Universidad
de La Laguna

Escuela Superior de
Ingeniería y Tecnología
Sección de Ingeniería Informática

Sistema Inteligente para la Toma de Pulsaciones Cardíacas de Forma Pasiva mediante Android Wear

Intelligence System to Measurement Heart Rate with a passive method by means of Android Wear.

Moisés Roberto Lodeiro Santiago

La Laguna, 03 de junio de 2015

D. **José Luis Roda García**, con N.I.F. 43356123-L profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas Nombre del Departamento de la Universidad de La Laguna, como tutor.

D. **Francisco Martín Fernández**, con N.I.F. 78.629.638-K FPI del Ministerio de Educación con venia docendi adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor.

C E R T I F I C A N

Que la presente memoria titulada:

"Sistema Inteligente para la Toma de Pulsaciones Cardíacas de Forma Pasiva mediante Android Wear"

ha sido realizada bajo su dirección por D. **Moisés Roberto Lodeiro Santiago**, con N.I.F. 78729188G.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a *03 de junio de 2015*

Agradecimientos

En primer lugar deseo expresar mi agradecimiento a los que han sido mis tutores durante estos últimos meses que ponen la guinda final a lo que han sido siete –no tan largos– años de vida universitaria, José Luís Roda García y Francisco Martín Fernández por haberme otorgado la oportunidad de ser parte de un equipo con los cuales he compartido proyectos e ilusiones durante este período (con los que espero continuar en un futuro) y que con sus ideas, apoyo y continuo seguimiento han logrado que a día de hoy esta memoria y lo que ello conlleva, sea posible.

A mis maestros que nunca desistieron en su labor de enseñanza aun cuando durante algunas clases mi cabeza estaba más allá de los cuatro muros que delimitaban la clase.

A muchos compañeros, amigos, de carrera a quienes también he de agradecer su apoyo constante, su inestimable ayuda durante interminables horas de clase, prácticas, reuniones y otros eventos. En especial a Ardiel G.R. por todo lo mencionado anteriormente y por estar ahí cuando siempre lo he necesitado.

Cómo no, a mis padres pilar básico en mi formación. Nunca han dejado que me rinda y siempre han estado ahí para lo que hiciera falta.

A mi hermana Laura, ella sola se merece un párrafo en esta página. Siempre ha estado ahí dando buenos consejos (en las buenas y las malas), apoyando desde un principio mi carrera y dándome un empujón en algunos momentos difíciles.

A mi novia Myriam G.R por darme fuerzas para continuar cuando parecía que me iba a rendir, por todas las horas que le he quitado a nuestra relación por quedarme en casa programando o peleándome con mi nula relación con la literatura para rellenar hojas y hojas de contenido en una memoria.

Licencia

Usted es libre de:

- **Compartir** — copiar y redistribuir el material en cualquier medio o formato
- **Adaptar** — remezclar, transformar y crear a partir del material para cualquier finalidad, excepto comercial.El licenciador no puede revocar estas libertades mientras cumpla con los términos de la licencia.

Bajo las condiciones siguientes:

- **Reconocimiento** — Debe **reconocer adecuadamente** la autoría, proporcionar un enlace a la licencia e **indicar si se han realizado cambios**. Puede hacerlo de cualquier manera razonable, pero no de una manera que sugiera que tiene el apoyo del licenciador o lo recibe por el uso que hace.
- **No hay restricciones adicionales** — No puede aplicar términos legales o **medidas tecnológicas** que legalmente restrinjan realizar aquello que la licencia permite.

Avisos:

- No tiene que cumplir con la licencia aquellos elementos del material en el dominio público o cuando su utilización esté permitida por la aplicación de **una excepción o un límite**.
- No se dan garantías. La licencia puede no ofrecer todos los permisos necesarios para la utilización prevista. Por ejemplo, otros derechos como los de **publicidad, privacidad, o los derechos morales** pueden limitar el uso del material.

Resumen

El objetivo de esta memoria es presentar el trabajo realizado de investigación, aprendizaje y desarrollo con dispositivos wearables bajo la plataforma Android en una versión 4.2 o superiores. En especial se ha hecho hincapié en las metodologías de trabajo así como en las tecnologías usadas, problemas encontrados y tomas de decisiones a las que me he enfrentado durante el desarrollo en estas nuevas tecnologías usadas para el desarrollo de una aplicación orientada al campo de la salud que se encarga de realizar mediciones de forma pasiva sin dar conocimiento al usuario sobre lo mismo y pensada para después de la monitorización, ofrecer un informe sobre el estado del usuario paciente.

Palabras clave: Android, Wearable, Reloj, Cardiograma.

Abstract

This report contains the data and experiences about my project "*Intelligence System to Measurement Heart Rate with a passive method by means of Android Wear*". Wearables are devices that integrates technology that we can carry in our bodies. I used the sensors who provides the wearable clock to measure the heart rate frequency and then, generate health reports in the handheld client. After that, you can see the results in a webpage or in the handheld client. I've being working with android versions 4.2 and highs ones. I've emphasized particularly in working methods and in the used technologies, problems encountered and decision-making to which I have faced during development.

Keywords: Android, Wearable, Clock, Cardiogram.

Índice

Definiciones

Estas definiciones han sido extraídas de fuentes como ***rae.es*** y ***wikipedia*** y han sido completadas en ocasiones con definiciones aclaratorias del uso por un servidor.

- **Cliente**: “Persona que utiliza con asiduidad los servicios de un profesional o empresa.” En nuestro caso, lo usamos como dispositivo al servicio del usuario.
- **Framework**: La palabra inglesa "**Framework**" (marco de trabajo) define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. En el desarrollo de *software*, un ***framework*** o ***infraestructura digital***, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de *software* concretos, que puede servir de base para la organización y desarrollo de *software*.
- **ORM**: El **mapeo objeto-relacional** (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia.
- **ADA**: Android Data Abstraction Framework es una herramienta que nos permite transformar nuestras clases de java, bajo unos criterios estructurales, para transformarlas en un ORM.
- **HandHeld**: El término ***handheld***, ***hand-held computer*** o ***hand-held device***, es un anglicismo que traducido al español significa “de mano” (computadora o dispositivo de mano) y describe al tipo de computadora portátil que se puede llevar en una mano mientras se utiliza.
- **JSON**: Acrónimo de ***JavaScript Object Notation***, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.
- **Wearable**: Dentro del sector tecnológico y más concretamente de la electrónica de consumo un **wearable** o **dispositivo wearable** es aquel dispositivo que se lleva sobre, debajo o incluido en la ropa y que está siempre encendido, no necesita encenderse y apagarse. Otras de sus características es que permite la multitarea

por lo que no requiere dejar de hacer otra cosa para ser usado y puede actuar como extensión del cuerpo o mente del usuario.

Smartwatch: Un **reloj inteligente** o pulsera inteligente, es un reloj de pulsera revolucionario, que ofrece funciones ampliamente mejoradas a las de un reloj de pulsera habitual, a menudo estas funciones son comparables con las de un PDA. Los primeros modelos de relojes inteligentes eran capaces de realizar funciones básicas como cálculos, traducciones o ejecutar mini juegos, pero los actuales relojes inteligentes ya son capaces de desempeñar funciones mejoradas, como las de un teléfono inteligente o incluso un ordenador portátil. Muchos relojes inteligentes pueden ejecutar aplicaciones móviles, algunos otros se ejecutan en un sistema operativo de teléfonos inteligentes para controlarlos, y unos pocos ya tienen las capacidades técnicas de un teléfono móvil. Estos dispositivos pueden incluir características como un acelerómetro, termómetro, altímetro, barómetro, brújula, cronógrafo, equipo de buceo, calculadora, teléfono móvil, GPS, pantalla gráfica, altavoz, agenda, reloj, etc. Puede utilizar auricular inalámbrico, manos libres, micrófono, módem u otro dispositivo externo.

SQLInjection: Ataque por el cual, mediante consultas SQL se consigue alterar el resultado principalmente programado para el usuario.

XSS: Cross-Site-Script consiste en incluir código JavasScript para alterar tanto visualmente como el comportamiento de una página (procesamiento de cookies, sesiones, etc.)

Motivación y objetivos

El principal motivo por el cual me embarqué en un proyecto que podría tener dimensiones bastante considerables como las que dispone el presente, fueron las ganas de realizar algo nuevo, el superar un reto y enfrentarme a algo desconocido hasta el momento por mí, algo que fuera innovador y tratar alguna tecnología nueva como pudieran ser los relojes smartwatch. Los wearables empezaron su auge no hace más de dos años con distintos tipos de dispositivos, desde relojes, camisetas, calzado (todo esto claramente con dispositivos electrónicos integrados que nos facilitaran y ayudaran en el día a día).

Digo novedoso pues es un terreno aún desconocido y que poco a poco ve la luz en la sociedad, lo que supondría un reto para poder trabajar con ello. Aparte de todo eso, otra de las motivaciones es el poder hacer algo que realmente sirva de ayuda. Cuando empecé a tratar el proyecto con mis tutores vimos un gran potencial en los datos que se podrían obtener del reloj. Lo adquirimos principalmente por su relación calidad precio y por su sensor de frecuencia cardíaca. Tras muchas ideas descartadas llegamos a la conclusión de que estaría bien realizar una aplicación que, aprovechando el rendimiento de estos sensores, pudiera realizar un seguimiento de un paciente sincronizando a su vez los datos con un dispositivo que pudiera mostrar de un modo gráfico el rendimiento cardíaco. Esto permitiría con un gran volumen de datos el liberarlos y obtener estadísticas que pudieran servir de ayudar para futuros estudios de la población.

Al principio, lo que podía parecer un mero trabajo de desarrollar una aplicación, se convirtió poco a poco en un trabajo de investigación, dejando a un lado el desarrollo. Una vez superada la fase de estudio e investigación, se desarrolló la aplicación para el wearable, dispositivo handheld y la versión web que son presentadas en los siguientes apartados de éste documento. Durante estos procesos se han estudiado el cómo funcionan los dispositivos de forma general (pasando por interconexiones, diseños y utilidad de los datos hasta la liberación de los datos).

Planteamiento del Problema

El principal objetivo era realizar una aplicación intuitiva de usar y que fuera compatible con los dispositivos Android. Esta aplicación debería ser capaz de realizar un seguimiento del ritmo cardíaco durante el día a día y esto sirviera para mostrar el mismo en dispositivos con el fin de poder liberar datos para el uso anónimo de los mismos con fines médicos.

Alcance del sistema a desarrollar

El primer alcance del sistema era desarrollar un proyecto llamado ***aHeartRate*** que se componía a su vez de dos sub-proyectos, ***Wear*** y ***Mobile***. Wear era el proyecto realizado en java-android para versiones iguales o superiores a la 4.2 con una interfaz gráfica nula (en principio) pues se activaría desde el inicio del reloj y se mantendría ejecutándose en segundo plano conectado con el dispositivo handheld. Para llevar a cabo este proyecto, la interconexión se debía hacer con un dispositivo real y físico pues, el emulador, como se planteó en un primer momento no dispone de sensores ni simuladores de los mismos para generar datos y simular la recogida de los mismos para un wearable. Al disponer de poco espacio, los datos no estarían más tiempo del necesario dentro del wearable.

Por otro lado teníamos el cliente para un dispositivo móvil que recibiría los datos enviados por el wearable y los procesaría en su base de datos interna para que finalmente queden reflejados a modo de gráfica. También se permitiría el envío de datos anónimos para una visualización de los mismos en una página web. De momento el sistema sería mono-usuario, lo que implica que los datos reflejados no serán reflejados indicando siquiera un identificador de usuario anónimo sino que sería una plataforma que mostraría los datos únicamente del dispositivo que se le indique.

Análisis del Estado del Arte

Ilustración 1 – El valor en alza de los wearables

No cabe duda que cada día la tecnología nos aborda, hasta tal punto que han llegado a formar parte de nuestro ser, incluso de sustituir a dispositivos analógicos de uso cotidiano como podía ser el teléfono fijo en casa, remplazado en la actualidad por modernos teléfonos móviles smartphones.

A los "nuevos" dispositivos que están relacionados con alguna parte de nuestro cuerpo y que ya forman parte de nuestra vida cotidiana se denominan "wearables".

Aunque fundamentalmente nuestro proyecto se ha basado en los dispositivos wearables (enfocados principalmente a los relojes), hay otras alternativas basadas, por ejemplo, en Arduino y otras placas de hardware abierto. Se pueden ver referencias a algunos ejemplos de estos sistemas en los apartados [18], [19] y [20] de la bibliografía.

