Universidad Central "Marta Abreu" de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Implementación de una aplicación para el procesamiento de electroencefalogramas en dispositivos móviles

Autor: Andy Daniel Navarro Taño

Tutor: DrC. Carlos Bazán Prieto

Santa Clara

2014

"Año 56 de la Revolución"

Universidad Central "Marta Abreu" de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Implementación de una aplicación para el procesamiento de electroencefalogramas en dispositivos móviles

Autor: Andy Daniel Navarro Taño

Tutor: DrC. Carlos Bazán Prieto

Email: cabazan@uclv.edu.cu

Santa Clara

2014

"Año 56 de la Revolución"

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Telecomunicaciones y Electrónica, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del Tutor

Firma del Jefe de Departamento
donde se defiende el trabajo

Firma del Responsable de
Información Científico-Técnica

PENSAMIENTO

Nunca andes por el camino trazado,

pues te conducirá únicamente hacia donde los otros fueron

Alexander Graham Bell

DEDICATORIA

Dedicada a toda mi familia. Muy especialmente a mi hermana por comprender y compartir todos mis sueños, a mi sobrinita Anly de apenas 4 meses de vida. A mi madre por haber sido incansable en sus regaños y sobre todo a mi padre, quien deseó siempre este logro mucho más que yo, y al cual hice sufrir miles de bochornos durante toda mi vida estudiantil.

AGRADECIMIENTOS

Ofrezco mis más sinceros agradecimientos a todas aquellas personas que de una forma u otra han contribuido en mi formación profesional, pues el concluir los estudios universitarios nunca es un logro únicamente personal. Especialmente quisiera agradecer a mi familia por todos estos años de sacrificio, a los amigos entrañables que encontré en esta etapa estudiantil y a los brillantes profesores que tuve en el centro. Muy especialmente deseo agradecer al profesor Carlos Bazan Prieto, mi tutor, quien mantuvo hacia mí un carácter afable y de buena voluntad, me sirvió de guía, y me brindó su apoyo incondicional aun en situaciones en las que no lo merecía.

TAREA TÉCNICA

- Estudio de las características de los sistemas operativos para dispositivos móviles
- Elección del sistema operativo móvil a usar como plataforma
- Estudio del método general de desarrollo de aplicaciones para el sistema operativo a utilizar, atendiendo los problemas de rendimiento
- Revisión bibliográfica de los algoritmos de codificación por subbandas y la codificación Golomb
- Estudio del formato de archivo digital de los electroencefalogramas
- Elección y estudio de las alternativas existentes para graficar señales sobre el sistema operativo a utilizar

Firma del Autor	Firma del Tutor

RESUMEN

Un electroencefalograma (EEG) es un método clásico no invasivo utilizado para registrar la actividad eléctrica del cerebro a través de electrodos situados en la superficie del cráneo. Mediante el estudio de los electroencefalogramas se realizan diagnósticos e investigaciones de enfermedades cerebrales. Estos generan elevados volúmenes de información lo cual constituye un inconveniente para el almacenamiento, procesamiento o transmisión, sobre todo cuando los equipos utilizados son ambulatorios, por lo que generalmente se incluyen esquemas de compresión para reducir el volumen de información a procesar.

En la actualidad existe una elevada disponibilidad de dispositivos móviles de uso personal que disponen de una alta capacidad de procesamiento. Es por esto que dichos dispositivos pueden ser útiles en el desarrollo de tareas complejas pertenecientes a diversas ramas de la ciencia, y en el campo de la medicina se les ha dado ya disímiles aplicaciones. En el presente trabajo se desarrolló una aplicación para el sistema operativo móvil Android que posibilita la visualización y compresión de electroencefalogramas.

Para la compresión de estas señales existen una serie de métodos de bajo costo computacional y en este trabajo se eligió el algoritmo Golomb sobre la base de las elevadas tasas de compresión y calidad de la señal reconstruida que este presenta, además de que el mismo fue desarrollado e implementado en nuestro centro.

TABLA DE CONTENIDOS

PENSAMIENTO i

DEDICATORIA ii

AGRADECIMIENTOS iii

TAREA TÉCNICA iv

RESUMEN v

INTRODUCCIÓN 1

CAPÍTULO 1. Electroencefalogramas y dispositivos móviles 4

Introducción: 4

1.1 Electroencefalograma 4

1.1.1 ¿Qué es un electroencefalograma? 4

1.1.2 Necesidad de dispositivos móviles en el procesamiento de electroencefalogramas 5

1.1.3 Necesidad de compresión para registros encefalográficos 6

1.2 Métodos de compresión 6

1.2.1 Clasificación 6

1.2.2 Parámetros para evaluar la calidad de los métodos de compresión 8

1.2.3 Método de compresión utilizado 10

1.3 Dispositivos móviles 10

1.3.1 Dispositivos móviles (Clasificación) 10

1.3.2 Elección del tipo de dispositivo para implementar la aplicación.....11
1.3.3 Sistemas operativos para dispositivos móviles (Smartphone).....11
1.3.4 Carácterísticas principales de Android 12
Conclusiones del Capítulo 15
CAPÍTULO 2. IMPLEMENTACIÓN DE LA APLICACIÓN 16
Introducción.....16
2.1 Implementación de la aplicación mediante el lenguaje Java 16
2.1.1 El lenguaje de programación Java 16
2.1.2 Entorno de desarrollo Eclipse.....17
2.2 El sistema operativo Android.....18
2.2.1 Arquitectura 18
2.2.2 Anatomía de una aplicación Android 21
2.2.3 Ciclos de vida de una aplicación Android. 24
2.3 El formato de datos europeo (EDF) 27
2.3.1 Lectura de las señales contenidas en los archivos de electroencefalogramas usando el lenguaje Java.....28
Incrementar el rendimiento en el proceso de visualización.....29
Obtención de fragmentos de la señal 29
2.4 Compresión de señales de electroencefalogramas 30
2.5 Conclusiones del capítulo 30
CAPÍTULO 3. FUNCIONAMIENTO DE LA APLICACIÓN.....31
3.1 Instalando una aplicación en Android manualmente 31
3.2 Manual de usuario de la aplicación.....32
Iniciando 32
Abriendo archivos y visualizando la información de los mismos 33

Visualizando señales del electroencefalograma 34

Opciones de visualización incluidas 36

Compresión y exportación de señales 38

3.3 Conclusiones del capítulo 40

CONCLUSIONES 41

RECOMENDACIONES 42

REFERENCIAS BIBLIOGRÁFICAS 43

INTRODUCCIÓN

El estudio de los electroencefalogramas (EEG) tiene un importante valor en el diagnóstico de múltiples enfermedades y trastornos cerebrales, y contribuye al tratamiento de sus padecimientos. Entre estas enfermedades y trastornos pueden citarse la epilepsia, los trastornos del sueño: insomnio, hipersomnia, parasomnia; y los trastornos del ritmo circadiano [1-5].

La grabación o registro de los electroencefalogramas genera grandes cantidades de datos, por ejemplo, un equipo digital que registre 32 canales durante 12 horas con una frecuencia de muestreo de 256 Hz y una resolución de 12 bits por muestras, demandaría capacidades de transmisión y almacenamiento de 98.4Kb/s y 531MB respectivamente. El valor elevado de la información contenida en los electroencefalogramas y los volúmenes que representa, hace que la transmisión y almacenamiento de estas sea un importante aspecto a tener en cuenta a la hora de diseñar los sistemas afines. Reducir el número de datos que representa la señal utilizando el mínimo de recursos posibles, es un reto que puede garantizar la implementación en equipos con muy bajo consumo de potencia y limitaciones de hardware, como son los dispositivos portátiles de uso personal. Esta es una temática que recibe en la actualidad mucha atención, especialmente, por la elevada disponibilidad que existe de estos dispositivos. [6]

Este trabajo tiene como objetivo desarrollar una aplicación que permita procesar electroencefalogramas en dispositivos móviles de uso personal. Esta aplicación posibilitará la visualización y compresión de dichas señales en estos dispositivos. Un programa de este tipo puede resultar muy útil para ciertos especialistas de las ciencias médicas como pueden ser los neurólogos, quienes podrán disponer de los electroencefalogramas de sus pacientes

en todo momento siempre y cuando estos posean un dispositivo móvil que cuente con el sistema operativo Android.

La mayoría del equipamiento médico para el registro de los electroencefalogramas en Cuba es analógico; pero como las nuevas tendencias de fabricación se desarrollan en favor de las tecnologías digitales, esta aplicación podrá ser difundida y utilizada por los especialistas quienes se podrán beneficiar de las ventajas que ofrece la misma. De ahí que el presente trabajo tenga como problema científico la inexistencia en Cuba de una aplicación de este tipo.

Para dar cumplimiento a nuestro objetivo general es necesario cumplir con los siguientes objetivos específicos:

- Diseñar una interfaz de usuario amigable y consistente con las funcionalidades de la aplicación.
- Implementar la funcionalidad de la aplicación en el lenguaje de programación Java.
- Desarrollar un sistema de visualización de señales que permita mostrar pequeños fragmentos de la misma con posibilidad de carga y visualización asincrónica de nuevos fragmentos.
- Integrar posibilidades de compresión de señales a la aplicación.

Organización del informe:

El informe se divide en introducción, capitulario, conclusiones y bibliografía. En el Capítulo 1: "Electroencefalogramas y dispositivos móviles" se explica la importancia de reducir los datos que representan los electroencefalogramas y la necesidad de contar con aplicaciones en dispositivos de uso cotidiano para el procesamiento, almacenamiento y visualización de los mismos; además se realiza una introducción al sistema operativo sobre el cual será desarrollada la aplicación. En el Capítulo 2: "Implementación de la aplicación", se describen los materiales y métodos que se utilizaron para el desarrollo de la misma. En el Capítulo 3: "Funcionamiento de la aplicación", se exponen los resultados y funcionalidades obtenidas después de haberse programado dicha aplicación. Por último, en las conclusiones del trabajo, se exponen y se valoran las características presentes en la aplicación así como los aportes que brinda la misma. Al final se ofrecen recomendaciones basadas en las deficiencias y carencias presentes.

