Arrays; import java.util.Date;

public class HApduService extends HostApduService {
    private static final String TFM_AID = "F0394148148100";
    // Cabecera del comando ISO-DEP HEADER para seleccionar un AID
    // Formato: [Clase | Instruccion | Parametro 1 | Parametro 2]
    private static final String SELECT_APDU_HEADER = "00A40400";
    // Estado "OK" enviado como respuesta al comand SELECT AID 0x9000)
    private static final byte[] SELECT_OK_SW = HexStringToByteArray("9000");
    // Estado "DESCONOCIDO" enviado como respuesta a un comando de APDU invalido (0x0000)
    private static final byte[] UNKNOWN_CMD_SW = HexStringToByteArray("0000");
    private static final byte[] SELECT_APDU = BuildSelectApdu(TFM_AID);
    // Método que se invoca al interrumpirse la comunicación NFC
    @Override
    public void onDeactivated(int reason) {    }
    // Método que se invoca al recibir un Comando APDU
    @Override
    public byte[] processCommandApdu(byte[] commandApdu, Bundle bundle) {
        if (Arrays.equals(SELECT_APDU, commandApdu)) {
            Date date = new Date();
            Long actual = date.getTime();
            Long fecha = AlmacenAutorizacion.GetFecha(this)+10000;
            byte[] autBytes;
            if (fecha > actual) {
                String autData = AlmacenAutorizacion.GetPulso(this);
                if (autData.length()<4) {autData="0"+autData;}
                autBytes = autData.getBytes();
                Vibrator v = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
                v.vibrate(200);
            } else {
                String autData = "XXX";
                autBytes = autData.getBytes();
                Toast.makeText(this,"Medida caducada", Toast.LENGTH_SHORT).show();
            }
            return ConcatArrays(autBytes, SELECT_OK_SW);
        } else {
            return UNKNOWN_CMD_SW;
        }
    }
    // Método para construir un APDU
    public static byte[] BuildSelectApdu(String aid) {
        // Formato: [CLASE | INSTRUCCION | PARAMETRO 1 | PARAMETRO 2 | LONGITUD | DATOS]
        return HexStringToByteArray(SELECT_APDU_HEADER + String.format("%02X",
                aid.length() / 2) + aid);
    }
    /* Metodos para el manejo de cadenas de bytes hexadecimales*/
    public static byte[] HexStringToByteArray(String s) throws IllegalArgumentException {
        int len = s.length();
        if (len % 2 == 1) {
            throw new IllegalArgumentException("La cadena Hex debe tener un numero par de
caracteres");
        }
        byte[] data = new byte[len / 2];
        for (int i = 0; i < len; i += 2) {
            data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
                    + Character.digit(s.charAt(i+1), 16));
        }
        return data;
    }
    public static String ByteArrayToHexString(byte[] bytes) {
        final char[] hexArray =
{'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
        char[] hexChars = new char[bytes.length * 2];
        int v;
        for (int j = 0; j < bytes.length; j++) {
            v = bytes[j] & 0xFF;
            hexChars[j * 2] = hexArray[v >>> 4];
```

```
            hexChars[j * 2 + 1] = hexArray[v & 0x0F];
        }
        return new String(hexChars);
    }
    public static byte[] ConcatArrays(byte[] first, byte[]... rest) {
        int totalLength = first.length;
        for (byte[] array : rest) {
            totalLength += array.length;
        }
        byte[] result = Arrays.copyOf(first, totalLength);
        int offset = first.length;
        for (byte[] array : rest) {
            System.arraycopy(array, 0, result, offset, array.length);
            offset += array.length;
        }
        return result;
    }
}
```

III.VI. Layout: Main Activity

```
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Espera a que el wearable envíe al menos una medida.\nEntonces acerca el
smartphone al lector.\n\nLas medidas son válidas durante 10 segundos."
        tools:layout_editor_absoluteX="27dp"
        tools:layout_editor_absoluteY="29dp" />
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Esperando medida"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.628" />
</android.support.constraint.ConstraintLayout>
```