¿Es sólo una tendencia de moda o va a ser una nueva revolución móvil?

Los dispositivos *wearables* empiezan a verse poco a poco en algunos de nuestros usos diarios: En el trabajo, haciendo deporte o jugando, en los centros de salud… Esta relación se establece a través de diferentes sensores y forma de comunicación entre los dispositivos y nuestro cuerpo. Podemos destacar actualmente los relojes inteligentes, el calzado deportivo, pulseras, etc. La idea básica es disponer de información de nuestro cuerpo de igual forma que lo hacemos ya de diferentes electrodomésticos, coches y demás.

Ya desde los años 70 comienzan a definirse los primeros sistemas relacionados con captar información de nuestro cuerpo. **El primer wearable moderno**, según el concepto que tenemos en la actualidad, se le atribuye al profesor de la Universidad de Toronto **Steve Mann** a final de la década de 1970. Sin embargo no fue hasta la época de los 80 cuando Hewlet Packard o Casio lanzaron sus **relojes calculadora** que todos conocemos y que se convirtieron en el **primer wearable en llegar al mercado de masas.** No es hasta 2010 cuando empiezan a aparecer los primeros desarrollos basados en wearables. El año 2014 puede considerarse el año clave para el despegue de estos sistemas debido a que las marcas tecnológicas más conocidas han comenzado ya su comercialización. La Plataforma Android de Google se ha convertido en un ecosistema de desarrollo de aplicaciones para dispositivos móviles. Muchos son los dispositivos que soportan diferentes versiones del sistema Android que junto con la plataforma de su adversario principal, Apple (a día de la redacción de esta memoria acaba de presentar su smartwatch iWatch), abarcan la mayoría de los dispositivos existen en el mercado mundial.

Ventajas de los **wearables**

Los usuarios de wearables destacan varias de las ventajas que estos dispositivos ofrecen. Entre dichas ventajas destacan con un 77% de los casos, el ser más eficientes.

Ilustración 2 - Ventajas de los wearables (centrodeinnovacionbbva.com)

Por otro lado, también destacan que no todo son ventajas y que también hay otros inconvenientes como los que se muestran a continuación.

Ilustración 3 - Desventajas de los wearables (centrodeinnovacionbbva.com)

Uso de los wearables en el mundo.

En la actualidad es más que notable el uso de los nuevos dispositivos wearables. En especial en países de oriente como China e India que lideran el uso de estos con alrededor de un 81% en el uso para trabajos y en un 73% en uso personal.

Ilustración 4 - Usos (centrodeinnovacionbbva.com)

Este año compraremos

La presente campaña mediatica indica que, si a día de hoy tuviéramos que comprar un dispotivo wearable se trataría en mayor medida de una pulsera, normalmente dedicada a la actividad física, seguido de relojes, ropa y, en último lugar, unas gafas.

Mejoras de los wearables en nuestro

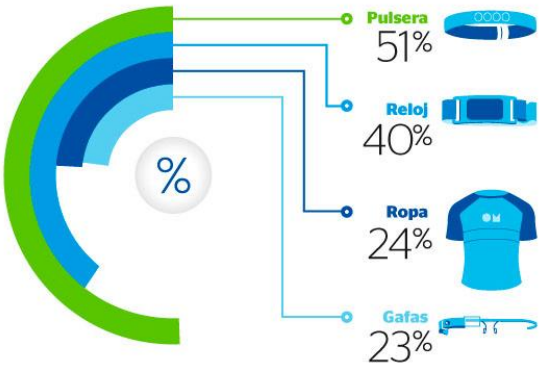


Ilustración 5 – Porcentajes de wearables en posibles compras

día a día

Más allá del uso de ocio, los wearables van pisando fuerte en la mejora de nuestro día a día. Hoy en día es posible, con tan sólo una pulsera, llevar un seguimiento de nuestra salud, realizar compras sin sacar nuestra cartera, u otras funcionalidades como el seguimiento geoposicionado para niños, recogida de datos automatizados del entorno, etc.

Ilustración 6 - Otros usos (centrodeinnovacionbbva.com)[1]

Tecnologías usadas

Durante el presente proyecto se han usado diversas tecnologías interconectadas entre sí para poder lograr concluir esta fase del proyecto con éxito. Las tecnologías usadas y su función han sido las siguientes:

- **Java:** Lenguaje de programación usado el motor de Oracle-Java como principal lenguaje de programación. Todo esto, para poder portar la versión a dispositivos móviles bajo una máquina virtual interna de Android llamada Dalvik Virtual Machine (DVM [5] (hasta su versión 5).
- **Android:** Se ha usado la capa de Android que ofrece Google de modo oficial para realizar las tareas de interconexión con los dispositivos y el sistema operativo que lleva el mismo nombre. La versión utilizada ha sido la 4.2 que es compatible con versiones iguales o superiores a la misma.
- **Gradle:** Es una herramienta que automatiza la construcción de los proyectos como por ejemplo tareas como la compilación, revisión de archivos y versiones. Ha sido usada bajo el IDE Android Studio con el fin de preservar la estructura íntegra del proyecto tanto en local como en futuras versiones en remoto.
- **XML:** Como lenguaje de maquetado para diseños. El XML es un lenguaje de marcas que en Android tiene la principal función de permitir maquetar los diseños de un modo jerárquico.
- **JSON:** Es un tipo de empaquetamiento de datos. Ha sido usado para poder transportar los datos desde un dispositivo a otro o bien para guardar internamente la estructura de los datos.
- **SQLite:** Envuelto en la capa de abstracción del framework ADA nos encontramos con el motor de base de datos SQLite, interno de Android, para gestionar los datos tanto en el reloj como en el dispositivo handheld.

Configuración del Proyecto

Para la configuración y desarrollo del proyecto se ha usado AndroidStudio [6]. Un IDE basado en IntelliJ Idea[7] que permite un rápido desarrollo de aplicaciones basadas en Android. Dispone de múltiples ejemplos en los que el programador se puede apoyar (como ha sido mi caso) para poder llevar a cabo su programa. Por defecto, al crear un proyecto bajo AndroidStudio nos crea un proyecto Gradle que, como comenté en el apartado anterior, sirve para mantener una estructura bien definida del proyecto y mantener organizado el mismo.

No han sido necesarios cambios en el IDE ni la instalación de ningún plugin externo para llevar a cabo el proyecto.

Librerías y Componentes externos

Para una mejor experiencia tanto por parte del programador como por parte del usuario al usar la aplicación, se han usado algunos aspectos, definiciones y librerías que circulan por internet bajo licencia GPL.

- **ADA Framework** [4]:
  Del inglés **"**Android Data Abstraction Framework", dicho framework nos ofrece un sistema de interconexión de clases a modo de ORM nativo para Android (versiones 2.2+) que interactúa de fondo con una base de datos SQLite facilitando el almacenaje y recuperación de datos sin interactuar directamente con consultas SQL (aunque pueden ser usadas de igual manera). Esto ha sido usado principalmente para el almacenamiento y recuperación de los datos cardiacos obtenidos por la aplicación mediante el sensor del reloj.
- **Material Design** [8]
  "Durante el pasado Google I/O 2014, la conferencia que da Google cada año, se presentaron muchas novedades siendo una de ellas este nuevo diseño. **Material Design es un concepto, una filosofía, unas pautas enfocadas al diseño utilizado en Android**, pero también en la web y en cualquier plataforma. Material Design recibe su nombre por estar **basado en objetos materiales**. Piezas colocadas en un espacio (lugar) y con un tiempo (movimiento) determinado.

Es un diseño donde la **profundidad, las superficies, los bordes, las sombras y los colores juegan un papel principal.** Precisamente este diseño basado en objetos es una manera de intentar aproximarse a la realidad, algo que en un mundo donde todo es táctil y virtual es difícil. Material Design quiere guiarse por las leyes de la física, donde **las animaciones sean lógicas, los objetos se superpongan pero no puedan atravesarse el uno al otro y demás.**" Todo esto se podrá ver con más detalle en las aplicaciones desarrolladas y en el apartado dedicado al diseño de esta misma guía.

- **Otras librerías (gráficas)**
  Durante el presente proyecto se han usado otras librerías para maquetar el diseño de algunas partes de los clientes (tanto el reloj como handheld). Estas han sido localizadas bajo licencias GPL gratuitas y se hace referencia a ella en cada parte de los clientes en este proyecto.

Control de versiones

Para el control de versiones del proyecto se ha usado git[9] como cliente principal teniendo dos ramas principales: La master y la trunk (de pruebas). Actualmente el código se almacena en la plataforma bitbucket[10] en el repositorio moilodsan/tfg [11].

Cliente Wearable

Durante el transcurso de esta sección definiremos los aspectos más importantes del proyecto. En especial haremos hincapié en el proceso de desarrollo e investigación que se ha llevado a cabo en clientes wearables con respecto al cliente handheld. Debido a que Android empieza a ser una plataforma más que conocida a nivel mundial, no pararemos tanto en el trabajo realizado en el dispositivo.

Un cliente Wearable, a partir de ahora, será definido como una aplicación que llevaremos en nuestro wearable, es decir, reloj, pulsera, gafas, etc. En nuestro caso, se trata de la aplicación ***aHeartRate***.

Cambiando el chip - Programemos para un wearable

Todo aquel programador que en un momento u otro de su vida se ha enfrentado a programar para la plataforma de Android habrá notado la gran facilidad para realizar programas (incluso más hoy en día con aplicaciones como la desarrollada por el MIT para casi hacer una aplicación realizando un mero puzle conocida como AppInventor[12]). A medida que pasan los años los dispositivos van teniendo más y más recursos (y ya no sólo hablo de los dispositivos con Android). Disponemos de teléfonos móviles con 120gb de espacio en disco, 4gb de memoria interna, un procesador de 8 núcleos y una pantalla de 5 pulgadas, suficientes para correr cualquier programa.

En cambio, con los dispositivos wearables la cosa no es tan espléndida. Dando un paso atrás al pasado nos hemos encontrado con un dispositivo que, contando con lo que consuma el sistema operativo Android interno, dispone de 512mb de RAM, 4GB de almacenamiento interno y una pantalla de menos de 2 pulgadas (1.65 aunque no es operativa en el 100% del área porque hay que descontar el borde) lo que nos lleva a enfrentarnos, por decirlo de alguna manera, a los primeros programas que se realizaban para Android. Hay que tener en cuenta los tipos de datos usados, la gestión de la memoria (ya no vale el empezar a abarcar recursos como si no hubiera mañana porque en muchas ocasiones el mismo sistema operativo te limita). Ahora tampoco disponemos

de una gran pantalla como la del móvil para poner un área de texto e ir mostrando mensajes de debug (y ya ni hablar de la consola de debug que dispone, por ejemplo, el IDE Android Studio con el que hemos trabajado).

Plataforma de soporte de la aplicación

En este apartado hablaremos sobre las especificaciones del Samsung Gear Live[13], el reloj "wearable" que hemos usado para realizar el proyecto. El reloj fue adquirido en la página de Google y traído a Canarias por la compañía iCanduty pues la empresa de Silicon Valley no envía – de momento – a las islas Canarias.

Ilustración 7 – Imagen de presentación Samsung Gear Live

General	Red	No
Tamaño	Dimensiones	37.9 x 56.4 x 8.9 mm
	Peso	59 g
Pantalla	Tipo	Super AMOLED capacitivo, 16M colores
	Tamaño	
Memoria	1	• 4GB memoria interna, 512MB RAM • Procesador 1.2GHz • Sensor acelerómetro • Giroscopio • **Monitor de ritmo cardíaco** • Certificación IP67: a prueba de polvo y **agua**
Características	OS	Android Wear
	Cámara	No
	Otros	• Brújula digital • Cronómetro • Podómetro • Bluetooth v4.0 LE • Servicios Google • Comandos de voz
Batería		Standard, Li-Ion 300 mAh

Compatibilidad de la aplicación con otros relojes Wearables

La compatibilidad del software desarrollado se podría decir que es elástica pues se adapta a los dispositivos actuales de este tipo. Es decir, en un primer momento se ha diseñado la interfaz del programa para el wearable antes mencionado, no obstante el diseño se ha realizado mediante una interfaz circular que permite migrar el software, si los requisitos del sistema lo permiten, a otro tipo de relojes que actualmente están en el mercado donde la forma es circular. Este es el caso, por ejemplo, del motorola360[14]. Hay que tener en cuenta que, para funcionar, debe tener un sensor que permita la recepción y medición de frecuencia cardíaca como tiene el GearLive.