CAPÍTULO 1. Electroencefalogramas y dispositivos móviles

Introducción:

En este capítulo se expone la necesidad de poder procesar electroencefalogramas en dispositivos de uso personal y cotidiano como pueden ser los teléfonos inteligentes o smartphones, los cuales a pesar de contar con un bajo consumo de potencia presentan altas capacidades de procesamiento computacional. Para esto se da una breve explicación sobre los electroencefalogramas y sus aplicaciones. Se expone además la necesidad de contar con métodos de compresión para estos registros debido a sus elevados volúmenes de información y se explican de forma general los parámetros utilizados para evaluar la calidad de los mismos. Por último se analizan los sistemas operativos para dispositivos móviles existentes y se exponen las características del sistema operativo móvil elegido para el desarrollo de nuestra aplicación, además de que se da una explicación del por qué se ha decidido seleccionar dicho sistema operativo.

1.1 Electroencefalograma

1.1.1 ¿Qué es un electroencefalograma?

Un electroencefalograma (EEG) es un método no invasivo mediante el cual se registra la actividad eléctrica en el cerebro (ondas cerebrales), a través de electrodos colocados superficialmente en determinados puntos sobre el cuero cabelludo. La amplitud de las ondas cerebrales depende del nivel de sincronismo entre las neuronas. En esencia, es captar y registrar mediante electrodos la suma en el tiempo de los potenciales pos sinápticos generados por las neuronas en la corteza cerebral. Estas ondas manifiestan un

comportamiento oscilatorio y variable en el tiempo que difiere en dependencia de la posición en que estén colocados los electrodos.[7]

Aunque en la actualidad existen técnicas muy novedosas que permiten estudiar la actividad cerebral como son, por ejemplo, la tomografía por emisión de positrones (PET Positron Emission Tomography), las resonancias magnéticas (MRI Magnetic Resonance Imaging), entre muchas otras que proporcionan imágenes en tres y dos dimensiones con buena resolución espacial, el estudio electroencefalográfico sigue siendo una técnica de diagnóstico muy útil pues brinda una excelente resolución temporal. Esta técnica además requiere bajos costos para su utilización [8].

Aplicaciones del EEG

Los estudios electroencefalográficos son usados en diversos campos dentro de la medicina para realizar múltiples diagnósticos. Mediante el estudio de los EEG se pueden distinguir trastornos psiquiátricos funcionales de trastornos producidos por lesiones orgánicas cerebrales. El EEG ha contribuido también a la investigación de la naturaleza del sueño. Por ejemplo en los estudios polisomnográficos, basados en el análisis del conjunto de señales biológicas, se le otorga mayor importancia a los electroencefalogramas.

El EEG es además una de las principales pruebas para diagnosticar epilepsia así como para determinar el tipo y la localización de los ataques. Los estudios de los EEG también son utilizados en las interfaces cerebro computadora, en la monitorización en tiempo real durante operaciones quirúrgicas y para la monitorización en las salas de cuidados intensivos. Además se pueden estudiar zonas de la corteza cerebral encargadas de determinadas funciones al estimular los nervios aferentes sensitivos, como el ojo (a través de la luz) o el oído (con el sonido).[1-5]

1.1.2 Necesidad de dispositivos móviles en el procesamiento de electroencefalogramas

El vertiginoso desarrollo de las tecnologías digitales ha facilitado cada vez más la reducción de los componentes electrónicos y elevado sus prestaciones, allanando así, el

camino a los equipos portátiles. Con dispositivos móviles que cuenten con alguna aplicación para el procesamiento y almacenamiento de EEG se pueden implementar métodos de análisis para dichas señales que detecten trastornos y anomalías automáticamente. De esta forma se podría disminuir la probabilidad de que el especialista pase por alto algún aspecto de importancia. Otra ventaja radica en la comodidad a la hora de almacenar y transmitir los datos, en caso de que el dispositivo cuente con capacidad de conexión a alguna red.

1.1.3 **Necesidad de compresión para registros encefalográficos**

Debido a la larga duración de los registros EEG y los elevados volúmenes de datos que generan existe la necesidad de reducir los mismos sin dañar la información que representan. Un equipo digital de EEG típicamente puede registrar entre 32, 64 o incluso 128 canales EEG durante períodos que pueden superar las 24 horas, generando 32, 64, o 128 MB de datos por hora y tasas de transmisión de 75, 150 y 300 Kbps. En los estudios del sueño por ejemplo, se requiere del almacenamiento durante varias horas de múltiples canales. Esto implica que el volumen de datos generado exija capacidades de almacenamiento y transmisión considerables en los equipos de análisis y registro de EEG [9].

Cuando los equipos son portátiles, el consumo de potencia representa un elemento importante a considerar debido a que estos no cuentan con una fuente de energía permanente sino que deben ser alimentados por baterías. El consumo de potencia se puede minimizar si se reducen la cantidad de datos de la señal pues con esto se logra reducir la potencia de transmisión y los requerimientos de almacenamiento. Además la reducción de los datos facilita la manipulación de los mismos.

1.2 Métodos de compresión

1.2.1 Clasificación

Los algoritmos de compresión se clasifican en dos grandes grupos de acuerdo a la compresión que logran[10-15]:

- Algoritmos que no incluyen pérdidas o casi sin pérdidas

- Algoritmos de compresión con pérdidas.

Esta clasificación está basada en la comparación de la señal original y la señal reconstruida después de la compresión. Lo ideal resulta que la señal reconstruida después de la compresión presente la menor cantidad de pérdidas de su contenido original, pero los algoritmos que no incluyen pérdidas logran bajas tasas de compresión siendo esto un inconveniente para determinadas aplicaciones en las que se requiere tener la menor cantidad de datos posible, ya sea para su almacenamiento, transmisión o procesamiento [9].

No obstante en el caso de señales biomédicas y señales en general, se puede tolerar cierta cantidad de pérdidas de información en dependencia de la señal en cuestión y la relevancia de la información que se pierde en las aplicaciones usadas o en el diagnóstico clínico posterior. En el tratamiento digital de las señales biomédicas la compresión con pérdidas es más usada debido a las altas tasas de compresión logradas. Así, el método de compresión a usar está condicionado generalmente en dependencia de la parte de la información de la cual se puede prescindir.

Otras clasificaciones de métodos de compresión se basan en las técnicas usadas para lograr la compresión dividiendo los métodos en tres grandes clases:

- Algoritmos de compresión directo: Se basan en la extracción de puntos significativos, guardando muestras que contienen información importante sobre la señal y descartando el resto.
- Algoritmos de compresión por transformación: Se basan en la estimación de la información relevante de la señal en algún dominio transformado.
- Algoritmos de compresión por extracción de características: Se basan en obtener una representación de la señal según un conjunto de parámetros (características) que definen la información relevante.

También están caracterizados por otros factores como son los requerimientos de procesamiento, los recursos de memoria y el retardo en la codificación, los que dependen de las técnicas que se emplean para realizar la compresión [9].

1.2.2 Parámetros para evaluar la calidad de los métodos de compresión

Existen algunas métricas que permiten evaluar y comparar de manera objetiva los resultados alcanzados por los distintos métodos de compresión. Para caracterizar estos métodos se tienen en cuenta fundamentalmente las tasas de compresión que logran y la similitud entre la señal reconstruida a partir de la compresión con respecto a la señal original. La similitud entre las señales, reconstruida y original, se estima mediante criterios conocidos como parámetros de calidad o de distorsión.

Tasa de compresión

La tasa de compresión representa la relación entre el total de bits de la señal original y el número total de bits de los datos de la señal comprimida [16]. Es el parámetro que generalmente se quiere optimizar en los algoritmos de compresión. Esta se puede presentar usando varias expresiones siendo la más común:

$$TC = \frac{señal\ original\ (bytes)}{señal\ comprimida\ (bytes)} \quad (1)$$

También se puede presentar normalizada y como redundancia:

Tasa de compresión normalizada:

$$TC\ normalizada = \frac{1}{TC} \quad (2)$$

Como redundancia:

$$R = \left(1 - \frac{1}{TC}\right) * 100 \quad (3)$$

Parámetros de calidad

Mediante los parámetros de calidad se establece el parecido de la señal antes y después de la compresión. La calidad de la señal puede ser medida a través de diferentes criterios que no siempre son equivalentes entre sí, entre los más utilizados se encuentran el porciento de distorsión residual (PRD) y el error medio cuadrático (RMSE) [17].

Porcentaje de Distorsión Residual (PRD)

$$PRD = \sqrt{\frac{\sum_{i=1}^{N}(yi-\breve{y}i)^2}{\sum_{i=1}^{N} yi^2}} * 100\% \quad (4)$$

Dónde:

yi (Señal original)

$\breve{y}i$ (Señal reconstruida después de la compresión)

El PRD representa una indicación global de la distorsión entre dos señales y puede utilizarse fácilmente como un parámetro para controlar la calidad del proceso de compresión pues permite establecer en qué grado se ha conservado la forma de onda original de la señal, [18] . Los métodos de compresión con pérdidas requieren un adecuado balance entre la distorsión residual (PRD) y la tasa de compresión (CR), [19] .

Error medio cuadrático (RMSE)

$$RMSE = \sqrt{\frac{1}{N} * \sum_{n=1}^{N}(x[n] - \breve{x}[n])^2} \quad (5)$$

Dónde:

x[n](Señal original)

$\breve{x}[n](Señal\ reconstruida\ después\ de\ la\ compresión)$

El error medio cuadrático normalizado RMSE, (Root mean square error), es otra medida de distorsión global y presenta el valor de amplitud del error en las condiciones absolutas que para el caso del electroencefalograma puede expresarse en las unidades de voltaje [20]; lo cual es considerablemente útil en el caso de las señales EEG.