Existen otros dispositivos que no serán compatibles debido a que las especificaciones, tanto de hardware como de software no son compatibles con el sistema como pueden ser relojes de bajo coste “crossfit” usados para únicamente indicar la hora y tomar las pulsaciones en un determinado momento.

Analizando los datos

Los datos recogidos fueron analizados empíricamente desde el principio. Quería ver cómo se comportaban con respecto a las pulsaciones reales de modo que siempre que tomaba muestras de los mismos, lo hacía tomando muestreos de 50 valores. Me daba cuenta que por norma general los 10 primeros datos eran erróneos (desde que el sensor se enciende hasta que realmente empieza a tomar valores pueden pasar unos segundos, y son éstos los datos a descartar). De modo que de 50 datos siempre me quedaba con 40, un número excesivo. De esos 40 datos pude observar, realizando comparaciones a modo visual que los datos que más se solían acercar al valor real indicados por el pulsímetro eran los valores máximos o cercanos a estos. También me di cuenta que si los valores se repiten en todo el muestreo, significa que los datos han sido erróneos en su medición y se descartan. En la versión de debug no se descartan para poder testear y se toma el único valor único para representar la pulsación. Estos valores se suelen obtener cuando se tiene la batería en carga o se lanza a tomar las pulsaciones sin estar en contacto directo con la piel (reloj mal colocado, fuera de la muñeca, etc.).

Limitaciones

El dispositivo cuenta con, no muchas limitaciones, pero sí importantes que podrían hacer que, en caso de no contar con éstas, se tratase de un dispositivo sumamente potente e independiente. Ahora mismo depende en su totalidad de un cliente wearable que le permita realizar conexiones a internet. Si el reloj, por ejemplo, quisiera realizar una búsqueda en google, primero habría que indicarle al reloj lo que éste quiere buscar, luego se interconectaría con el dispositivo móvil y sería éste el encargado de hacer la búsqueda, conectarse con el dispositivo wearable y luego mostrar la información en él. Podría disponer de una ranura, por ejemplo, para tarjetas de datos con conexión a internet. Esto supondría que los problemas que más adelante son explicados, quedarán en una mera anécdota.

Otro gran problema es la escasa memoria pues el dispositivo cuenta con tan sólo 512MB de RAM compartidas con el sistema operativo. Esto hace que a la hora de realizar una programación orientada a ese dispositivo uno tenga que hacer más de una triquiñuela y cuidar mucho los tipos de datos usados y pantallas a mostrar para no agotar la memoria del dispositivo (es por eso que al final se optó por una simple pantalla). Otro hándicap en lo que a limitaciones se refiere es la poca cobertura de bluetooth, a pesar de contar con un chip en su versión 4, el tamaño y la construcción limitan los metros ofrecidos por un dispositivo normal.

Diseño de interfaz

Desde un primer momento y tras ver las limitaciones del dispositivo, decidí implementar una interfaz que fuera cómoda e intuitiva para el usuario. Quise dotarla únicamente de un botón central que activara y desactivara la función en el reloj, para ello usé las librerías "RippleBackground"[15] obtenidas de un repositorio gratuito de github. Éstas te permiten poner un botón (como se ve en la imagen de la derecha) donde al hacer click se expanden unas ondas azules en dirección a los extremos para dejar constancia de que el servicio está en movimiento (en ejecución). Cuando se vuelve a hacer click sobre el mismo botón las ondas desaparecen indicando que el servicio ha quedado inactivo (detenido). A continuación se mostrará un pequeño fragmento de cómo de simple es el código usando las librerías antes mencionadas.

Ilustración 8 – Captura aplicación wearable detenida

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#0D47A1">
    <com.skyfishjy.library.RippleBackground

xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/content"
        app:rb_color="#90CAF9"
        app:rb_radius="32dp"
        app:rb_rippleAmount="4"
        app:rb_duration="3000"
        app:rb_scale="6">
        <ImageView
            android:layout_width="64dp"
            android:layout_height="64dp"
            android:layout_centerInParent="true"
            android:id="@+id/centerImage"
            android:src="@drawable/heartbutton" />
    </com.skyfishjy.library.RippleBackground>
</LinearLayout>
```

Ilustración 9 – Adaptación de los diseños a distintos tipos de diseño del wearable

Para empezar a usar este tipo de interfaz basta con definir un par de variables y de establecer esa interfaz como predeterminada de arranque.

```
    private RippleBackground rippleBackground = null;
```

```
setContentView(R.layout.rect_ripple);
            rippleBackground = (RippleBackground) findViewById(R.id.content);
            ImageView imageView = (ImageView) findViewById(R.id.centerImage);
            imageView.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {

                    if (switchButton) {
                        stopAlarm();
                        rippleBackground.stopRippleAnimation();


                    }else {

                        startAlarm(view);
                        rippleBackground.startRippleAnimation();

                    }

                    switchButton = !switchButton;

                    }
            });
```

Problemas encontrados

Uno de los mayores problemas que me encontré durante el desarrollo fue el acceso a la base de datos desde una consola. Para poderte conectar a la base de datos y explorar el contenido de las tablas generadas debes conectarte mediante ADB, esto suponía el tener que tener el dispositivo conectado al ordenador (o vía bluetooth) para poder acceder mediante la consola. Una vez realizada la conexión me di cuenta de que con el dispositivo handheld o wearable sin tener acceso como root no me iba a ser posible el acceso a la aplicación de gestión de Sqlite3 de modo que al final tuve que hacerme, como se aprecia en el modo debug explicado más adelante, una propia interfaz dentro del reloj y del handheld que me permitiera hacer lo propio de un CRUD (Create, Read, Update and Delete).

Hay otros problemas en lo referente a la conexión y formato de los dispositivos pero esto se tocará más adelante en el apartado de interconexiones.

Por último y más común en el ámbito del desarrollo de aplicaciones, fue la concurrencia de los múltiples hilos que se manejan en la aplicación. Cada elemento en la interfaz es un hilo independiente, según java, y si a eso le sumamos que hay procesos de fondo generados por una AsyncTask que en ciertas ocasiones necesitas controlar el flujo de las llamadas, se puede (y pudo) convertir en un dolor de cabeza. Afortunadamente este tipo de problemas son “fáciles” aunque costosos de solventar y fueron corregidos a medida que se fue desarrollando la aplicación.

Código

En este apartado veremos segmentos de código que considero importantes para esta memoria. Después de cada segmento se explica la funcionalidad y en qué elementos intervienen.

Manifest.xml : Es el archivo de configuración principal del proyecto Wear. En los siguientes segmentos de código vemos el servicio definido que será usado para la medición de forma pasiva del reloj y la meta, que indica el código del sensor de toma de pulsaciones que junto al permiso "`user-permission`" de lectura del BODY_SENSORS permite que la aplicación pueda recoger los datos oportunos.

```
<service android:name=".sensorServices.HeartRateSensor"
    android:exported="false"
    android:enabled="true"
    android:label="heartRateBG">
    <intent-filter>
        <action android:name="com.google.android.gms.wearable.BIND_LISTENER" />
    </intent-filter>
</service>

<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

Se le indica al manifest que se usarán los sensores corporales.

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

Se le indica al manifest que el hardware usado es un reloj.

```
<uses-feature android:name="android.hardware.type.watch" />
```

MainActivity.java

Uno de los métodos más importantes es el de enviar la información almacenada en el reloj al dispositivo handheld. Para ello creé una función que se encarga de, a la hora de sincronizar con el dispositivo handheld, enviar los datos almacenados en el reloj recogiendo estos del ORM.

```
1  private void sendStoredData() {
2
3          // Si hay elementos por enviar en la BDD se mandan antes que el último
4          if( dataContext.heartDataORM.size() > 0 ){
5
6              ArrayList<String> tmpData = new ArrayList<String>();
7
8              for( int i = 0 ; i < dataContext.heartDataORM.size() ; i++ ) {
9
10                 HeartRateModel tmp = dataContext.heartDataORM.get(i);
11
12                 if (tmp.heartData.isEmpty()){
13                     DebugApp.Logi("Temporal Vacío");
14                 }else{
15                     Collections.addAll(tmpData, tmp.heartData.split(","));
16                     sendDataToMobile(tmpData, tmp.date);
17                 }
18
19                 dataContext.heartDataORM.remove( i );
20                 try {
21                     dataContext.heartDataORM.save();
22                 } catch (AdaFrameworkException e) {
23                     e.printStackTrace();
24                 }
25
26                 tmpData.removeAll( tmpData );
27
28             }
29
30         }
31
32     }
```

Otro método que destaca por su simplicidad es la acción de mensaje de confirmación recibido. Como se puede observar en el código es muy limpio dado que gracias a la interfaz proporcionada por Google recibir un mensaje de otro dispositivo es así de sencillo.

```
1      @Override
2      public void onMessageReceived( MessageEvent messageEvent ) {
3
4          String receivedData = new String( messageEvent.getData() );
5          if( !pendingReceivedConfirm.contains( receivedData ))
6              pendingReceivedConfirm.add( receivedData );
7
8      }
```

Y por último, aunque no menos importante, unas líneas de la función “synchronized sendDataToMobile” que se encarga de enviar el contenido de los datos al dispositivo handheld. La función tiene en torno a las 100 líneas (líneas de debug incluídas) de modo

que no lo pondré en la memoria pero sí el cómo se crea una tubería “pipe” o canal de comunicación para el envío de datos mediante la API de Google.

```
            // Definimos el formato de fecha a enviar
            DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
            String measurementDateStr = dateFormat.format( measurementDate );
            PutDataMapRequest dataMap = PutDataMapRequest.create("/dataa");
            dataMap.getDataMap().putStringArrayList("heartRateData", heartDataArrayList);
            dataMap.getDataMap().putString("measurementDate", measurementDateStr);
            PutDataRequest request = dataMap.asPutDataRequest();
```

Como se puede ver en esas 7 líneas, se empaquetan los datos tal y como se van a enviar (un arraylist y un tipo date que terminan siendo strings) y se crea un dataMap bajo la tubería (pipe) “/dataa”. Posteriormente, como se verá en el apartado de interconexiones, se crea una tarea de fondo (para no bloquear la interfaz gráfica) para enviar los datos. Este método ha sido usado en ambos clientes.

HeartRateManager.java

La clase HeartRateManager se encarga de gestionar todo lo relacionado con el inicio y fin de la lectura del sensor. Inicia el sensor, lo para, recibe los datos y los envía.

```
public HeartRateManager(Context appContext){

        rates = new ArrayList<String>();
        mSensorManager = (( SensorManager )appContext.getSystemService(
appContext.SENSOR_SERVICE ));
        mHeartRateSensor = mSensorManager.getDefaultSensor( SENSOR_TYPE_HEARTRATE );

    }
```

Cada vez que el sensor detecta una pulsación se invoca al método “onSensorChanged” que se encarga de comprobar que la medición no sea errónea y añade el valor al array de valores.

```
if( sensorEvent.values[0] > 0 ){
            rates.add( Float.toString( sensorEvent.values[0] ) );
        }
```

En esta misma clase existe un método propio de la clase AsyncTask que a la hora de ejecutar la instancia, si ha pasado mucho tiempo y no se han obtenido datos, borra el contenido de los datos obtenidos y devuelve “-1” para indicar que han sido erróneos.

AlarmReceiver.java

Esta clase se encarga de hacer de intermediaria. Una vez se establece la alarma en la clase principal se envía una señal a esta clase y ésta se encarga de crear una nueva instancia de medición.

```
1     public void cambiarRate( ArrayList<String> rates ){
 2
 3         Intent maintActivityIntent = new Intent( contextoApp.getApplicationContext() ,
MainActivity.class );
 4         maintActivityIntent.putExtra( "heartRate" , rates );
 5         maintActivityIntent.setFlags( Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_NO_ANIMATION );
 6
 7         contextoApp.getApplicationContext().startActivity( maintActivityIntent );
 8         DebugApp.Vibrar( this.contextoApp );
 9
10     }
11
12     @Override
13     public void onReceive(Context arg0, Intent arg1) {
14
15         contextoApp = arg0;
16         DebugApp.Vibrar( contextoApp );
17         new HeartRateManager( contextoApp ).execute( this );
18         HeartRateSensor.setNewAlarm();
19
20     }
```

Habilitar conexión mediante bluetooth sin necesidad de cables

El dispositivo wearable es sin duda un gran juguete con el que pasar horas trasteando. El inconveniente en este proyecto es que para tomar las pulsaciones hay que tener el sensor pegado a la piel. Para pasar un archivo compilado apk a nuestro reloj debemos conectar la base de la batería y posteriormente vincular por USB el dispositivo a nuestro ordenador. Esto puede resultar algo tedioso el realizarlo cada vez que vayamos a probar nuestro proyecto de modo que ideé un pequeño script de apenas dos líneas que abre un puerto de escucha por el protocolo TCP en el puerto 4444 y que se conecta por el ADB del otro dispositivo (que sí debe estar conectado al PC) de modo que se pueda enviar esta aplicación sin necesidad de estar conectando y desconectando los otros dispositivos. Para ello sólo hay que abrir el AndroidWear, en ajustes e indicar que quieres activar la depuración por Bluetooth. Una vez hecho esto (tanto en el reloj, habilitando el modo de depuración por bluetooth como en el dispositivo handheld, ejecutamos las siguientes líneas de código en un terminal).

```
./adb forward tcp:4444 localabstract:/adb-hub
```

```
./adb connect localhost:4444
```

Unos segundos después podremos ver que ambos dispositivos quedan vinculados y que desde nuestro AndroidStudio podemos enviar la apk a nuestro reloj sin necesidad de que haya sido conectado al ordenador.

Ejecución dentro del Wear

Lo primero, una vez instalada la aplicación en el terminal móvil (en mi caso la unidad de prueba fue una Asus Nexus 7), debería aparecer en el wearable, después de hacer "tap" en la pantalla de inicio, el icono que da acceso a la aplicación.

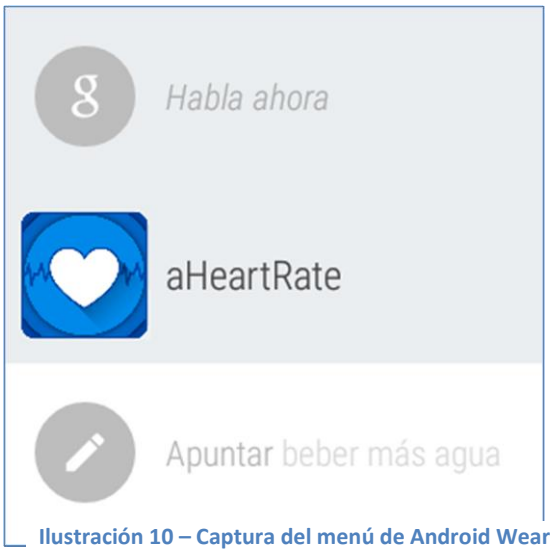


Ilustración 10 – Captura del menú de Android Wear

Una vez entremos en la aplicación nos encontraremos con sólo un botón (y nada más). Este diseño se realizó así para aportar sencillez imitando los patrones que sigue Apple con sus diseños.

Al hacer tap (lo que sería un click con el dedo) en dicho botón la aplicación en el reloj empezará a funcionar, tomando en primera instancia las pulsaciones cuando se activa la medición. Sabremos que esto es así cuando veamos aparecer unos círculos concéntricos que salen desde el centro de la pantalla a los bordes como se puede ver en la segunda y tercera figura.

Ilustración 11 – Captura de inicio de la aplicación en el wearable

Ilustración 12 – Evolución de la captura con respecto a la ilustración anterior

Para finalizar la medición tan sólo debes pulsar nuevamente sobre dicho botón para que la aplicación internamente detenga los servicios de escucha y pare la temporización de mediciones.

Ilustración 13 – Mensaje de alerta cuando se detiene el servicio

Modo debug

Como hice mención en el primer apartado de esta sección, el trabajar con nuevas tecnologías es siempre un muro a franquear. Nunca fui consciente, hasta que me puse manos a la obra y trabajé con el dispositivo wearable, lo que podría suponer el disponer de uno. Cuando uno está acostumbrado a trabajar con Android tiene la costumbre (buena o mala) de imprimir cosas por la pantalla del terminal. Nos encontramos que en los wearables eso no es del todo posible debido a que las dimensiones de estos dispositivos no suelen ser lo suficientemente grandes para imprimir mensaje alguno. De todos modos, en una primer aproximación a lo que fue el prototipo de testeo del cliente wearable, sí se incluyó una pantalla de debug, un panel desde donde se podía hacer funcionar la aplicación, borrar datos, etc.

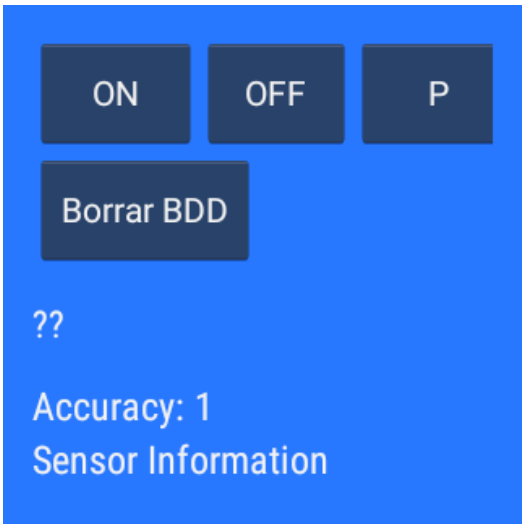


Ilustración 14 – Pantalla inicial en modo debug

Ilustración 15 – Pantalla debug cuando se han tomado pulsaciones

Antes de empezar con el cómo funciona el modo de debug creo conveniente comentar cómo se activa éste en el reloj. Dado que no es una funcionalidad que el usuario vaya a necesitar con frecuencia y que sólo debe ser usada para el desarrollo, se ha incluido una variable dentro del archivo de configuración del proyecto "Wear/ConfigurationVariables.java" llamada DEBUG_APP, generalmente establecida a false.

```
1 // Variable de tipo debug para imprimir logs o no y activar debug
2     public static boolean DEBUG_APP = false;
```

Al cambiar ésta a true y cargar la APP en el móvil aparecerá esta pantalla que funciona de la siguiente manera.

En la pantalla vemos cuatro botones, cada uno tiene una funcionalidad distinta que interactúan con los diferentes sensores y funcionalidades del reloj para poder probar las diferentes configuraciones. Justo debajo del último botón, “Borrar BDD” vemos dos campos de texto (en la imagen de la izquierda) y tan sólo uno en la de la derecha. En primera instancia vemos un botón de “ON” que activará el sensor y creará las “alarmas temporales” que le indican al reloj cuándo medir. Actualmente la medición está controlada en minutos por la variable measurementTime.

```
1     // Tiempo de espera entre medición y medición ( minutos )
2     public static long measurementTime = 5;
```

Hay que añadir que la alarma, después de la versión KITKAT no es exacta (y menos en éste tipo de dispositivos) pues la intención es ahorrar batería.

*“**Note:** Beginning with API 19 (KITKAT) alarm delivery is inexact: the OS will shift alarms in order to minimize wakeups and battery use. There are new APIs to support applications which need strict delivery guarantees” [16]*

Después nos encontramos con el botón “OFF” que se limita a detener los servicios de fondo y cancela la medición de pulsaciones a menos que en ese momento se estén tomando en cuyo caso sigue el ciclo de la vida del programa hasta terminar de leerlas y luego para. El botón “P” imprime (Print) por consola siempre y cuando el dispositivo esté conectado al modo debug (debido a las restricciones de pantalla) todo el contenido de la base de datos gestionada por el Framework ADA que hace de intermediario como un ORM con algunas clases. Por último, el botón de borrar BDD que recorre todos los objetos contenidos en el ORM y los borra.

Cliente Handheld

En este apartado no veremos nada con detalle debido a que la mayor carga de trabajo se la ha llevado el cliente wearable. El cliente del handheld es un mero receptor de información por el mismo método que se envían y reciben datos desde el wearable. Al recibir un dato nuevo se genera un md5 y éste es enviado al wearable para confirmar mediante el hash que los datos se han enviado correctamente. Para el diseño, al igual que en el apartado anterior, se han usado librerías externas que veremos a continuación basadas en el Material Design de Google.

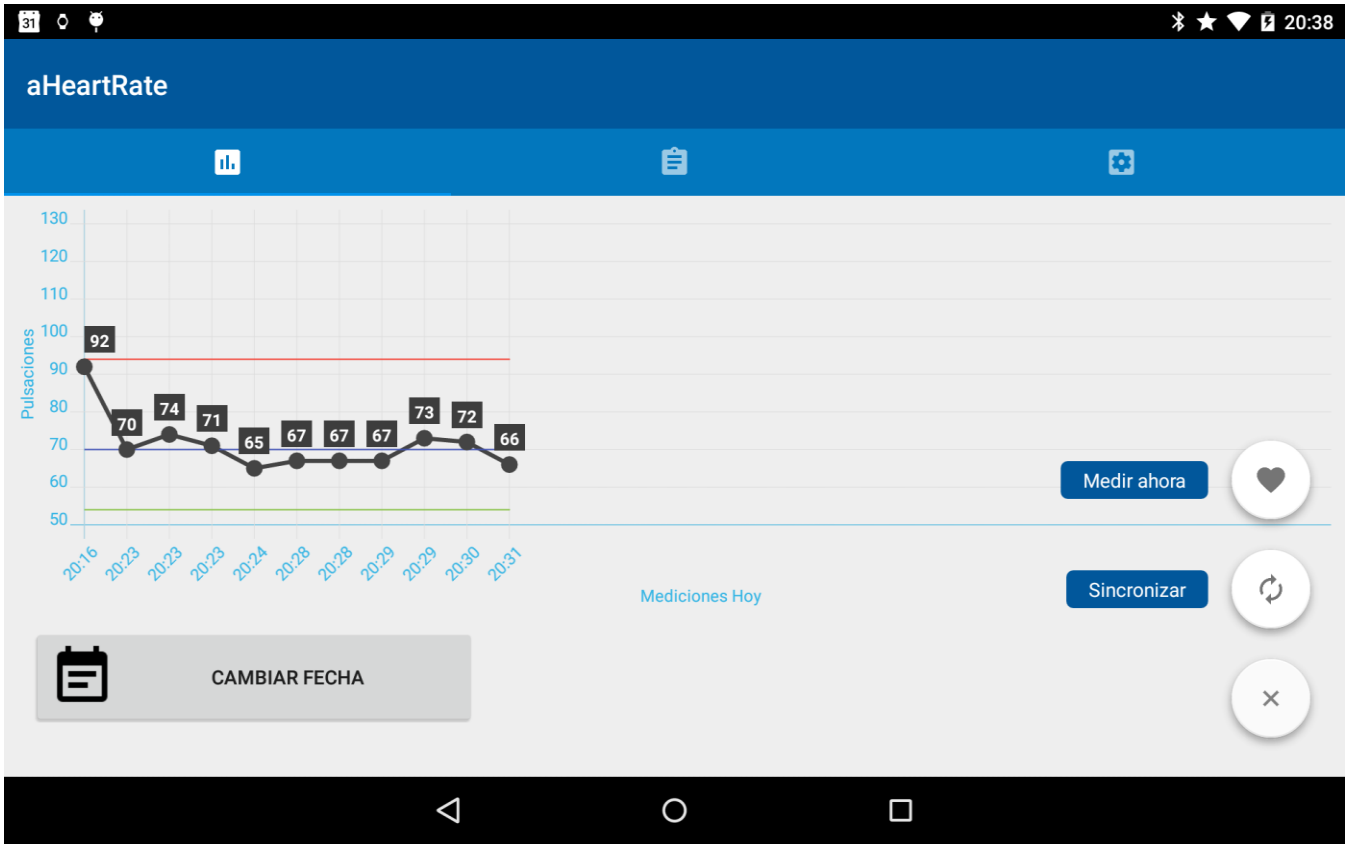
Descripción del sistema en móvil

El sistema del handheld, una vez iniciado, se intenta conectar con el dispositivo wearable mediante las librerías Google y funciones propias. El dispositivo handheld se queda a la escucha esperando a recibir los datos del otro dispositivo pareado. Cuando recibe los datos, éste lo comunica al wearable para confirmar que todo esté correcto y envía los datos al ORM (mismo modelo que en el reloj) para luego refrescar una gráfica que permita al usuario mantener el control del día a día.