1.2.3 Método de compresión utilizado

Para el caso de los electroencefalogramas se han utilizado disimiles métodos de compresión, [9, 21]. En la aplicación se ha empleado el algoritmo de descomposición por subbandas y codificación Golomb [15] pues el mismo fue desarrollado en nuestra facultad y ha sido evaluado y validado sobre distintas arquitecturas de hardware de bajo consumo de potencia [22] .

1.3 Dispositivos móviles

1.3.1 Dispositivos móviles (Clasificación)

Existe en la actualidad una gran cantidad de dispositivos que se pueden incluir dentro de la categoría de móviles. Los mismos se definen como dispositivos de pequeño tamaño con algunas capacidades de procesamiento, conexión permanente o intermitente a una red, y memoria limitada; diseñados específicamente para una función pero que pueden llevar a cabo otras más generales. Entre ellos se encuentran los teléfonos móviles.

Los teléfonos móviles fomentados por los cambios tecnológicos y sociales han evolucionado rápidamente y ampliado sus prestaciones introduciendo acceso a internet, cámaras de alta resolución, conexión WiFi, transmisión Bluetooth, pantallas táctiles, sensores o localización por GPS, etc. Los teléfonos móviles que incluyen esta variedad de servicios y en cierto grado cuentan con un nivel de procesamiento superior al de los teléfonos móviles típicos son denominados Smartphone cuya traducción el español sería "Teléfono Inteligente". El mercado de la telefonía móvil es en la actualidad uno de los más grandes y lucrativos, dentro de este el que tiene más crecimiento es el de los Smartphones [23] [24].

1.3.2 Elección del tipo de dispositivo para implementar la aplicación

La popularidad y demanda que ha alcanzado este tipo de dispositivos ha sido realmente exponencial [25] lo que hace que represente una propuesta atractiva utilizarlos para la implementación de la aplicación.

Además, al ser un tipo de dispositivo bastante difundido, representa una ventaja contar con aplicaciones para los mismos que brinden posibilidades de almacenamiento y transmisión de electroencefalogramas, que pueden no estar incluidas en los dispositivos específicos encargados de obtener los EEG.

Limitaciones

El hardware presente en estos dispositivos es mucho más pobre, hablando en términos de procesamiento, del que puede darse en las computadoras convencionales. Las capacidades de procesamiento o de memoria pueden ser tan distintas que marquen la diferencia entre poder o no, ejecutar una aplicación. Por esto al desarrollar alguna aplicación para estos dispositivos móviles, la misma ha de adaptarse adecuadamente a estas limitaciones de memoria y procesamiento de datos proporcionando una ejecución óptima. Constituye entonces una necesidad realizar la implementación de la aplicación de manera tal que demande la menor cantidad de procesos o recursos posibles.

1.3.3 Sistemas operativos para dispositivos móviles (Smartphone)

Los sistemas operativos de los dispositivos móviles son los encargados de controlar el dispositivo y determinar su funcionamiento, similares en principios a los sistemas operativos para ordenadores personales, tales como Windows, Linux o Max OS X con la diferencia de que al tratarse de dispositivos móviles estos sistemas operativos son más simples para adaptarse a las limitaciones de hardware de los mismos.

A diferencia de los computadores tradicionales, no existe en los dispositivos móviles un sistema operativo que domine abrumadoramente el mercado. La creciente demanda de

dispositivos móviles por usuarios a nivel mundial, especialmente la demanda de smartphones, ha creado una intensa competencia entre productores de Software como son los gigantes Google, Microsoft, y Apple, así como líderes de la industria móvil como Nokia, Research In Motion (RIM), y Palm OS, esto claramente ha provocado una diversidad de opciones en cuanto a sistemas operativos, siendo los más populares [26]:

- Android
- iPhone OS
- Symbian
- Palm OS
- Windows Phone
- BlackBerry OS

Elección del sistema operativo

Como se pudo apreciar en el mercado existe variedad de plataformas para móviles. Para implementar la aplicación se eligió el sistema operativo Android ¿Por qué se decidió utilizar Android? Porque es la plataforma que mayor cuota de mercado presenta actualmente [27] además de ser basada en Linux y tener código abierto.

El sistema operativo Android está cobrando cada día mayor importancia en el mercado de la telefonía móvil, debido a que cada día más fabricantes lo usan en sus teléfonos. Además Android proporciona a los fabricantes la posibilidad de modificarlo y adaptarlo a las características de sus dispositivos con un coste muy bajo. Con este panorama cada vez aparecen en el mercado más adaptaciones del sistema operativo hechas por los diferentes fabricantes, diferenciando así sus productos de los de la competencia.

1.3.4 Carácterísticas principales de Android

Android es un sistema operativo para dispositivos móviles basado en Linux (Software de código abierto) [28] , desarrollado por Google y por la Open Handset Alliance. Android está orientado a los Smartphone porque todas las aplicaciones de Google son accesibles a través de Internet y su uso es mayor en este tipo de dispositivos.

Estas son sus principales características [29]:

- Incluye un SDK que proporciona las herramientas necesarias para desarrollar aplicaciones en la plataforma Android utilizando el lenguaje de programación Java.
- Proporciona un entorno de desarrollo variado, incluyendo un plug-in para uno de los entornos de desarrollo más popular, Eclipse, con un emulador integrado para ejecutar y comprobar las aplicaciones.
- Dispone de un framework que permite la reutilización de componentes y gráficos.
- Soporta formatos comunes de audio, video e imagen (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
- Soporta telefonía GSM (dependiente del hardware), Bluetooth, EDGE, 3G, y WiFi (dependiente del hardware).
- Soporta cámara, GPS y brújula.
- Viene con un motor de navegación integrado (Web kit), el mismo que utilizan los Mac o los IPhone.
- Viene con un conjunto de aplicaciones básicas (cliente de correo electrónico, calendario, mapas, navegador, etc).
- Las aplicaciones se ejecutan sobre una máquina virtual (Dalvik) optimizada para requerir poca memoria y poder ejecutar varias instancias simultáneamente sin que el dispositivo se ralentice. Esto es importante porque en Android cada aplicación se ejecuta en un proceso separado que utiliza su propia instancia de la máquina virtual.
- Los ejecutables tienen la extensión .dex, una versión optimizada de los .class.

Versiones del sistema operativo

Android ha visto numerosas actualizaciones desde su liberación inicial. Estas actualizaciones al sistema operativo base, típicamente, arreglan fallos previos y agregan nuevas funciones. Generalmente cada actualización del sistema operativo Android es desarrollada bajo un nombre en código de un elemento relacionado con postres.

Versiones [30] :

- 1.0 Liberado el 23 de septiembre de 2008.

- 1.1 Liberado el 9 de febrero de 2009.
- 1.5 (Cupcake) Basado en el kernel de Linux 2.6. 27. Liberado el 30 de abril de 2009. 1.6 (Donut) Basado en el kernel de Linux 2.6.29.
- 2.0/2.1(Eclair) Basado en el kernel de Linux 2.6.29. Liberado el 26 de octubre de 2009.
- 2.2 (Froyo) Basado en el kernel de Linux 2.6.32.
- 2.3 (Gingerbread) Basado en el kernel de Linux 2.6.35.7 Actual en smat. Liberado el 6 de diciembre de 2010.
- 3.0 / 3.1 / 3.2 (Honeycomb). Mejor soporte para tablets.
- 4.0 (Ice CreamSandwich). Versión que unifica el uso en cualquier dispositivo, tanto en teléfonos, tablets, televisores, netbooks, etc.

Conclusiones del Capítulo

Teniendo en cuenta la elevada disponibilidad de dispositivos móviles que existe y que estos presentan altas capacidades de procesamiento computacional a partir de un bajo consumo de potencia, se tomó la decisión de utilizar un medio de este tipo para procesar electroencefalogramas de manera ambulatoria. Esto le puede brindar beneficios a los especialistas médicos ya que los mismos podrán disponer en todo momento de los electroencefalogramas de sus pacientes.

Después de analizar todas las alternativas de sistemas operativos para dispositivos móviles se tomó la decisión de emplear Android pues el mismo es el que mayor cuota de mercado presenta y con esto se incrementan las posibilidades de que una mayor cantidad de usuarios se beneficien del mismo.

CAPÍTULO 2. IMPLEMENTACIÓN DE LA APLICACIÓN

Introducción

En este capítulo se describen las tecnologías y herramientas que se emplearon en el desarrollo de la aplicación. Además se explica la lógica utilizada para resolver los problemas de rendimiento así como los patrones de desarrollo de las aplicaciones Android.

2.1 Implementación de la aplicación mediante el lenguaje Java

Las aplicaciones para la plataforma Android como se expuso en el primer capítulo se desarrollan utilizando el lenguaje de programación Java [31-33]. En este epígrafe se describe de manera general como se realizó la programación de la aplicación en este lenguaje. Para una comprensión más detallada basta con remitirnos al código fuente del programa ya que el mismo se encuentra documentado bajo las normas del conocido javadoc [34].

2.1.1 El lenguaje de programación Java

Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. El lenguaje es muy parecido en sintaxis a C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel como la manipulación directa de punteros a memoria. La memoria es gestionada mediante un recolector de basura lo que aísla al programador de la necesidad de hacer liberaciones explícitas de la memoria asignada dinámicamente y, de esta manera, contribuye a eliminar

las fugas de memoria que comúnmente afectan a programas desarrollados en C y C++ [35, 36].

Una de las características más peculiares de este lenguaje es la posibilidad que ofrece de crear aplicaciones independientes a la plataforma donde va a ser ejecutada. Esta peculiaridad, viene dada por la existencia de una máquina virtual denominada máquina virtual de Java o JVM (Java Virtual Machine). Al compilar un programa en Java no se ejecuta un binario directamente sobre el hardware, sino un código intermedio llamado bytecode en ficheros cuya extensión es .class.

La JVM es la encargada de traducir estos bytecodes convirtiéndolos al código particular del hardware utilizado. Para cada hardware debe haber una JVM específica, ya sea un teléfono móvil, un ordenador con Windows, o un equipo electrodoméstico etc.