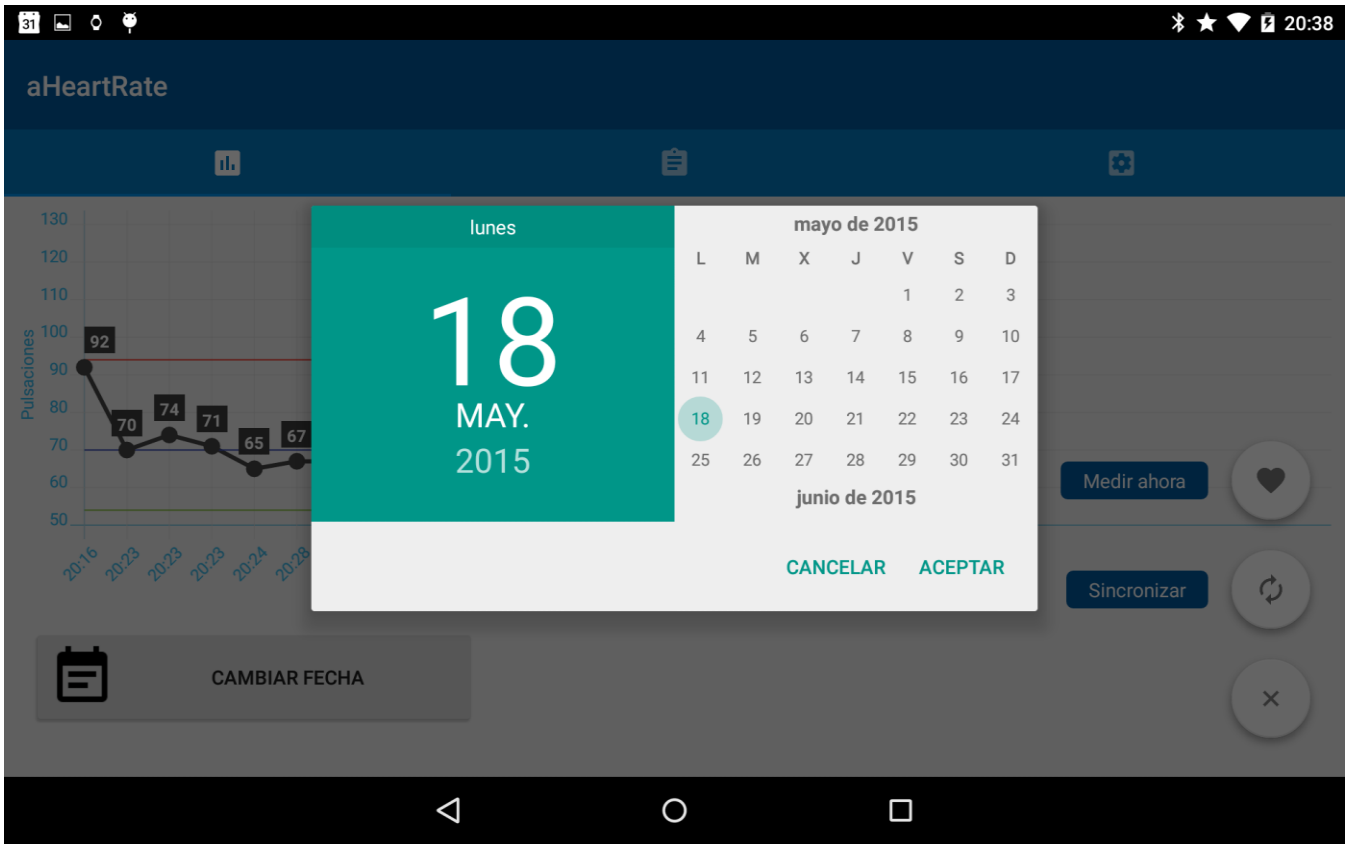
Desde la pantalla principal, como se puede ver en la captura, existe un botón desplegable en la esquina inferior derecha que al hacer click despliega dos opciones:

- Medir ahora: Envía una señal al reloj por la tubería con la señal “medir” y esto fuerza una medición en el reloj a menos que ya se encuentre realizando una.
- Sincronizar: Si por algún motivo no ha pasado el ciclo de medir y quieres sincronizar los datos del reloj con la aplicación principal del handheld, al hacer click en este botón se forzará una sincronización como si se hubiera vuelto a recuperar la conexión entre ambos. Aparte de lo comentado, también sirve para incluir datos nuevos para mediciones sincronizadas con el calendario de google.

16 Captura pantalla normal en cliente handheld



17 Captura pantalla con botón de cambiar fecha



18 Captura Calendario

Material Design en versiones de Android 4

Una gran desventaja con trabajar con dispositivos con versiones antiguas de Android o que dispongan de un hardware que no sea compatible con los nuevos diseños es la no-adaptabilidad a las nuevas tecnologías y nuevos diseños. A la aplicación, desde un principio, se le ha querido dar un aspecto pulido, limpio y nuevo. Por ello se optó desde un primer momento por el uso de los conocidos patrones de diseño "Material Design" creados por Google optando por unos diseños planos pero con volumen para destacar ciertos elementos. Para poder optar a ellos en versiones previas a la versión 5 de Android se han usado librerías externas que han permitido la maquetación de este tipo de diseño en dispositivos compatibles, al menos, con la versión 4.2 de Android (y no previas pues hay problemas de compatibilidad a la hora de sincronizar, como veremos más adelante).

Para la versión de handheld (Tablet / móvil) se han usado las siguientes librerías:

- android-floating-action-button Usada principalmente para aportar un botón de funcionalidad en Android.
- hellocharts-android Usada para representar las gráficas cardíacas en el cliente handheld.
- MaterialTabs Usado para el esquelto principal de la aplicación que se divide en tabs que hacen de divisor entre las diferentes secciones de la aplicación.

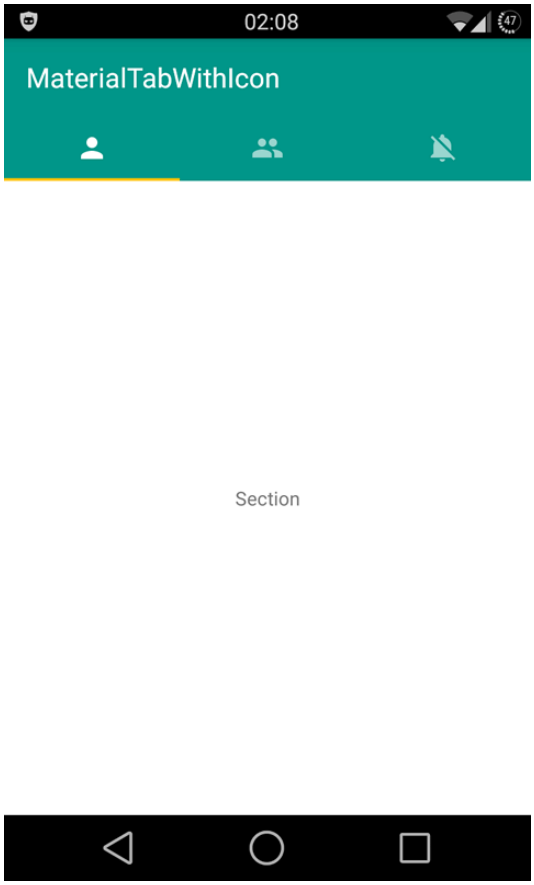


Ilustración 19 – Sistema base de la plantilla del cliente handheld que permite simular "tabs" con deslizamiento entre las pantallas

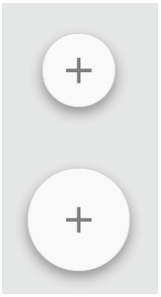


Ilustración 20 – Botón que se incluye en la parte inferior derecha de la aplicación handheld

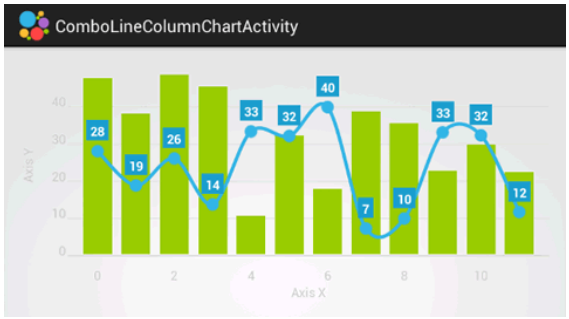


Ilustración 21 – Ejemplo de librería gráfica usada en el cliente handheld

Servicios de fondo (receivers)

Los servicios de fondo de los que dispone el handheld son muy similares a los explicados con anterioridad en el wearable. Es decir, se pone a la escucha por la api de interconexión y espera recibir datos nuevos. Cuando estos datos son recibidos son analizados por la misma aplicación y reportado el estado al reloj en caso de acierto o fallo en el envío.

Interconexiones

Cuando uno piensa en interconexiones en Android se le vienen a la mente diferentes modos de envío de datos de un dispositivo a otro lo que nos permite, como dice el título, interconectar de un modo u otro, ambos dispositivos. Para la interconexión entre un dispositivo wearable y un dispositivo móvil o Tablet hace falta cumplir algunos requisitos dado que al ser un sistema "nuevo" impone algunas restricciones para su uso.

Descripción de sistemas de Interconexión en Android

En primer lugar y una de las restricciones con mayor problemática es la versión de Android a manejar. Google impone que la versión debe ser al menos la 4.2.1 en el terminal móvil para poder bajar la aplicación oficial "Android Wear" y disponer de una rom oficial. Esto supuso un gran problema pues el dispositivo con el que trabajé disponía de una versión 4.1 y no podía interconectar el móvil con el reloj. Los móviles que hoy en día disponen de una versión 4.2.1 o superiores suelen ser de gama alta lo que encarece el presupuesto a la hora de querer disponer de un dispositivo wearable. En mi caso, al poco tiempo de recibir el reloj, adquirí un dispositivo "OnePlus One" donde sí pude trabajar con el dispositivo.

Una vez descargado y actualizado el Android Wear nos encontramos con que debemos parearlo con uno, y sólo un, dispositivo. Si intentas a posteriori parearlo con otro dispositivo, no te dejará debido a que siguen un tipo de conexión 1 a 1. Si quieres desprenderte del dispositivo con el que has estado trabajando y parearlo con otro no sólo debes desvincularlos sino que debes reiniciar el wearable a estado de fábrica, lo que es una enorme desventaja.

Para poder parear ambos dispositivos se debe disponer de Bluetooth para que esté siempre activo. El consumo de batería no se ve demasiado afectado pues sólo se usa en los momentos de sincronización enviando una señal desde un dispositivo a otro.

Problemas encontrados

Durante el desarrollo de la aplicación, tanto en su versión Wear como en la versión para handheld, se encontraron diversos problemas relacionados con la interconexión entre dispositivos.

Quitando lo comentado en el apartado anterior sobre el número de dispositivos pareados y el tener que disponer siempre de una conexión bluetooth, nos encontramos con problemas tales como interferencias en las señales, capacidades y desconexiones.

La fuerza de una señal de bluetooth, en su versión 4.0 es de aproximadamente 12~20 metros (dependiendo del fabricante) y de un rango menor para otras versiones como la 2. Esto hace que la interconexión entre el reloj y el móvil se dificulte en gran medida si nos dejamos el teléfono/Tablet en algún lugar alejado donde la señal no sea lo suficientemente buena y no haya una pérdida de paquetes importantes.

Esto quiere decir que, ¿si en un momento de medición intentamos enviar los datos al dispositivo se perderían? No. Google ha implementado un servicio cerrado que en caso de no poder enviar unos datos por cobertura, desde que ésta se recupere envía lo último que se le ha querido enviar al dispositivo. Si sólo tomáramos pulsaciones cada 6 o más horas no habría problema porque, por lo general en las fechas que corren no pasamos más de ese período de tiempo sin nuestro Smartphone/Tablet y si se mide y no se consigue comunicar (porque bien esté apagado, fuera de rango u otro motivo por el cual pudiera fallar la transferencia de datos), al volver a reconectar podríamos disponer de los datos. Pero ¿qué pasa si medimos con mayor frecuencia? Antes comenté que sólo se enviaba el último dato, el resto queda eliminado por el smartwatch debido a, imagino, la falta de recursos para almacenar. Esto supuso un gran problema a la hora de programar la interconexión de datos.

El que lea esto puede pensar que una solución factible sería el empaquetamiento de datos en algún formato de modo que si no hubo respuesta por parte del cliente handheld se pudieran empaquetar todos aquellos datos que no se pudieran enviar. Yo también lo pensé… Google, en su api oficial para enviar datos limita, sin saber por qué, la longitud de los datos a enviar desde el reloj hasta el dispositivo. Esto quiere decir que, como se había comentado, si se quiere empaquetar los datos para poder enviarlos como el último y, en caso de que no se pueda enviar sí poder almacenarlos, no es posible.

Los casos donde este tipo de envío es un problema son los siguientes:

- Que el handheld se quede sin batería. Es muy probable que nuestro dispositivo se quede sin batería. También podría suceder que a nuestro wearable le suceda lo mismo (debido a que no sólo se usará para nuestra aplicación y su batería se puede ver afectada por otros procesos que agoten la batería).
- Que se aleje tanto que el reloj NO lo detecte. Un fallo que me llamó mucho la atención es que si alejas el dispositivo handheld e intentas enviar información desde el dispositivo wearable, da como válido el envío (cuando no ha sido recibido).
- Que el dispositivo quede desemparejado, que aunque no es un caso común se podría dar. El cliente del wearable envía los datos pero no llega a ningún destinatario.

Soluciones al apartado anterior

Para poder proseguir con el proyecto tuve que lidiar y afrontar los problemas explicados en el apartado anterior.

Para empezar, tenía que encontrar una solución al problema de detectar si los datos se habían enviado correctamente, para ello ideé un algoritmo para poder programar un protocolo propio de comunicación basado en el clásico “ping pong” del protocolo de comunicación ping[1] sólo que modificando el comportamiento de éste para recibir una información predeterminada por el programa.

Esto se llevó a cabo con el siguiente algoritmo secuencial:

En primera instancia, desde el wearable, se empaquetan los datos. Al principio se usaba JSON[3] como modo de empaquetar el objeto y enviarlo como un string, luego se optó por dejar que Google, con su api, enviara un objeto del tipo ArrayList debido a que es uno de los tipos nativos que es soportado en la interconexión de datos, permitiendo así el no empaquetar los datos en formato JSON de modo manual. Una vez se lanza el envío (ping) se pone un contador a 5 segundos y se guarda en local un hash que representa los datos enviados. Si durante los próximos 5 segundos se recibe un mensaje con el mismo hash que se ha guardado significa que los datos no sólo se han enviado sino que no ha

habido modificación durante el proceso (se podría llegar a realizar un ataque MITM[2]). Si se ha recibido correctamente (pong) se comprueba en la base de datos interna que ese hash está contenido y se borra de la base de datos usando el ORM del Framework ADA (recordemos que no siempre nos puede llegar el último hash enviado debido a la (des)sincronización con los hilos, es por eso que se ha guardado en un hash clave-valor -donde la clave es un md5- de los datos enviados).

Todo este tráfico de datos se hace de fondo, aunque los dispositivos tengan la pantalla apagada. Esto se hace mediante servicios, demonios del sistema Linux que trae Android por defecto y que suelen ser utilizados para comunicaciones en fondo. Se explicará con mayor detalle en el próximo apartado.

```
1  Wearable.MessageApi.addListener( mGoogleApiClient , new MainActivity() );
2  Wearable.NodeApi.addListener( mGoogleApiClient , this );
```

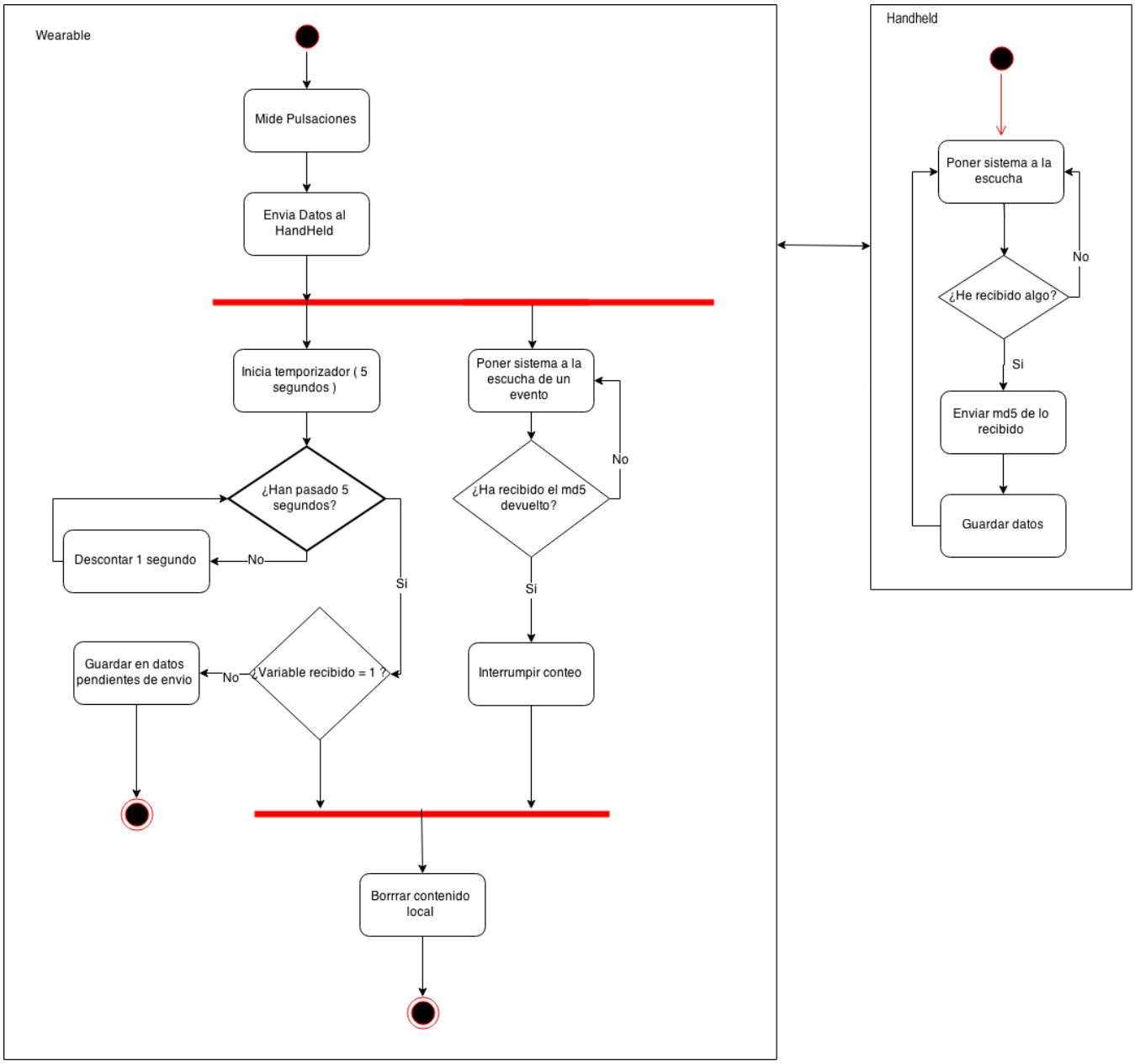


Ilustración 22 Representación general del algoritmo de interconexión de datos

Un fragmento de este código (con las partes más interesantes) es el siguiente (se han eliminado comentarios y mensajes de debug para ahorrar líneas.

```
1  new AsyncTask < HeartRateModel, Void, String > () {
2
3          @Override
4          synchronized protected String doInBackground(HeartRateModel...params) {
5
6              try {
7                  TimeUnit.SECONDS.sleep(5);
8              } catch (InterruptedException e) {
9                  e.printStackTrace();
10             }
11
12             ArrayList < String > tmpData = new ArrayList < String > ();
```

```
13              Collections.addAll(tmpData, params[0].heartData.split(","));
14
15              if( !pendingReceivedConfirm.contains( MD5.get( tmpData.toString())))
{
16
17                  if ( !params[0].heartData.equals("-1") )
18                      addNewDataToORM(params[0].date, tmpData );
19
20              } else {
21
22                  pendingReceivedConfirm.remove( MD5.get( params[0].heartData ) );
23
24                  try {
25                      dataContext.heartDataORM.fill();
26                  } catch ( AdaFrameworkException e ) {
27                      e.printStackTrace();
28                  }
29
30                  if (dataContext.heartDataORM.size() > 0) {
31
32                      for (int i = 0; i < dataContext.heartDataORM.size(); i++ )
33
34                          if( MD5.get(
                                dataContext.heartDataORM.get(i)
                                .heartData.toString())
                                .equals( MD5.get( params[0].heartData )))
35                              dataContext.heartDataORM.remove(i);
36
37
38                  }
39
40              }
41
42              return null;
43
44          }
45
46      }.execute( tmpHeartRateModel );
```

(WikiPedia) 1 Ping:

Como programa, ping es una utilidad diagnóstica en redes de computadoras que comprueba el estado de la comunicación del host local con uno o varios equipos remotos de una red a IP por medio del envío de paquetes ICMP de solicitud y de respuesta. Mediante esta utilidad puede diagnosticarse el estado, velocidad y calidad de una red determinada.

(WikiPedia) 2 MITM:

En criptografía, un ataque man-in-the-middle o JANUS (MitM o intermediario, en español) es un ataque en el que se adquiere la capacidad de leer, insertar y modificar a voluntad, los mensajes entre dos partes sin que ninguna de ellas conozca que el enlace entre ellos ha sido violado. El atacante debe ser capaz de observar e interceptar mensajes entre las dos víctimas. El ataque MitM es particularmente significativo en el protocolo original de intercambio de claves de Diffie-Hellman, cuando éste se emplea sin autenticación.

Interconexión en modo pasivo (de fondo)

Un servicio, en Android, se define como un proceso de fondo que no interactúa con el usuario. En los sistemas operativos GNU Linux este tipo de servicios son los conocidos como demonios (daemons). Se ocupan de realizar tareas en segundo plano en un hilo independiente del contexto principal (que generalmente maneja la interfaz de usuario en programas que dispongan de ella) debido a que, si se realizan grandes cálculos que mantengan la CPU ocupada en un tiempo prolongado, la interfaz de usuario principal podría quedar bloqueada.

Para nuestra aplicación se han definido servicios de comunicación de fondo que se encargan principalmente de comprobar el estado mutuamente entre un dispositivo y otro. Esto quiere decir que, apartando el hecho de comprobar el estado, se encarga de entregar y recibir mensajes.

Por parte del cliente del wearable tenemos una clase HeartRateSensor que implementa una interfaz de nodo ofrecida por google lo que implica que uno de los dos nodos finales y su comunicación serán cubiertos por esta clase.

Al crear una instancia de la clase se crea una alarma automáticamente. Una alarma es una clase interna de Android que se configura para poder ejecutar un código cada cierto tiempo, en este caso ha sido usada para crear una instancia métrica para las pulsaciones. Para evitar que se cree más de una instancia se ha usado un símil a un Singleton, es decir, un patrón que sólo permite la creación de un objeto de la clase para evitar colisiones y problemas en el caso de que hubieran dos o más instancias de la misma clase.

```
1     static public void setNewAlarm(){
2
3         if( pendingIntent != null )
4             manager.setExact( AlarmManager.RTC , System.currentTimeMillis()+(
ConfigurationVariables.measurementTime*60000 ) , pendingIntent );
5
6     }
```

También gestiona segmentos de código como por ejemplo cuando un cliente se encontraba desconectado y manda la señal de que está activo nuevamente. En ese caso, como se comentó en el apartado anterior, se fuerza la sincronización.

```
1      @Override
2      public void onPeerConnected(Node node) {
3          DebugApp.Logi("Se conectó el dispositivo al bluetooth");
4          MainActivity.MainActivityObject.forceSync();
5      }
```

Casos de Prueba

Para comprobar la fiabilidad del software, se han realizado diversos casos de prueba en personas reales que se ofrecieron voluntariamente para probar la viabilidad del proyecto. Para la selección de éstas se han tomado en cuenta varios factores como por ejemplo sexos, edad y condición física. La métrica para comprobar la fiabilidad de éstas son las desviaciones comparadas de un pulsímetro “profesional” con las reflejadas en la aplicación. En primera instancia se tomaron las pulsaciones, tal y como se comentó en un apartado anterior, en modo debug para ver qué valores eran los más cercanos a lo que indicaba el pulsímetro y así poder tomar una decisión de cuáles serían más fiables.

Durante el proceso de prueba no se estableció ninguna pauta física, es decir, no se le indicó a ninguno de los sujetos de prueba que tuvieran que estar en un estado de reposo o activo, tan sólo que tuvieran puesto el dispositivo wearable encima y no se lo quitaran. Las mediciones fueron realizadas, como caso de prueba, cada 10 minutos y un total de 15 veces en distintos horarios para comprobar que esto no afectara. A todos ellos, durante las primeras fases de prueba en distintas situaciones, se les tomaron las pulsaciones con un pulsímetro profesional justo antes de iniciar las mediciones y después (se descartó el uso del durante por si esto alteraban las mediciones del dispositivo wearable).