Java incluye además bibliotecas que ofrecen apoyo para el desarrollo de sus aplicaciones, algunas de estas bibliotecas son:

- java.io. Biblioteca para aplicaciones básicas de entrada/salida.
- java.lang. Biblioteca de operaciones de la Máquina Virtual.
- java.util. Biblioteca para utilidades estándar.

Para programar es necesario el kit de desarrollo JDK (Java Develoment Kit); un conjunto de herramientas como son: un compilador de línea de comandos javac, la máquina virtual de Java con la que se puede ejecutar aplicaciones java, una herramienta de documentación javadoc [34], y una herramienta para empaquetar proyectos jar.

2.1.2 Entorno de desarrollo Eclipse

Para realizar la implementación en Java de la aplicación se utilizó Eclipse [37], un entorno de desarrollo integrado (IDE por sus siglas en inglés) multiplataforma y de código abierto que cuenta además con módulos para desarrollar aplicaciones Android. Eclipse es un IDE que no está orientado específicamente hacia ningún lenguaje de programación en concreto. El uso de un determinado lenguaje estará supeditado a la existencia de un módulo que le de soporte. Con la versión estándar del entorno Eclipse se distribuye el módulo necesario para programar en lenguaje Java, su nombre es JDT.

Eclipse Originalmente fue desarrollado por IBM aunque actualmente forma parte de una organización independiente llamada Fundación Eclipse, una organización sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. Eclipse se concibe entonces como un programa libre.

Características

Eclipse consta de un editor de texto con resaltado de sintaxis y realiza la compilación del código en tiempo real mostrando los errores en el código instantáneamente. Dispone además de un apartado de plug-ins para añadir diferentes controles y formas de realizar los proyectos.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. El IDE también hace uso de un espacio de trabajo, en este caso, crea un grupo de metadata en un directorio donde aloja archivos de configuración en texto plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente [38].

2.2 El sistema operativo Android

En el primer capítulo se realizó una introducción al sistema operativo Android. En el siguiente apartado se describe su arquitectura para comprender mejor su funcionamiento.

2.2.1 Arquitectura

La figura 2.1 [39] , muestra como está constituida la arquitectura de Android. Cada una de las capas del esquema utiliza servicios ofrecidos por las anteriores, y brinda a su vez los suyos propios a las capas de niveles superiores.

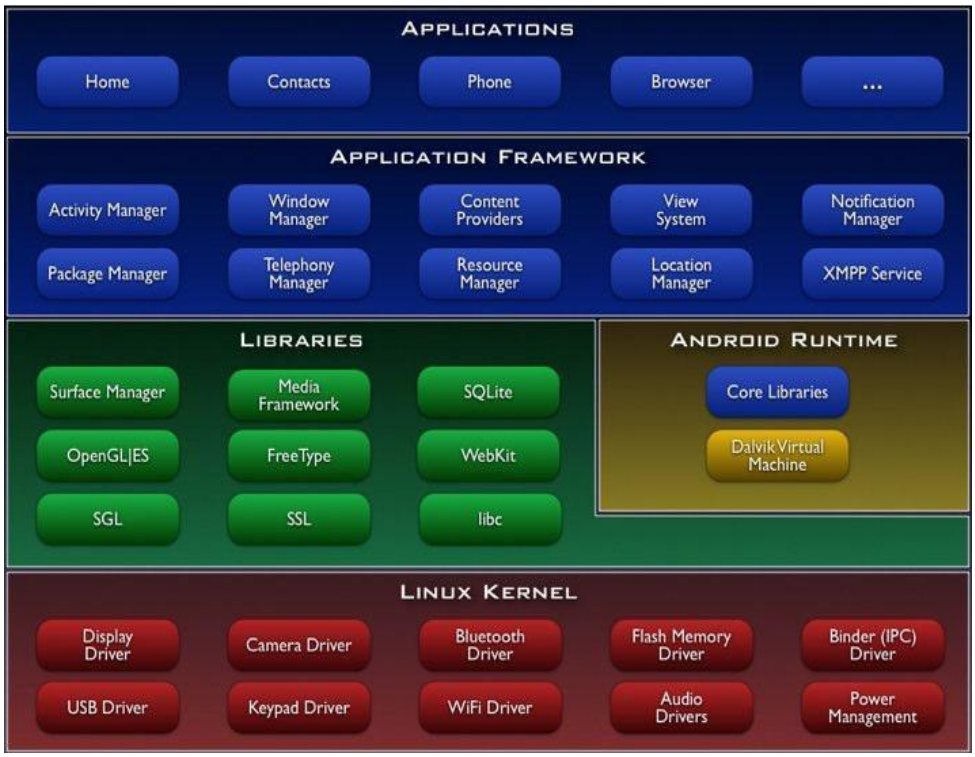

Figura 2.1. Arquitectura de Android.

Linux Kernel: Android está construido sobre la base de un sólido y probado núcleo de Linux. El núcleo actúa como una capa de abstracción entre el hardware y Android. Internamente Android usa Linux como agente de memoria, procesos, gestión de redes y otros servicios de sistemas operativos. Los usuarios nunca tendrán contacto directo con Linux y los programas no lo llamaran directamente.

Librerías (Las Bibliotecas nativas):

La próxima capa sobre el kernel que contiene Android son las bibliotecas nativas. Android incluye un set de librerías escritas en lenguaje C/C++ usadas por varios componentes del sistema. Estas capacidades se exponen a los desarrolladores a través del framework de aplicaciones de Android, el cual interactúa con las librerías mediante JNI (Java Native Interface). Algunas son: System C library (implementación librería C estándar), librerías de medios, librerías de gráficos, 3d, bases de datos SQLite, entre otras.

Runtime de Android: Esta capa también se encuentra justo sobre el kernel de Linux. Incluye un set de librerías base que proveen la mayor parte de las funcionalidades

disponibles en las librerías base del lenguaje de programación Java, e incluye la máquina virtual Dalvik que es creación propia de ingenieros de Google. No se utiliza así la Java Virtual Machine (JVM) está bajo licencia de Sun Microsystems y tiene restricciones.

Dalvik está pensada y diseñada para que utilice poca memoria y pueda ejecutar varias instancias simultáneamente. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual. Dalvik ejecuta archivos en el formato Dalvik Executable(.dex), el cual está optimizado como se dijo para memoria mínima.

Framework de aplicaciones: Por encima de las librerías y el runtime se encuentra el framework de aplicaciones, esta capa proporciona los bloques de constructores de alto nivel que se usan para crear las aplicaciones. Los desarrolladores tienen acceso completo a las interfaces de programación de aplicaciones (APIs) del framework usado por las aplicaciones base. La arquitectura está diseñada para simplificar el reúso de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Éste mismo mecanismo permite que los componentes sean reemplazados por el usuario.

Las partes más importantes del Framework de aplicaciones son las siguientes:

- Activity Manager: Controlan el ciclo de vida de las aplicaciones.
- Content Providers: Estos objetos encapsulan los datos que son necesarios compartir entre aplicaciones.
- Resource Manager: Proporciona acceso a recursos que no son código como pueden ser gráficos, cadenas de texto etc.
- Notification Manager: Permite a las aplicaciones mostrar alarmas personalizadas en la barra de estado.

La capa de aplicaciones

Es la capa más alta en el diagrama de arquitectura de Android. Los usuarios finales verán sólo estos programas, despreocupados de toda la acción que sigue debajo de este nivel o capa. Las aplicaciones base incluidas por lo general en los dispositivos, son cliente de email, programa de SMS, calendario, mapas, navegador, contactos, y otros. Todas las aplicaciones están escritas en el lenguaje de programación Java.

2.2.2 Anatomía de una aplicación Android

Dentro de una aplicación de Android hay cuatro componentes principales: Activitity, Listeners, Servicios y Content Providers. Todas las aplicaciones de Android están formadas por algunos de estos elementos o combinaciones de ellos.

Activity.

Las Activities (o Actividades) son el elemento constituyente de Android más común. Para implementarlas se utiliza una clase por cada Actividad que extiende de la clase base Activity. Cada clase mostrará una interfaz de usuario, compuesta por Views (o Vistas). Cada vez que se cambie de Vista, se cambiará de Actividad, como por ejemplo en una aplicación de mensajería que se tiene una Vista que muestra la lista de contactos y otra Vista para escribir los mensajes. Cuando cambiamos de Vista, la anterior queda pausada y puesta dentro de una pila de historial para poder retornar en caso necesario. También se pueden eliminar las Vistas del historial en caso de que no se necesiten más. Para pasar de vista en vista, Android utiliza una clase especial llamada Intent.

Intents.

Un Intent es un objeto mensaje y que, en general, describe que quiere hacer una aplicación. Las dos partes más importantes de un Intent son la acción que se quiere realizar y la información necesaria que se proporciona para poder realizarla, la cual se expresa en formato de identificador uniforme de recursos (URI) [40]. Un ejemplo seria ver la información de contacto de una persona, la cual mediante un Intent con la acción ver y la URI que representa a esa persona se podría obtener. Relacionado con los Intents hay una

clase llamada IntentFilter que es una descripción de que Intents puede una Actividad gestionar. Mediante los IntentFilters, el sistema puede resolver Intents, buscando cuales posee cada actividad y escogiendo aquel que mejor se ajuste a sus necesidades. El proceso de resolver Intents se realiza en tiempo real, lo cual ofrece dos beneficios:

- Las actividades pueden reutilizar funcionalidades de otros componentes simplemente haciendo peticiones mediante un Intent.
- Las actividades pueden ser remplazadas por nuevas actividades con IntentFilters equivalentes.

Listeners

Los Listeneres se utilizan para reaccionar a eventos externos (por ejemplo, una llamada). Los Listeners no tienen interfaz gráfica pero pueden utilizar el servicio Notification Manager para avisar al usuario. Para lanzar un aviso no hace falta que la aplicación se esté ejecutando, en caso necesario, Android la iniciará si se activa el Listeners por algún evento.

Servicios

Un Servicio es básicamente un código que se ejecuta durante largo tiempo y sin necesidad de interfaz gráfica, como puede ser un gestor de descarga en el cual se indican los contenidos a descargar y posteriormente el usuario puede acceder a una nueva Vista sin que el gestor se interrumpa. En caso de que haya múltiples servicios a la vez, se les puede indicar diferentes prioridades según las necesidades.

Content Provider.

En Android, las aplicaciones pueden guardar su información en ficheros, bases de datos de tipo SQLite [41, 42]…Pero en caso de que lo que se quiera sea compartir dicha información con otras aplicaciones, lo necesario es un Content Provider. Un Content Provider es una clase que implementa un conjunto estándar de métodos que permite a otras aplicaciones guardar y obtener la información que maneja dicho Content Provider.

AndroidManifiest.

En Android existe un archivo de tipo lenguaje extensible de marcado (XML) [43] llamado AndroidManifest que, aunque no forme parte del código principal de la aplicación, es necesario para su correcto funcionamiento. Este archivo es el fichero de control que le dice al sistema que tiene que hacer con todos los componentes anteriormente mencionados en este apartado que pertenecen a una aplicación en concreto.

Tipos de aplicación de Android.

Un *AndroidPackage(.apk)* es el fichero que contiene el código de la aplicación y sus recursos, y que posteriormente se instala en el dispositivo para poder ejecutar la aplicación.

Tareas.

Una tarea en Android es lo que el usuario ve como una aplicación y el desarrollador ve como una o más Actividades donde el usuario interacciona y va pasando de Vista en Vista. Dentro de las tareas, una actividad toma el papel de punto de entrada (será la primera en mostrarse cuando se ejecute la aplicación) y las demás, si hay, formarán parte de la misma tarea, a la espera de ser instanciadas.

Procesos.

En Android, los procesos se ejecutan a nivel de kernel y el usuario normalmente no tiene constancia de ellos. Todo el código de la aplicación se suele ejecutar en un proceso dedicado pero también se puede especificar si sólo se quiere que se ejecute en el proceso una determinada clase o componente de la aplicación. Los principales usos de los procesos son:

- Mejorar la estabilidad o seguridad de las aplicaciones.
- Reducir la sobrecarga de proceso ejecutando el código de múltiples aplicaciones en el mismo proceso.
- Ayudar al sistema a gestionar los recursos separando partes de código pesado en un proceso separado que puede ser eliminado independientemente de otras partes de la aplicación.

Hilos

Android evita la creación de hilos adicionales por parte de un proceso, manteniendo la aplicación en un sólo hilo al menos que no los cree la propia aplicación. Esto repercute de manera importante en las llamadas a instancias a Actividades, Listeners y Servicios, ya que sólo pueden ser hechas por el hilo principal del proceso en el que están corriendo. Por otro lado, al no crearse un hilo por cada instancia, dichas instancias no deben realizar operaciones largas o bloqueantes cuando son llamadas, de lo contrario, bloquearían todos los demás componentes del proceso.

2.2.3 Ciclos de vida de una aplicación Android.

Cada aplicación de Android corre en su propio proceso, el cual es creado por la aplicación cuando se ejecuta y permanece hasta que la aplicación deja de trabajar o el sistema necesita memoria para otras aplicaciones. Una característica fundamental de Android es que el ciclo de vida de una aplicación no está controlado por la misma aplicación sino que lo determina el sistema a partir de una combinación de estados como pueden ser que aplicaciones están funcionando, que prioridad tienen para el usuario y cuanta memoria queda disponible en el sistema. De esta manera, Android sitúa cada proceso en una jerarquía según la importancia que tengan para el sistema, como se puede ver a continuación.

- **Proceso foreground**: Proceso (activity) en primer plano con el que interactúa el usuario. Este proceso será el último que eliminará el sistema.
- **Proceso visible**: Proceso (activity) que no está en primer plano pero sí es visible (por ejemplo, un cuadro de dialogo). Este proceso solo se elimina en estados de memoria muy críticos al igual que los procesos foreground.
- **Proceso background**: Proceso (activity) que no está visible en pantalla. Si se eliminara no tendría una repercusión directa en el usuario.
- **Proceso vacío**: Proceso que no está asociado con ningún componente visualmente activo. El sistema los elimina con frecuencia y puede mantenerlos vivos si hay memoria suficiente para mejorar el tiempo de activación de otro componente de la

aplicación. Activities, intent receivers o services son componentes que tienen un impacto considerable en el tiempo de ejecución de las aplicaciones.

Estructura de un proyecto Android.

Al crear un proyecto Android, en el workspace de Eclipse se genera una nueva carpeta con el nombre de dicho proyecto. Esta carpeta contiene una serie de subcarpetas y de ficheros necesarios para poder generar posteriormente la aplicación. Esta estructura será común a cualquier aplicación, independientemente de su tamaño y complejidad. En este apartado se intentará dar una visión general del significado y cometido de cada uno de estos elementos.

Cuando se crea un proyecto Android, se puede observar en el workspace de Eclipse una jerarquía similar a la mostrada en la figura 2.2. A continuación se detallan cada uno de sus elementos.

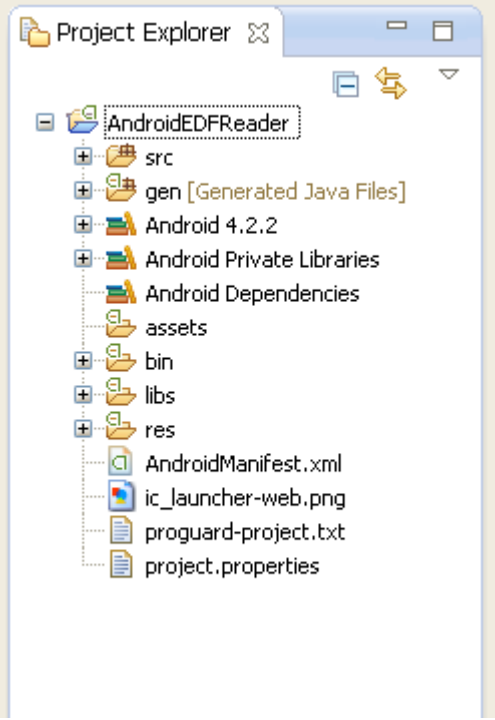

Figura 2.2. Arquitectura de Android.

Carpeta /src

Contiene los archivos fuente .java del proyecto. Utiliza la misma jerarquía de carpetas que la indicada por el nombre del paquete que se haya asignado. Por ejemplo, en el caso de la aplicación de ejemplo "AndroidEDFReader" se especificó el nombre de paquete "uclv.fie.androidedfreader".

Carpeta /gen.

Contiene una serie de elementos de código generados automáticamente al compilar el proyecto. Cada vez que generamos nuestro proyecto, la maquinaria de compilación de Android genera por nosotros una serie de ficheros fuente en Java, dirigidos al control de los recursos de la aplicación.

Carpeta /res.

Contiene todos los ficheros de recursos necesarios para el proyecto: videos, imágenes, cadenas de texto, etc. Los diferentes tipos de recursos se deberán distribuir entre las siguientes subcarpetas:

- /res/drawable/. Contienen las imágenes de la aplicación. Se puede dividir en /drawable-ldpi, /drawable_mdpi/, y /drawable_hpdi/ para utilizar diferentes recursos en dependencia de la resolución del dispositivo.
- /res/layout/ Contienen los ficheros de definición de las diferentes pantallas de la interfaz gráfica. Se puede dividir en /layout y /layout-land para definir distintos layouts dependiendo de la orientación del dispositivo.
- /res/anim/ Contiene la definición de las animaciones utilizadas por la aplicación.
- /res/menu/ Contiene la definición de los menús de la aplicación.
- /res/values/ esta carpeta tendrá ficheros XML que declaran valores de diferentes tipos. Por ejemplo, cadenas de texto, colores predefinidos, arrays de elementos,

dimensiones, estilos, etc. En cierta medida, los recursos aquí localizados pueden verse como declaraciones de variables que serán accedidas después desde el código.

- /res/xml/ Contiene los ficheros XML utilizados por la aplicación, XML es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium.
- /res/raw/ Contiene recursos adicionales, normalmente en formato distinto a XML que no se incluyan en el resto de carpetas de recursos.

Debido a que el diseño de una pantalla a través de código puede resultar muy complejo, Android soporta una sintaxis XML para diseñar pantallas. Android define un gran número de elementos especialmente hechos para la plataforma, cada uno de ellos representando una subclase específica de Android View. Se puede diseñar una pantalla de la misma forma que se diseña en HTML, es decir, como una serie de etiquetas anidadas y guardadas como se expuso anteriormente en un archivo XML dentro del directorio "res/layout/" de la aplicación.

Carpeta /bin.

Esta carpeta contiene los archivos binarios del proyecto generados a partir de los archivos fuente. Al igual que la carpeta “/src", se mantiene la misma jerarquía de subcarpetas que la representada en el nombre del paquete.

Fichero AndroidManifest.xml.

Este archivo contendrá la definición en XML de los principales aspectos de la aplicación como son por ejemplo su identificación (nombre, icono, etc), su contenido (pantallas, mensajes, entre otros).

2.3 El formato de datos europeo (EDF)

El formato de datos europeo (EDF) [44] es un formato simple y flexible para el intercambio y almacenamiento de señales biológicas y físicas de varios canales. Fue desarrollado por un grupo de ingenieros biomédicos europeos en el año 1987 y hecho público en el año 1992.

Es un formato abierto y no propietario y es utilizado para archivar, intercambiar y analizar los datos generados por dispositivos comerciales en un formato que es independiente del sistema de adquisición. De esta manera, los datos pueden ser recuperados y analizados por el software independiente. Presenta una gran variedad de software capaz de interpretarlo así como de archivos de muestra, muchos de los cuales se encuentran disponibles gratuitamente. Es el formato utilizado en la grabación de los electroencefalogramas por parte de los equipos médicos.

2.3.1 Lectura de las señales contenidas en los archivos de electroencefalogramas usando el lenguaje Java

Como se ha mencionado anteriormente existen muchas alternativas de software para la interpretación de estos formatos de archivos desde distintos tipos de lenguajes de programación. En la aplicación para realizar la tarea de leer las señales de los electroencefalogramas contenidas en archivos de extensión .edf para su posterior procesamiento fue empleada la clase EDFRead. El código fuente de esta clase se puede encontrar en el archivo de nombre EDFRead.java existente en el directorio src/EDF del proyecto. Entre los diferentes métodos y funciones públicas que la clase brinda disponemos de una en específico de nombre “getSignal”. La misma recibe un único argumento de tipo entero que sirve para indicar la señal que se desea obtener; este número debe estar en el intervalo de 1 a 23 ya que los electroencefalogramas disponen de 23 canales. Una vez que se le indica a la función mencionada un número de señal válido, la misma nos devolverá una matriz de valores de tipo “short” los cuales representan los diferentes valores de amplitud de la señal en diferentes instantes de tiempo.

Anteriormente se mencionaba que los electroencefalogramas se generaban a partir de grabaciones que se les hacían a los pacientes durante largos períodos de tiempo y que los mismos podían durar hasta 12 y 24 horas. Además, estas grabaciones se hacían a altas frecuencias de muestreo por lo que los registros obtenidos contenían elevados tamaños de almacenamiento. Para ilustrar lo antes mencionado con un ejemplo podemos decir que contamos con una señal de muestra de 4 horas de duración y la misma presenta 3686400 muestras con una razón de muestreo de 256 muestras en un segundo.

En Java, el tipo de datos “short” presenta un tamaño de 2 bytes de almacenamiento, por lo que al calcular el tamaño de esta señal, tenemos:

3686400 * 2 = 7372800 bytes = 7,03 MB

Como se puede ver, cargar esta señal en memoria es un proceso bien costoso computacionalmente y si se está empleando un dispositivo móvil Android de bajo consumo de potencia ocurre que el sistema operativo bloquea el proceso al detectar que se está realizando una tarea demasiado tiempo.

Incrementar el rendimiento en el proceso de visualización

Al visualizar señales de electroencefalogramas ocurre que las dimensiones de la pantalla en cuanto a la anchura del dispositivo con el que se visualiza son mucho menores que el tamaño de la señal, por tanto, en todo momento lo que se está mostrando es solo una pequeñísima parte de la señal y para ver el resto es necesario hacer desplazamientos horizontales sobre la misma. Normalmente en el proceso de visualización el primer paso consiste en cargar en memoria todos los datos de la señal para interpretarlos y posteriormente mostrarlos en pantalla. Resulta mucho más eficiente poder cargar en memoria solo los datos de la señal que se están mostrando y no todos los datos de la señal con el objetivo de ahorrar memoria y procesos en el inicio de la visualización. Para cargar solo la mínima información necesaria y aumentar así el rendimiento en el proceso de visualización, la señal se divide en fragmentos de pequeño tamaño que resultan mucho más fáciles de procesar computacionalmente. Inicialmente siempre se va a cargar el primer fragmento y a medida que sea necesario mostrar otras partes de la señal, se van cargando asincrónicamente los fragmentos correspondientes.

Obtención de fragmentos de la señal

Para dar solución al problema ilustrado anteriormente, fue necesario desarrollar una nueva función a la clase EDFRead que devolviera fragmentos de la señal y no toda la señal. Esta nueva función tiene por nombre “getSignalFragment” y acepta 2 argumentos, el primero indica la señal que se desea obtener y el segundo el índice del fragmento. Todos los

fragmentos tienen 1 segundo de duración por lo que en el caso de la señal anterior tendrán 256 valores cada uno y ocuparán 512 bytes cuyo tamaño es perfectamente manipulable por cualquier dispositivo Android.

2.4 Compresión de señales de electroencefalogramas

Como se ha explicado ya anteriormente, resulta muy conveniente emplear compresión en este tipo de señales debido al gran tamaño que presentan las mismas. Para llevar a cabo esta tarea en nuestra aplicación se ha utilizado una implementación del método de descomposición por subbandas y compresión Golomb desarrollada en java por el estudiante Jorge Pablo Sánchez Pérez, quien en el año 2012 y como parte de su trabajo de diploma, [22] validara la implementación de este método sobre diferentes arquitecturas de hardware de bajo consumo de potencia.

2.5 Conclusiones del capítulo

Para realizar el desarrollo de la aplicación primeramente fue necesario analizar la filosofía de programación en lenguaje Java utilizando el entorno de desarrollo integrado Eclipse. Además de esto se estudió la arquitectura del sistema operativo así como la estructura de archivos de un proyecto Android.

Después de haber realizado la implementación en lenguaje Java de la aplicación, usando todas las herramientas de desarrollo para el sistema operativo Android se llegó a la conclusión de que realizando un proceso de visualización con carga asincrónica de fragmentos, se evita el bloqueo del sistema que se produce al intentar cargar en memoria una señal completa de un electroencefalograma. De esta forma se resuelve el problema de rendimiento que ello implica.

CAPÍTULO 3. FUNCIONAMIENTO DE LA APLICACIÓN

Una vez que se ha finalizado la fase de desarrollo de la aplicación es posible hacer uso de la misma a través de la sencilla e intuitiva interfaz de usuario que presenta. En este epígrafe se detallan los pasos a seguir para llevar a cabo la instalación en un dispositivo Android y además se brinda el manual de usuario como primera referencia de consulta para aprender a usar dicha aplicación a la vez que se demuestran las funcionalidades presentes.

3.1 Instalando una aplicación en Android manualmente

Para realizar la instalación de nuestra aplicación es necesario hacerlo de forma manual. Para esto debe estar activada la opción de permitir instalar aplicaciones desde orígenes desconocidos que se encuentra en Ajustes->Seguridad->Orígenes desconocidos. Para acceder a los ajustes del sistema operativo se hace ejecutando una aplicación de nombre "Ajustes" que se encuentra dentro de la pestaña "Aplicaciones". Si dicha opción no está activada por defecto es necesario hacerlo. Una vez hecho esto, bastará con abrir nuestro selector de archivos y desplazarnos hasta el directorio donde se encuentren los ficheros de nombre "FileManager-2.0.2.apk" y "AndroidEDFReader.apk". Seguidamente se deben ejecutar ambos ficheros e indicar que se desea realizar la instalación de los mismos. El sistema operativo será el encargado de indicarnos si la instalación ha sido fallida o satisfactoria.

3.2 Manual de usuario de la aplicación

Iniciando

Una vez que se encuentre instalada correctamente la aplicación en el sistema operativo se podrá observar que un nuevo ícono como el mostrado en la figura 3.1 ha aparecido en la sección de aplicaciones y cuyo nombre es “AndroidEDFReader”.

Figura 3.1. Ícono de la aplicación

Para ejecutar dicha aplicación bastará con presionar sobre este ícono y seguidamente será mostrada en pantalla la actividad principal de la aplicación la cual se representa en la figura 3.2

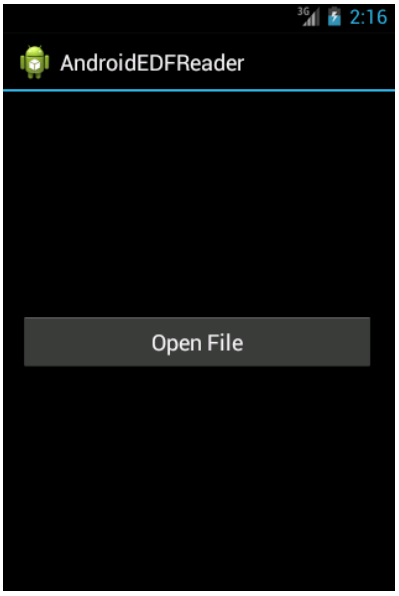

Figura 3.2. Actividad principal

Abriendo archivos y visualizando la información de los mismos

Una vez que se haya iniciado la aplicación será necesario elegir algún archivo de electroencefalograma para ser leído por la aplicación. Para esto es necesario pulsar sobre el botón "Open File" y seguidamente será ejecutado un selector de archivos para que el usuario pueda ubicar el archivo que desea leer ya que este se puede encontrar en diferentes ubicaciones como pueden ser la memoria interna, una memoria USB o una tarjeta SD. El selector de archivos presenta una interfaz similar al conocido explorador de Windows por lo que su comprensión es simple. La figura 3.3 muestra el aspecto de este selector de archivos al abrirlo por primera vez.

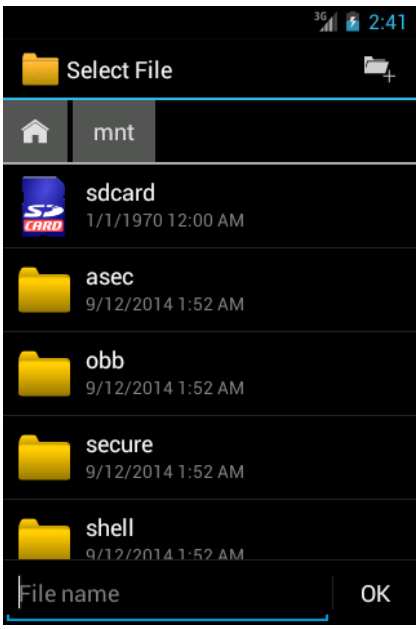

Figura 3.3. Selector de archivos

Una vez que se ubique el archivo deseado basta con presionarlo y seguidamente pulsar el botón "Ok". Después de esto el archivo será leído internamente y mostrada su información en una nueva actividad tal y como se puede ver en la figura 3.4

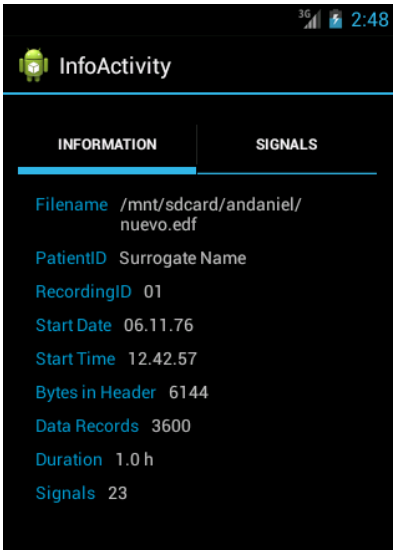

Figura 3.4. Información de un electroencefalograma

En esta información se brindan datos pertenecientes a la cabecera del electroencefalograma como el nombre del paciente, la fecha de realización, tiempo de duración, entre otros.

Visualizando señales del electroencefalograma

Como se puede observar la actividad donde se muestra la información del registro se encuentra dividida en 2 pestañas. La primera cuyo nombre es “Information”, se encuentra activa por defecto y es la encargada de mostrar la información referente al registro. La segunda pestaña llamada “Signals”, que se presenta en la figura 3.5, contiene el listado de todas las señales presentes en el registro.

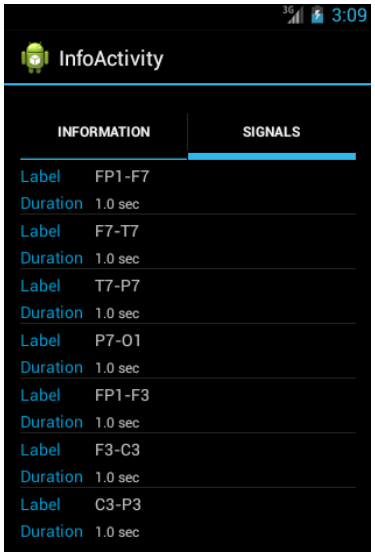

Figura 3.5. Señales contenidas en el registro

Si una de estas señales es presionada una nueva actividad será ejecutada mostrando inicialmente información sobre dicha señal tal y como se muestra en la figura 3.6

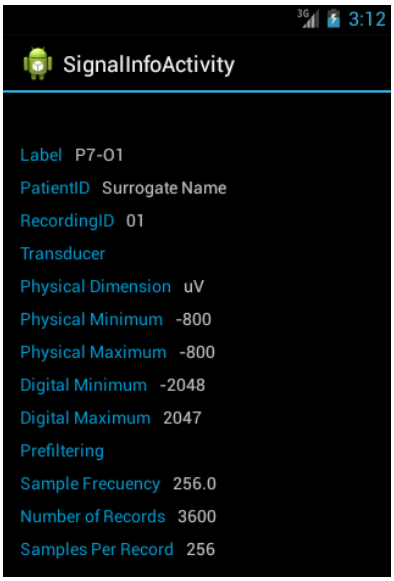

Figura 3.6. Información de una señal contenida en un registro

Para visualizar dicha señal se elige el submenú “Plot” del menú principal de la actividad en curso. Hecho esto se cargará en pantalla una nueva actividad semejante a la mostrada en la

figura 3.7 donde se podrá visualizar la gráfica de dicha señal y se dispondrá de varios controles de navegación.

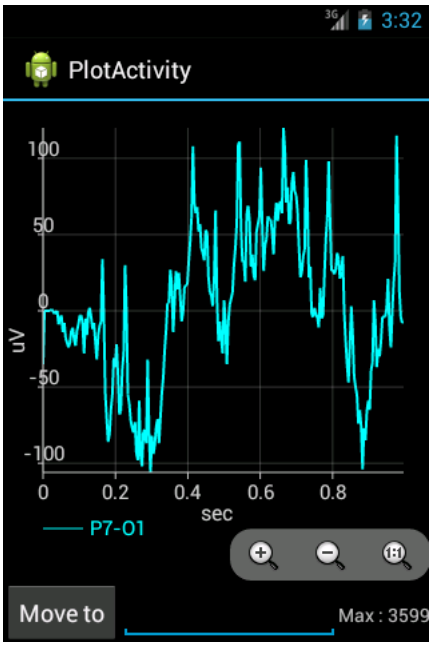

Figura 3.7. Gráfica de una señal

Sobre la gráfica de la señal es posible hacer acercamientos, alejamientos, desplazamientos cortos haciendo arrastres directamente sobre la señal, o desplazamientos largos indicando el instante de tiempo en segundos y presionando “Move to”.

Opciones de visualización incluidas

Para aumentar las posibilidades de la visualización se cuenta con 2 opciones extras a las que se puede acceder en el menú de dicha actividad. La primera se nombra “Show/Hide Signal Values” y consiste en mostrar u ocultar los valores de amplitud en los diferentes puntos de la gráfica. La segunda lleva por nombre “Plot New Signal” y consiste en añadir una nueva visualización de señal sobre la actual gráfica. Esto facilita hacer comparaciones visuales entre señales. En las figuras 3.8 y 3.9 se muestran ambas opciones de visualización.

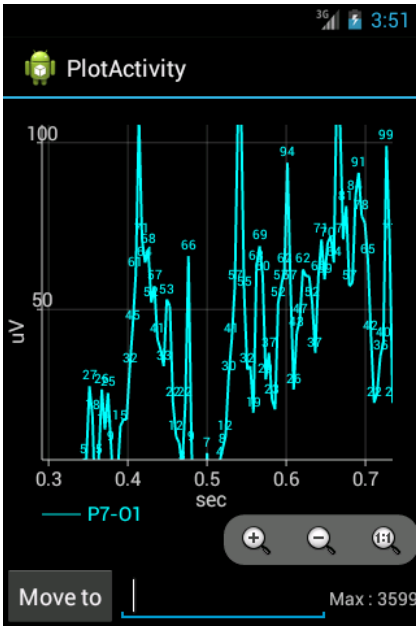

Figura 3.8. Visualización de los valores de amplitud sobre la señal.

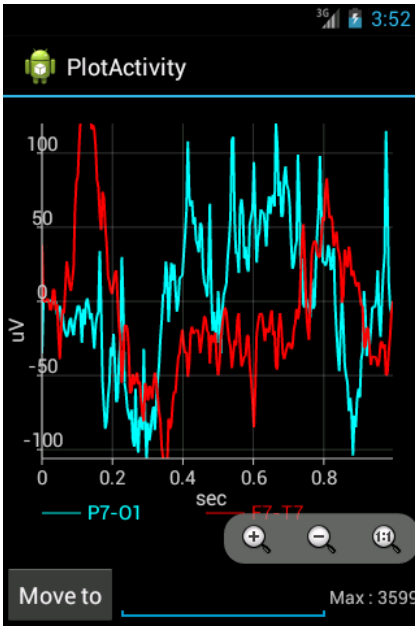

Figura 3.9. Visualización de varias señales.

Compresión y exportación de señales

A las señales presentes en los registros es posible exportarlas como ficheros independientes para un posterior análisis y esta exportación se puede hacer con compresión o sin ella. Para exportar una señal comprimida se elige la opción “Export Compressed Signal” del menú principal donde además se encuentran las opciones “Export Signal” y “Plot”. Seguidamente se elige el algoritmo de compresión a utilizar, por defecto la aplicación trae implementado el algoritmo de descomposición por subbandas y codificación Golomb [15] pero existen muchos otros que se podrían implementar sobre dicha aplicación, es por esto que existe este diálogo de selección aunque actualmente solo exista una posibilidad como se muestra en la figura 3.10.

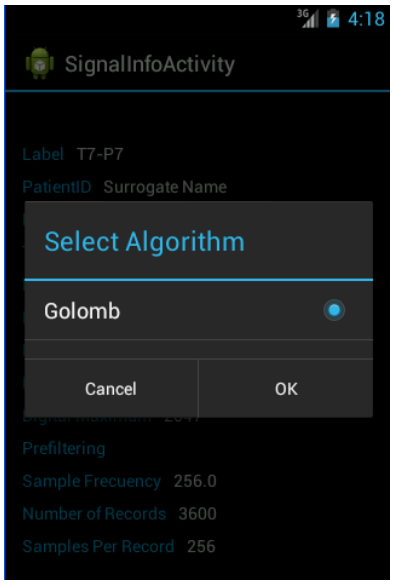

Figura 3.10 Elección del algoritmo de compresión

Luego de elegir el algoritmo de compresión a emplear es necesario definir los parámetros de dicho algoritmo así como la ubicación en la que se desea guardar el archivo comprimido. En la figura 3.11 se muestra la actividad donde se definen los parámetros del algoritmo de compresión utilizado.

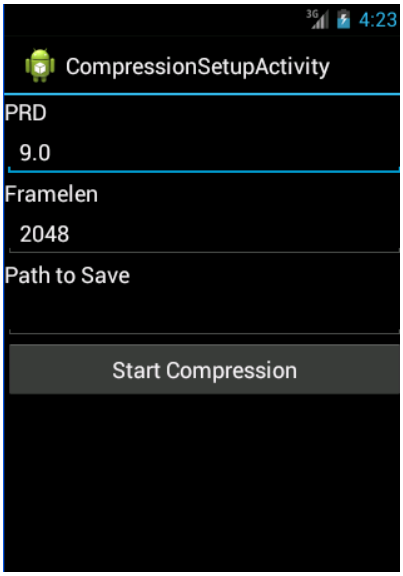

Figura 3.11. Parámetros del algoritmo de compresión por subbandas y codificación Golomb.

Para definir la ubicación donde se desea guardar el archivo comprimido se presiona en "Path to Save" y se ejecuta el selector de archivos para elegir el directorio. Una vez definidos todos los parámetros correctamente, basta con presionar el botón "Start Compression" y se lanzará una tarea sobre un hilo independiente en el que se llevará a cabo el proceso, como se muestra en la figura 3.12.

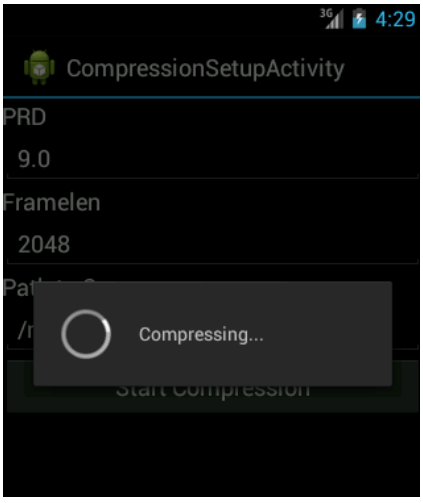

Figura 3.12. Visualización del proceso de compresión

3.3 Conclusiones del capítulo

Después de haberse concluido la implementación de la aplicación resulta conveniente destacar que la misma presenta una sencilla e intuitiva interfaz gráfica de usuario, por lo que su comprensión no debe presentar dificultad alguna para los usuarios. Se pudieron visualizar las señales completas sin que se bloqueara el dispositivo al detectar la ejecución de una tarea de larga duración, y por tanto, se considera eficiente el procesamiento de los electroencefalogramas en la misma. Además de esto, la aplicación implementa la compresión de estas señales por lo que se logra la reducción del volumen de datos y por tanto se aprovecha mejor la capacidad de almacenamiento del dispositivo.

CONCLUSIONES

Luego que se implementara la aplicación AndroidEDFReader sobre el sistema operativo móvil Android, fue posible procesar con eficiencia registros de señales de electroencefalogramas en dicho sistema operativo lo cual no era posible hasta el momento debido a los grandes volúmenes de datos que estos presentan y a la limitación en cuanto a consumo de potencia que tienen los dispositivos móviles. Además, se extendieron las capacidades del sistema operativo en cuanto a aplicaciones médicas y se pudo llegar a las siguientes conclusiones:

1. Se dispone de una aplicación para dispositivos móviles, orientada al procesamiento de electroencefalogramas que maneja eficientemente un volumen de información elevado.
2. Los resultados presentados demuestran la validez de la programación Java para estas aplicaciones.
3. Se dispone de un manual de usuario que permite instalar y utilizar las facilidades de la aplicación presentada.
4. Se implementa la función de compresión de señales de electroencefalogramas

RECOMENDACIONES

Para futuros trabajos en este tema, se deben tomar en cuenta las siguientes recomendaciones:

- Culminar la implementación del bloque decodificador para poder visualizar directamente las señales comprimidas.
- Programar la asociación de los formatos de archivos soportados con la aplicación, con el objetivo de que cuando se desee abrir un archivo de este tipo automáticamente se ejecute la aplicación cargando el archivo en cuestión. Actualmente es necesario abrir la aplicación y después ubicar el archivo.
- Diseñar íconos identificativos para el programa y sus formatos de archivo.
- Desarrollar la traducción al español de las cadenas de texto.

REFERENCIAS BIBLIOGRÁFICAS

1. Salazar, A.M., et al., *Utilidad del electroencefalograma de superficie en la evaluación prequirúrgica de los pacientes con epilepsia refractaria. Comparación de dos métodos.* Revista de Especialidades Médico-Quirúrgicas, 2006. **11**(3): p. 49-53.
2. Morillo, L.E. and V. González, *Utilidad de la gammagrafía cerebral y del electroencefalograma para descubrir lesiones hemisféricas focales.* Colomb. méd, 1985. **16**(2): p. 54-61.
3. Falip, M., et al., *Epilepsia generalizada idiopática. Utilidad de la semiología y del electroencefalograma para su clasificación.* Rev neurol, 2004. **39**(11): p. 1001-1005.
4. Mila, R.A.S., *Utilidad del electroencefalograma de superficie en los pacientes con epilepsia del lóbulo temporal.* Indice de Autores: p. 123.
5. Suástegui-Román, R.A., S. Garza-Morales, and M. Pérez-Ramírez, *Utilidad y costo del electroencefalograma. Experiencia de 1 000 casos en un hospital de tercer nivel en México.* Bol Med Hosp Infant Mex, 2007. **64**(3): p. 171-181.
6. Index, C.V.N., *Global mobile data traffic forecast update, 2010-2015.* White Paper, February, 2011.
7. Schlögl, A., *The electroencephalogram and the adaptive autoregressive model: theory and applications*. 2000: Shaker Germany.
8. Velasco, M.B. *Tratamiento de Señales Biomédicas*. in *Máster en Mecnologías de la Información y las Comunicaciones*. 2009. Universidad de Alcala.
9. Antoniol, G. and P. Tonella, *EEG data compression techniques.* Biomedical Engineering, IEEE Transactions on, 1997. **44**(2): p. 105-114.
10. Alberto, C. and B. Prieto. *SLD104 IMPLEMENTACIÓN EN TIEMPO REAL DE ALGORITMO DE COMPRESION DE SEÑALES EEG BASADO EN COFICACIÓN DE SUBBANDAS*. in *Informática Salud 2013*. 2012.
11. Bazán-Prieto, C., et al., *Compresión de señales electroencefalográficas epilépticas y normales.* Ingeniería Electrónica, Automática y Comunicaciones, 2012. **33**(1): p. 25-32.

12. Bazán-Prieto, C., et al., *Analysis of tractable distortion metrics for EEG compression applications.* Physiological measurement, 2012. **33**(7): p. 1237.

13. Bazán-Prieto, C., et al., *Retained energy-based coding for EEG signals.* Medical engineering & physics, 2012. **34**(7): p. 892-899.

14. Bazán-Prieto, C., et al. *Electroencephalographic compression based on modulated filter banks and wavelet transform*. in *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*. 2011. IEEE.

15. Bazán-Prieto, C., et al. *Compresión de Señales EEG Basada en Descomposición en Subbandas y Codificación de Golomb*. in *V Latin American Congress on Biomedical Engineering CLAIB 2011 May 16-21, 2011, Habana, Cuba*. 2013. Springer.

16. Cárdenas-Barrera, J., Lorenzo-Ginori, and J. Rodríguez-Valdivia, *A wavelet-packet based algorithm for EEG signal compression.* Informatics for Health and Social Care, 2004. **E**: p. 15-27.

17. Bazán-Prieto, C., et al. *Analysis of tractable distortion metrics for EEG compression applications*. in *Physiological measurement*. 2011.

18. M, C.-R.F.B.-V., et al., *A low computational complexity algorithm for ECG signal compression.* Medical Engineering & Physics, 2004. **26**: p. 553-568.

19. S.N.Sarbadhikari, S.K.M.a., *Iterative function system and genetic algorithm based EEG compression.* Medical Engineering & Physics, 1997. **19**: p. 605-617.

20. C, C.-B.J.B.-P., B.-V. M, and C.-R. F, *Análisis de esquemas de compresión de EEG basa-dos en bancos de filtros modulados y transformadas Wavelet*, in *Works-hop UAH-Cuba 2010*. 2010: Santiago de Cuba.

21. Ylöstalo, J., *Data compression methods for EEG.* Technology and Health Care, 1999. **7**(4): p. 285-300.

22. Sánchez-Pérez, J.P., *Implementación de un algoritmo de compresión para señales EEG en un dispositivo móvil.* 2012.

23. i Juan, A.P. *El mercado de las telecomunicaciones*. in *Telecomunicaciones, infraestructuras y libre competencia*. 2004. Tirant lo Blanch.

24. Figueroa, L., *Smartphones: una revolución en las comunicaciones.* Realidad y Reflexión, 2011, Año. 11, nùm. 33, p. 61-72, 2011.

25. Rocamador Murillo, G., *Sistema de etiquetado en Android para la valoración de la gravedad de los accidentes de tráfico.* 2012.

26. Figueredo, O.J., *Sistemas Operativos para Dispositivos Móviles.* ENTÉRESE BOLETIN CIENTÍFICO UNIVERSITARIO, 2006: p. 74.

27. Alzueta Iturralde, J., *MetroClick: aplicación de un metrónomo para sistema operativo Android.* 2014.

28. Bovet, D.P. and M. Cesati, *Understanding the Linux kernel.* O'Reilly Media, 2006.

29. Gironés, J.T., *El gran libro de Android*. 2012: Marcombo.

30. Developers, A., *Platform versions.* Διαθέσιμο On-line στο: http://developer. android. com/resources/dashboard/platform-versions. html [Accessed: 18/04/2012], 2012.

31. Rogers, R., et al., *Android application development: Programming with the Google SDK*. 2009: O'Reilly Media, Inc.

32. Meier, R., *Professional Android 4 application development*. 2012: John Wiley & Sons.

33. Skogberg, B., *Android Application Development.* 2010.

34. Microsystems, S. *Javadoc*. Available from: java.sun.com/j2se/javadoc/.

35. Kernighan, B.W., D.M. Ritchie, and P. Ejeklint, *The C programming language*. Vol. 2. 1988: prentice-Hall Englewood Cliffs.

36. Musser, D.R., G.J. Derge, and A. Saini, *STL tutorial and reference guide: C++ programming with the standard template library*. 2009: Addison-Wesley Professional.

37. Burnette, E., *Eclipse IDE Pocket Guide*. 2005: " O'Reilly Media, Inc.".

38. Chen, Z. and D. Marx, *Experiences with Eclipse IDE in programming courses.* Journal of Computing Sciences in Colleges, 2005. **21**(2): p. 104-112.

39. *What is Android ?* ; Available from: http://developer.android.com/guide/basics/what-is-android.html.

40. Consorsium, W.W.W.; Available from: http://www.w3.org/Provider/Style/URI.

41. Owens, M. and G. Allen, *The definitive guide to SQLite*. Vol. 1. 2006: Springer.

42. Newman, C., *SQLite (Developer's Library)*. 2004: Sams.

43. Bray, T., et al., *Extensible markup language (XML).* World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210, 1998.

44. *European Data Format*. Available from: http://www.edfplus.info/specs/edf.html.