Sujeto A

Sujeto A

Sexo	Masculino
Edad	26 años

Wearable	Pulsímetro	Diferencia
89	89	0
90	90	0
87	87	0
79	80	-1
97	96	1
85	85	0
89	89	0
89	88	1
90	90	0
102	100	2
90	90	0
-1	93	-94
93	93	0
88	89	-1
85	85	0

Problemas encontrados con Sujeto A

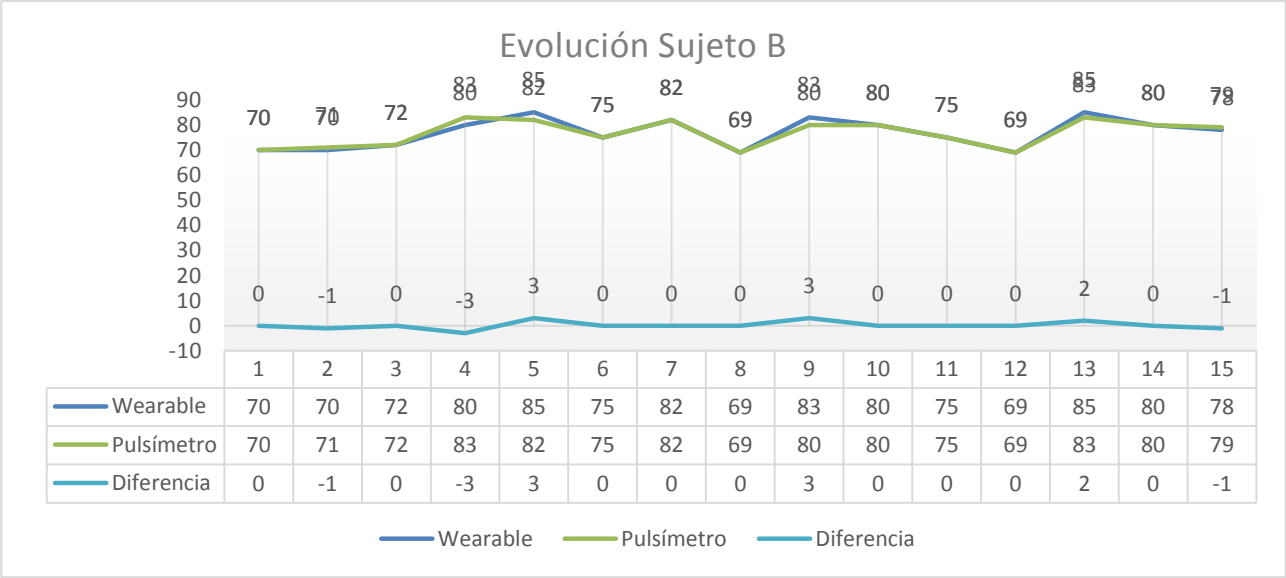
Durante el análisis de los datos del Sujeto A hubo un fallo a la hora de medir que queda reflejado en los datos como -1. Esto puede ser ocasionado cuando, a la hora de medir, el reloj no consiga tomar las pulsaciones de un modo adecuado (ya sea por no poder recolectar datos suficientes o bien por no recoger ningún dato). A la hora de pasar estos datos al dispositivo handheld son ignorados pues son datos de mediciones no válidas y quedan totalmente descartados.

Sujeto B

Sujeto B

Sexo	Femenino
Edad	25 años

Wearable	Pulsímetro	Diferencia
85	86	-1
86	86	0
83	85	-2
90	87	3
95	95	0
86	87	-1
89	89	0
90	93	-3
90	88	2
91	91	0
95	95	0
98	102	-4
92	92	0
89	88	1
93	93	0



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Wearable	70	70	72	80	85	75	82	69	83	80	75	69	85	80	78
Pulsímetro	70	71	72	83	82	75	82	69	80	80	75	69	83	80	79
Diferencia	0	-1	0	-3	3	0	0	0	3	0	0	0	2	0	-1

Problemas encontrados con Sujeto B

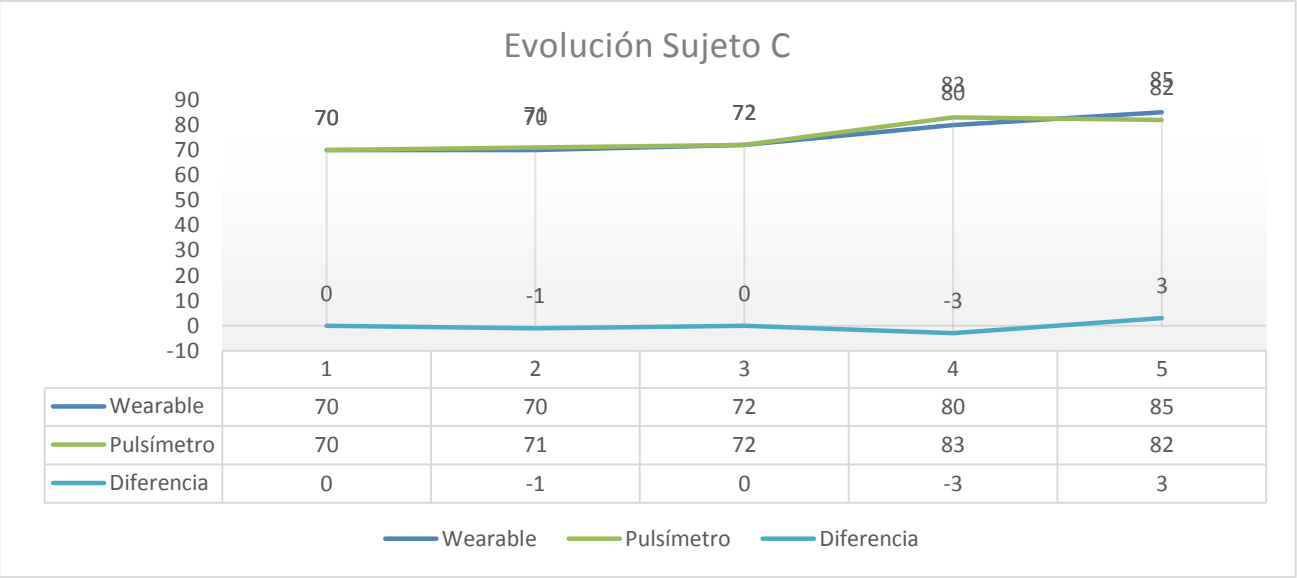
A la hora de tomar los datos en el sujeto B no hubo problema alguno salvo que en un momento puntual el dispositivo se quedó sin batería durante la quinta medición (marcada en la tabla de datos) por lo que hubo que recargar el dispositivo y volver a tomar mediciones.

Sujeto C

Sujeto C

Sexo	Femenino
Edad	30 años

Wearable	Pulsímetro	Diferencia
67	67	0
72	73	-1
75	75	0
69	69	0
72	71	1
67	67	0



Problemas encontrados con Sujeto C

Este caso fue fallido en primera instancia debido a que la correa del reloj no podía cerrarse lo suficiente como para poder disponer del sensor cercano a la piel en cualquier posición de la muñeca. Para solventarlo y comprobar la fiabilidad del sensor, tan sólo se tomaron 5 muestras en reposo teniendo el sensor pegado a la piel y el wearable sujeto con una mano.

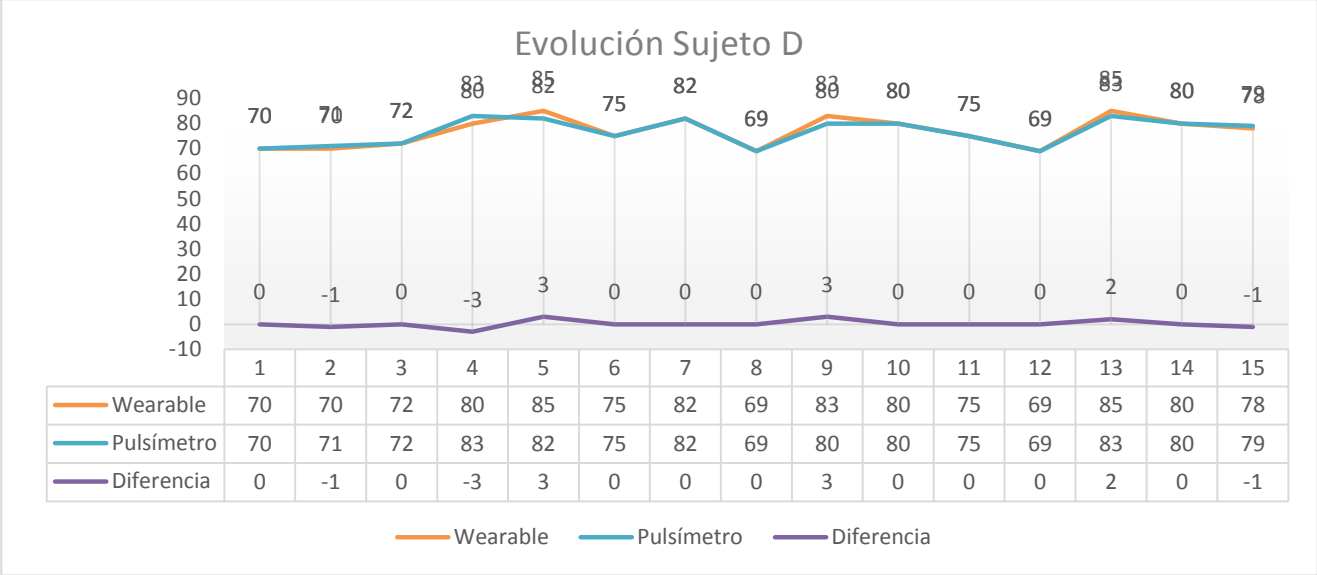
Sujeto D

Sujeto D

Sexo	Masculino
Edad	70 años

Wearable	Pulsímetro	Diferencia
80	80	0
75	76	-1
70	70	0
69	69	0

72	73	-1
67	68	-1
72	71	1
80	80	0
69	69	0
72	72	0
95	94	1
95	91	4
87	87	0
69	70	-1
70	70	0



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Wearable	70	70	72	80	85	75	82	69	83	80	75	69	85	80	78
Pulsímetro	70	71	72	83	82	75	82	69	80	80	75	69	83	80	79
Diferencia	0	-1	0	-3	3	0	0	0	3	0	0	0	2	0	-1

Problemas encontrados con Sujeto D

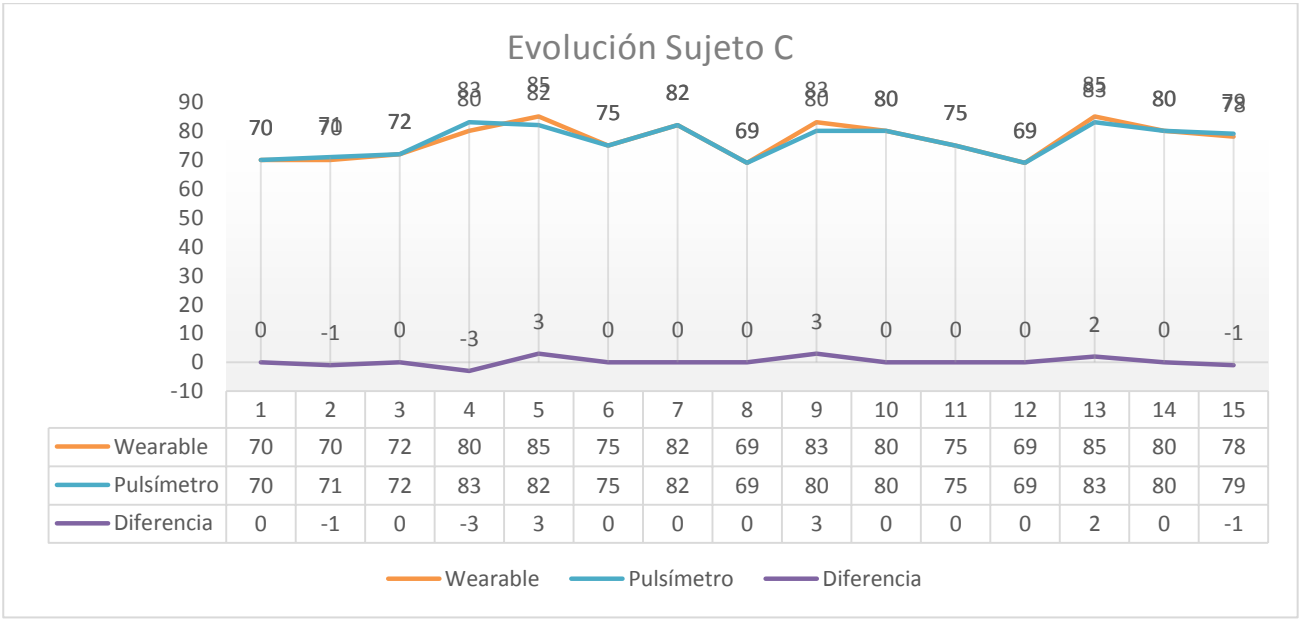
No hubo problemas con este sujeto

Sujeto E

Sujeto E

Sexo	Femenino
Edad	60 años

Wearable	Pulsímetro	Diferencia
70	70	0
70	71	-1
72	72	0
80	83	-3
85	82	3
75	75	0
82	82	0
69	69	0
83	80	3
80	80	0
75	75	0
69	69	0
85	83	2
80	80	0
78	79	-1



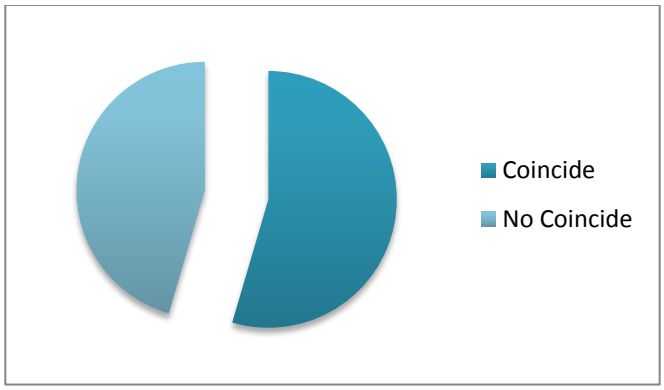
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Wearable	70	70	72	80	85	75	82	69	83	80	75	69	85	80	78
Pulsímetro	70	71	72	83	82	75	82	69	80	80	75	69	83	80	79
Diferencia	0	-1	0	-3	3	0	0	0	3	0	0	0	2	0	-1

Problemas encontrados con Sujeto E

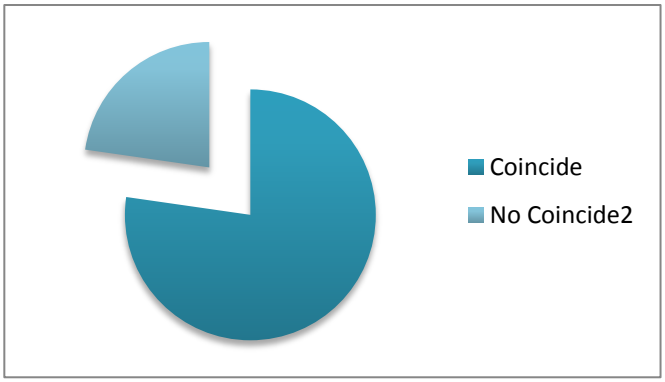
No hubo problemas con este sujeto salvo los comentados en otro caso de prueba de error en medición que se observan al final.

Análisis de los resultados

Tras observar y juntar todos los datos, he podido observar que de los 66 datos recogidos, el 54% (36) de los datos son acertados mientras que el 45% (30) restante, no. Esto puede parecer una gran diferencia pero hay que tener en cuenta que el error máximo de pulsaciones por minuto oscila entre 1 y 4 pulsaciones lo que supone un gran éxito.

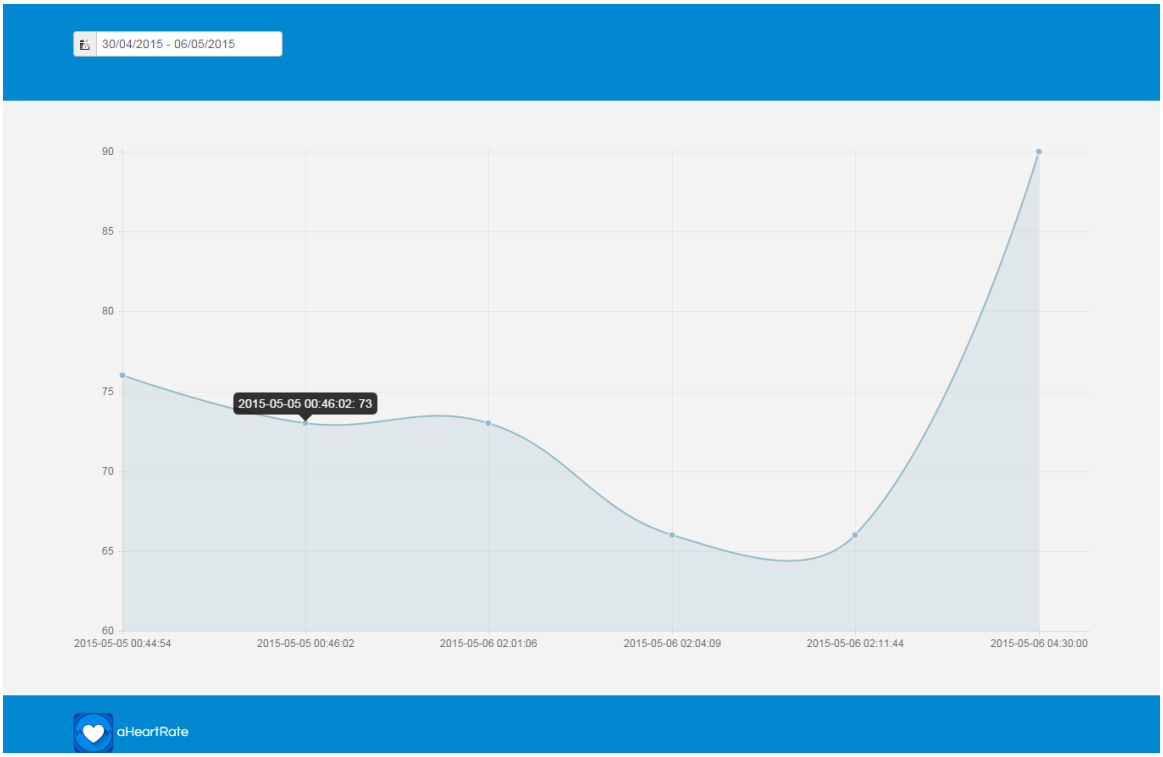


Si al comparar estableciéramos como criterio el que el error fuera +-1, el porcentaje de aciertos ascendería a un total de 77% de aciertos (51) frente a 22% de “fallos” (15) lo que da un acercamiento aún mayor al ofrecido por lo que se considera un pulsímetro profesional. Esto no indica que se pueda usar con fines médicos sustituyendo en su totalidad al pulsímetro, pero sí será bastante fiable para hacer estimaciones.

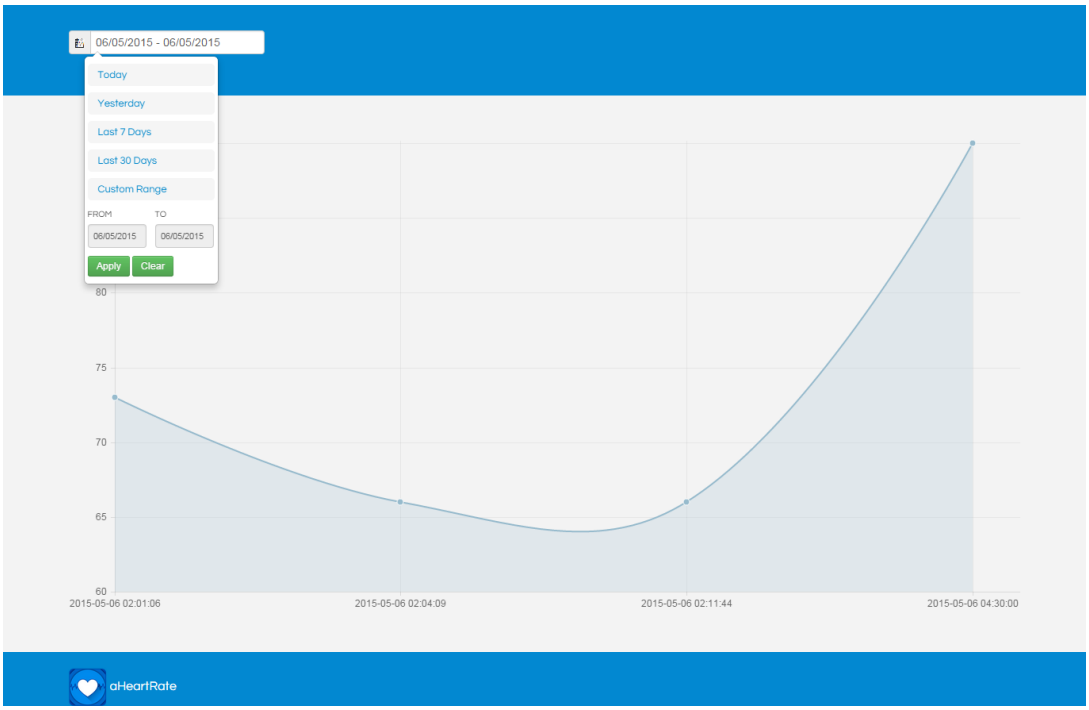

Coincide
No Coincide2

Servidor web

El servidor web fue probado bajo un entorno Linux con los servicios LAMPP (Linux , MySQL , PHP y PhpMyAdmin). Para conectar la aplicación en el entono de prueba con el servidor web se debe realizar una redirección del tráfico desde el eclipse/intelij a la IP 10.0.0.2 por el puerto 80. Esto, internamente, apuntará a localhost/127.0.0.1 . El modo de enviar los datos es mediante POST. Estos se reciben y mediante un ORM son mapeados con una clase y guardados en la base de datos MySQL. De momento carece de sistemas de seguridad (aunque el propio ORM incluye algunos para evitar ataques de inyección SQL, XSS y CRF). Después los datos son mostrados como veremos a continuación usando las librerías jQuery de chartjs[17].



23 Datos en versión web mostrados con la librería chartjs



24 Datos en versión web (2)

Líneas futuras

Una vez finalizada la entrega del presente proyecto, como líneas futuras del mismo, se esperan las siguientes metas de futuro:

- Realizar una mejora en el código de la aplicación, optimizando tanto código como modulación.
- Integrar la aplicación con otros dispositivos wearables como pudiera ser el mencionado moto360.
- Dotar al sistema de una plataforma multi-usuario de modo que varios wearables puedan estar interconectados con el dispositivo handheld para elaborar mejores informes.
- Realizar investigaciones en base a los resultados obtenidos.
- Interconexión con otros dispositivos y otras tecnologías para ofrecer mejores resultados en los datos.

Presupuesto

	Coste (€)
Samsung Gear Live	200€
Transporte	40€
Desarrollo de aplicaciones	
Horas de desarrollo estimadas (300)	4500
Horas de investigación (50)	750
Total	5490

Bibliografía y fuentes

[1] Centro de Innovación BBVA .: http://www.centrodeinnovacionbbva.com . Accedida en Marzo de 2015

[2] Miguel Ángel Moreno Álvarez .: Desarrollo de aplicaciones Android Seguras. Ed 0xWord (2015) http://0xword.com/es/libros/35-libro-desarrollo-aplicaciones-android.html

[3] Bruce Eckel .: Piensa en Java 5ª edición (2014)

[4] Ada Framework https://github.com/mobandme/ADA-Framework

[5] Dalvik VM: http://es.wikipedia.org/wiki/Dalvik Accedida en Marzo de 2015

[6] Android Studio https://developer.android.com/sdk/index.html

[7] IntelijIdea https://www.jetbrains.com/idea/

[8] Material Design http://www.google.com/design/

[9] Git : http://es.wikipedia.org/wiki/Git

[10] Bitbucket bitbucket.org

[11] Repositorio: https://bitbucket.org/moilodsan/tfg

[12] Appinventor: http://appinventor.mit.edu/explore/

[13] Samsung Gear Live: http://www.samsung.com/global/microsite/gear/gearlive_design.html

[14] Moto360: https://moto360.motorola.com/

[15] Ripple Background: https://github.com/skyfishjy/android-ripple-background

[16] Alarm Manager: http://developer.android.com/reference/android/app/AlarmManager.html

[17] ChartJS: http://www.chartjs.org/

[18] Wireless ECG Monitoring System using Mobile Platform PandaBoard http://ijiset.com/v1s9/IJISET_V1_I9_84.pdf

[19] OpenHardwareExG http://openelectronicslab.github.io/OpenHardwareExG/

[20] e-Health Sensor Platform V2.0 for Arduino and Raspberry Pi [Biometric / Medical Applications] http://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical