REPÚBLICA DEL ECUADOR
Escuela Politécnica Nacional
"E SCIENTIA HOMINIS SALUS"

La versión digital de esta tesis está protegida por la Ley de Derechos de Autor del Ecuador.

Los derechos de autor han sido entregados a la "ESCUELA POLITÉCNICA NACIONAL" bajo el libre consentimiento del (los) autor(es).

Al consultar esta tesis deberá acatar con las disposiciones de la Ley y las siguientes condiciones de uso:

- Cualquier uso que haga de estos documentos o imágenes deben ser sólo para efectos de investigación o estudio académico, y usted no puede ponerlos a disposición de otra persona.
- Usted deberá reconocer el derecho del autor a ser identificado y citado como el autor de esta tesis.
- No se podrá obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de licencia que el trabajo original.

El Libre Acceso a la información, promueve el reconocimiento de la originalidad de las ideas de los demás, respetando las normas de presentación y de citación de autores con el fin de no incurrir en actos ilegítimos de copiar y hacer pasar como propias las creaciones de terceras personas.

Respeto hacia sí mismo y hacia los demás.

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO INALÁMBRICO PARA EL REGISTRO DE LA FRECUENCIA CARDÍACA

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

BYRON HERNÁN MENA ACOSTA
byronmena@mail.ru

DIRECTOR: ING. CARLOS ROBERTO EGAS ACOSTA, MSc.
carlos.egas@epn.edu.ec

Quito, junio 2017

DECLARACIÓN

Yo, Byron Hernán Mena Acosta, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Byron Hernán Mena Acosta

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Byron Hernán Mena Acosta, bajo mi supervisión.

Ing. Carlos Roberto Egas Acosta, MSc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

AGRADECIMIENTOS

Agradezco a mis padres por su apoyo y paciencia, a mis familiares que estuvieron pendientes brindándome una mano y aquellas personas que pasaron a convertirse en mis mejores amigos y amigas e hicieron posible cumplir este sueño.

DEDICATORIA

Este trabajo va con especial dedicación a mis padres Nancy Acosta y Byron Mena por su apoyo incondicional, por cada vez que me ayudaron a levantarme cuando flaqueaba, apoyarme en cada una de mis extrañas iniciativas y estar a mi lado cuando los necesitaba, muchas gracias por ayudarme a terminar con este logro que sinceramente les pertenece en su totalidad a ustedes.

CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

ÍNDICE DE SEGMENTOS DE CÓDIGO

RESUMEN

En el presente trabajo de titulación se contempla la realización de un prototipo encargado de registrar la frecuencia cardíaca de una persona en una base de datos utilizando IPv4 e IEEE 802.15.4 para la transmisión de la información, y una aplicación en Java para visualizar los datos almacenados.

El trabajo consta de 4 capítulos partiendo de un análisis teórico, seguido de una descripción de las herramientas a utilizar para finalmente concluir con el diseño e implementación del prototipo.

En el primer capítulo se exponen los fundamentos teóricos de la frecuencia cardíaca, las redes inalámbricas y los distintos componentes utilizados para la elaboración del prototipo. En el segundo capítulo se expone el diseño de cada uno de los componentes que conforman el prototipo. En el tercer capítulo se exponen la implementación y pruebas de cada uno de los componentes que conforman el prototipo, así como del prototipo terminado.

En el último capítulo se exponen las conclusiones y recomendaciones de este trabajo de titulación.

PRESENTACIÓN

Se realiza un prototipo encargado de registrar la frecuencia cardíaca de una persona utilizando una red de sensores inalámbricos, cada sensor estará conectado a una persona para cumplir con el objetivo.

El valor de la frecuencia cardíaca de cada persona será enviado a un nodo Gateway, este nodo con la ayuda de una computadora almacenará el valor de frecuencia en la base de datos. Para la visualización de la información almacenada en la base de datos se desarrolló una aplicación utilizando el lenguaje de programación Java.

Con este trabajo de titulación se presenta opciones para el desarrollo de nuevas aplicaciones que permitan el registro de signos vitales de manera automática e inalámbrica.

CAPÍTULO I

MARCO TEÓRICO

Las tecnologías inalámbricas sugieren grandes cambios en la creación de redes de datos, y permiten tener un escenario favorable para las redes integradas. Estas redes al limitar el número de cables facilitan la creación de redes de comunicaciones móviles completamente distribuidas en cualquier lugar.

En las redes de sensores inalámbricos se utilizan las tecnologías de la información, las comunicaciones inalámbricas y las técnicas de miniaturización proporcionando una amplia gama de soluciones que pueden ser integradas en varios ambientes como en el sector salud.

En el presente capítulo se iniciará explicando el funcionamiento del pulso cardiaco para posteriormente dar una breve explicación de las herramientas utilizadas en el desarrollo del prototipo.

1.1 CORAZÓN Y EL APARATO CIRCULATORIO

El corazón y el aparato circulatorio también conocidos en conjunto como aparato cardiovascular participan en el transporte de nutrientes y recolección de desechos de cada una las células que conforman el cuerpo humano.

El corazón es el encargado de suministrar la presión necesaria en cada latido para impulsar la sangre y el aparato circulatorio conforma una red que permite el paso de sangre a cada uno de los tejidos.

1.1.1 CORAZÓN Y EL APARATO CIRCULATORIO

El corazón está formado por dos bombas separadas: un corazón derecho que bombea sangre a los pulmones y otro izquierdo que bombea sangre hacia los órganos periféricos, de color azul y rojo respectivamente como se muestra en la **Figura 1.1**. El ventrículo izquierdo expulsa su contenido inicialmente por la aorta[1] y a

[1] **Aorta:** Vaso principal que nace del ventrículo izquierdo del corazón.

medida que la sangre va alejándose del corazón lo hace por vasos[2] cada vez más pequeños hasta alcanzar a los distintos tejidos que conforman el cuerpo humano.

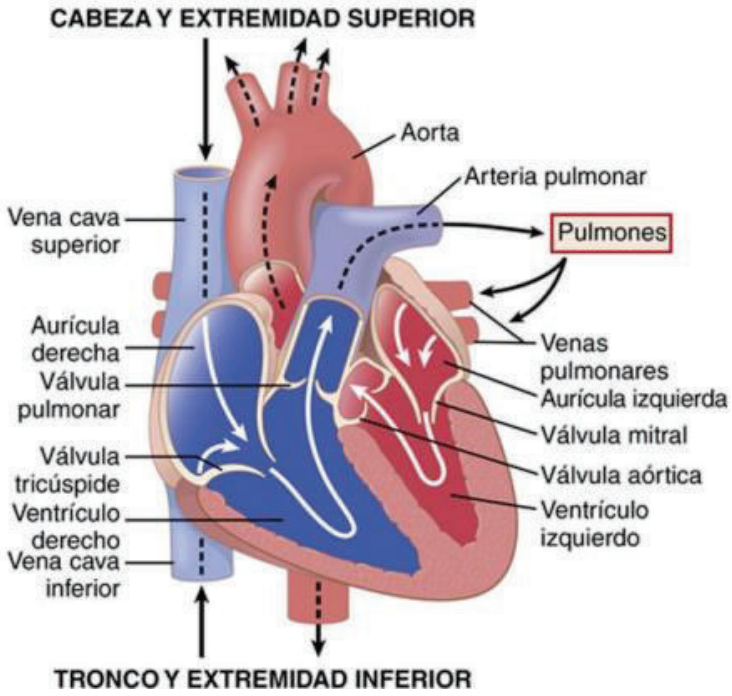


Figura 1.1 Estructura del corazón y trayecto del flujo sanguíneo por las válvulas cardíacas[1].

Las contracciones continuas del corazón toman el nombre en conjunto de ritmicidad cardíaca [1] y se deben a diversos mecanismos que contribuyen a la transmisión de potenciales de acción[3] por todo el músculo cardiaco.

1.1.2 CICLO CARDIACO

Ciclo cardiaco es el conjunto de fenómenos que son producidos desde el comienzo de un latido hasta el comienzo del siguiente.

El inicio del ciclo cardiaco está dado por la generación de un potencial de acción en el nódulo sinusal se localiza en la pared supero lateral de la aurícula derecha como se muestra en la **Figura 1.2**, próximo al orificio de la vena cava superior.

---

[2] **Vaso:** Conducto por el que circula sangre.
[3] **Potencial de acción:** Cambios rápidos del potencial celular de membrana por un intercambio de iones.

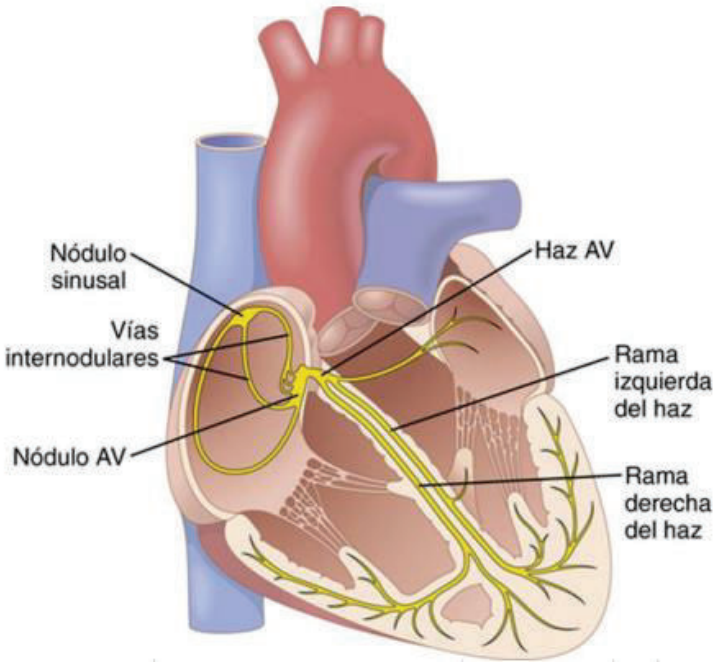


Figura 1.2 Sistema cardionector del corazón[1].

1.1.3 DIÁSTOLE Y SÍSTOLE

El ciclo cardiaco comprende un período de relajación y uno de contracción de todo el músculo cardiaco, denominados diástole y sístole respectivamente. El ventrículo izquierdo encargado de suministra sangre a la circulación periférica, durante la diástole y la sístole produce variaciones de la presión aórtica, variaciones que se distribuyen por el sistema circulatorio. Estas variaciones alteran el tamaño de los vasos y se utilizan en el campo médico para determinar la frecuencia cardíaca mediante la palpación del pulso [2].

En la **Figura 1.3** se muestran los diferentes acontecimientos que se producen durante el ciclo cardiaco, de arriba hacia abajo, las tres curvas superiores muestran los cambios de presión en la aorta, en el ventrículo izquierdo y en la aurícula izquierda, respectivamente. La cuarta curva representa los cambios de volumen en el ventrículo izquierdo, la quinta el electrocardiograma y la sexta el fonocardiograma[4].

[4] **Fonocardiograma:** Registro de los ruidos que se producen en el corazón (principalmente las válvulas cardíacas) durante su función de bombeo.

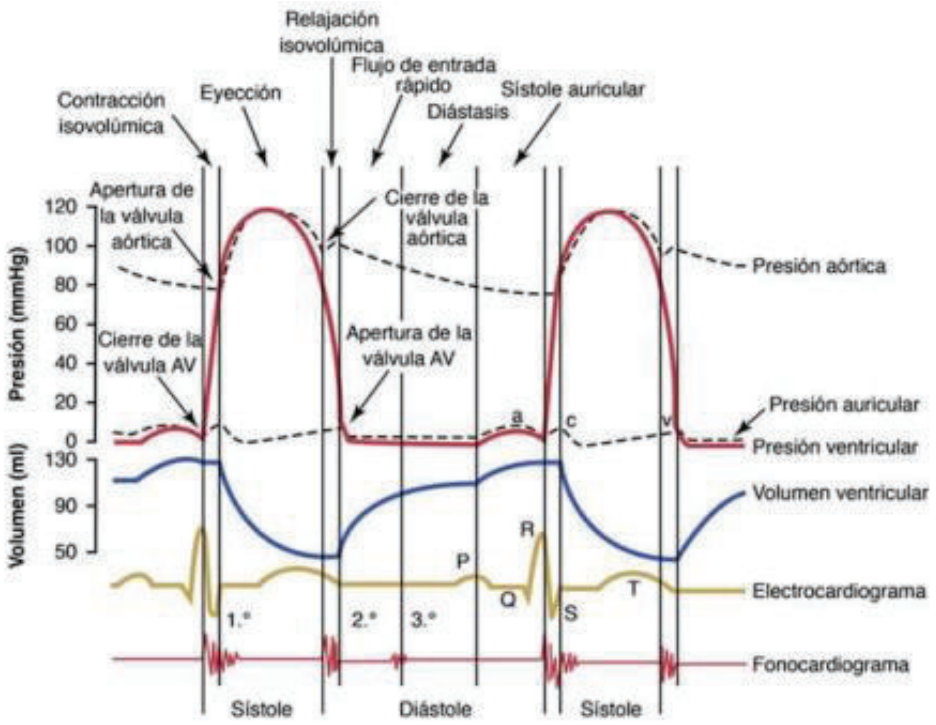


Figura 1.3 Acontecimientos del ciclo cardiaco para la función de la bomba izquierda del corazón, presión aórtica, electrocardiograma y fono cardiograma[1].

1.1.4 FISIOLOGÍA DEL MÚSCULO CARDIACO

El corazón está formado por tres tipos de músculos cardiacos: músculo auricular, músculo ventricular y fibras musculares especializadas de excitación y conducción. Los músculos auricular y ventricular poseen gran fuerza de contracción mientras que las fibras especializadas de excitación y conducción se contraen sólo débilmente debido a que poseen pocas fibras contráctiles; las fibras especializadas de excitación y conducción presentan descargas eléctricas rítmicas automáticas en forma de potenciales de acción o conducción de los potenciales de acción por todo el corazón controlando el ritmo cardiaco.

1.1.5 CONTROL DEL CORAZÓN POR LOS NERVIOS SIMPÁTICOS Y PARASIMPÁTICOS

La eficacia de la función de bomba del corazón también está controlada por los nervios simpáticos y parasimpáticos (vagos) que como se observa en la **Figura 1.4** inervan de forma abundante el corazón.

Para niveles dados de presión auricular de entrada a la cantidad de sangre que circula cada minuto (gasto cardiaco) con frecuencia se puede aumentar más de un 100% por la estimulación simpática. Por el contrario, el gasto se puede disminuir hasta un valor tan bajo como cero o casi cero por la estimulación parasimpática o también llamada estimulación vagal [3].

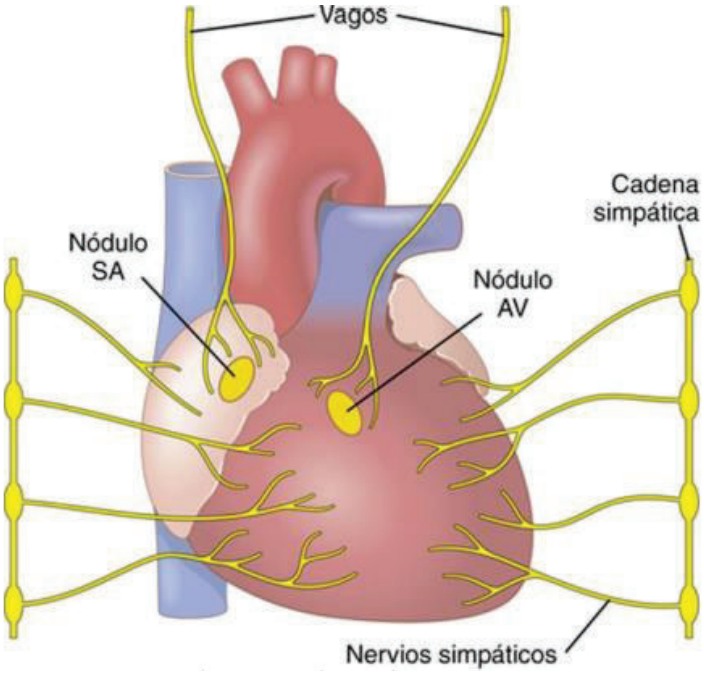


Figura 1.4 Nervios simpáticos y parasimpáticos o vagos del corazón[1].

1.1.6 MECANISMO DE EXCITACIÓN DEL CORAZÓN POR LOS NERVIOS SIMPÁTICOS

La estimulación simpática intensa puede aumentar la frecuencia cardíaca en seres humanos de adultos jóvenes desde la frecuencia normal de 60 a 100 latidos por minuto hasta 200 y raras veces, incluso 250 latidos por minuto[1]. Por el contrario, la inhibición de los nervios simpáticos del corazón puede disminuir la función de bomba del corazón en un grado moderado de la siguiente manera:

En condiciones normales, las fibras nerviosas simpáticas que llegan al corazón descargan continuamente a una frecuencia baja que mantiene el bombeo aproximadamente un 30% por encima del que habría sin estimulación simpática. Por

tanto, cuando la actividad del sistema nervioso simpático disminuye por debajo de lo normal, este fenómeno produce una reducción tanto de la frecuencia cardíaca como de la fuerza de contracción del músculo ventricular, reduciendo de esta manera el nivel de bombeo cardiaco hasta un 30% por debajo de lo normal.

1.1.7 ESTIMULACIÓN PARASIMPÁTICA O VAGAL DEL CORAZÓN

La estimulación intensa de las fibras nerviosas parasimpáticas de los nervios vagos que llegan al corazón puede interrumpir el latido cardiaco por unos segundos, pero después el corazón habitualmente "escapa" y late a una frecuencia de 20 a 40 latidos por minuto mientras continúe la estimulación parasimpática.

La estimulación vagal intensa puede disminuir la fuerza de contracción del músculo cardiaco en un 20% a un 30% [3].

1.1.8 LA CIRCULACIÓN

La función de la circulación consiste en atender las necesidades del organismo, es decir el transporte de nutrientes y recolección de desechos a cada uno de los tejidos del organismo para lograr la supervivencia y la óptima funcionalidad de las células que lo conforman.

El flujo sanguíneo[5] está determinado por la necesidad de nutrientes del organismo y estás depende de la presión arterial y el gasto cardiaco[6].

1.1.9 CARACTERÍSTICAS FÍSICAS DE LA CIRCULACIÓN

La circulación en el cuerpo está dividida en circulación sistémica, mayor o periférica y circulación menor o pulmonar. La circulación menor es la encargada de oxigenar la sangre, mientras que la circulación mayor es la encargada de llevar oxígeno y nutrientes a las células que conforman el cuerpo[3].

Los componentes que conforman la circulación son:

- Las arterias que transporta sangre con una elevada presión y velocidad.

---

[5] **Flujo sanguíneo:** Volumen de sangre por unidad de tiempo que pasa por vaso.
[6] **Gasto cardiaco:** Volumen de sangre expulsado por un ventrículo por unidad de tiempo.

- Las arteriolas son las ramas más pequeñas del sistema arterial y controlan los conductos a través de los cuales se libera la sangre a los capilares.
- Los capilares encargados del suministro de líquidos y nutrientes a los tejidos.
- Las vénulas que recoge la sangre de los capilares.
- Las venas que transportan la sangre de regresar corazón.

1.1.10 TRANSMISIÓN DE LOS PULSOS DEL CORAZÓN HACIA LAS ARTERIAS PERIFÉRICAS

Durante la sístole el corazón expulsa la sangre a gran velocidad e inicia expandiendo la porción proximal de la aorta, la inercia de la sangre impide que esta se mueva bruscamente hacia la periferia. A medida que la presión en la aorta aumenta esta inercia es superada y la distensión se propaga como se muestra en la **Figura 1.5**.

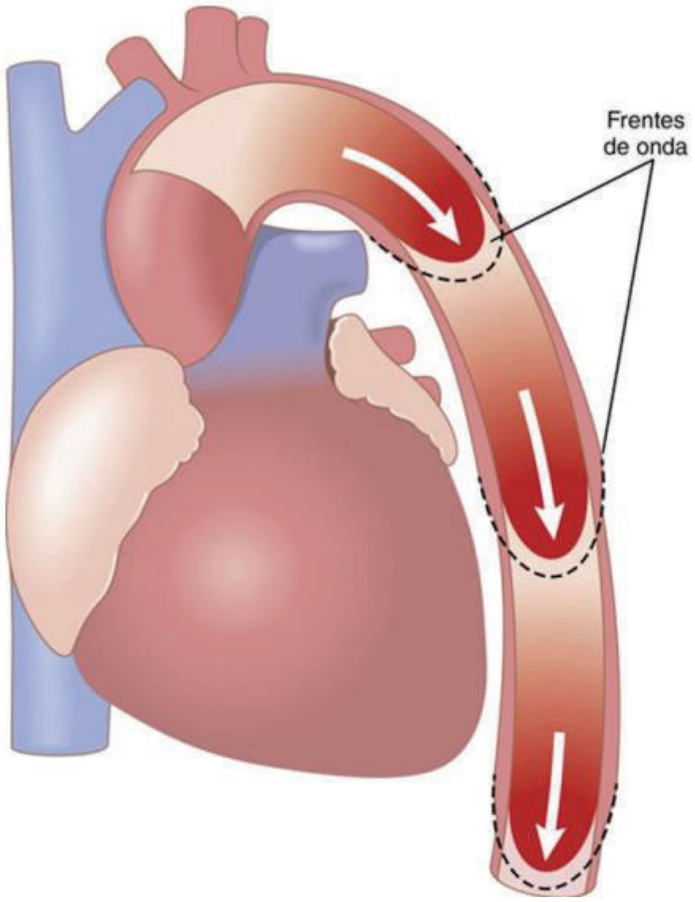


Figura 1.5 Transmisión del impulso de presión[1]

Las pulsaciones a medida que se alejan del corazón experimentan una disminución progresiva en su presión máxima como se puede observar en la **Figura 1.6**.

En los capilares las pulsaciones tan sólo pueden ser observadas cuando la presión a nivel de la aorta ha sido elevada, el motivo de esta disminución es la resistencia que ofrecen las arterias al movimiento de la sangre y de la distensibilidad[7] de los vasos que amortiguar las pulsaciones.

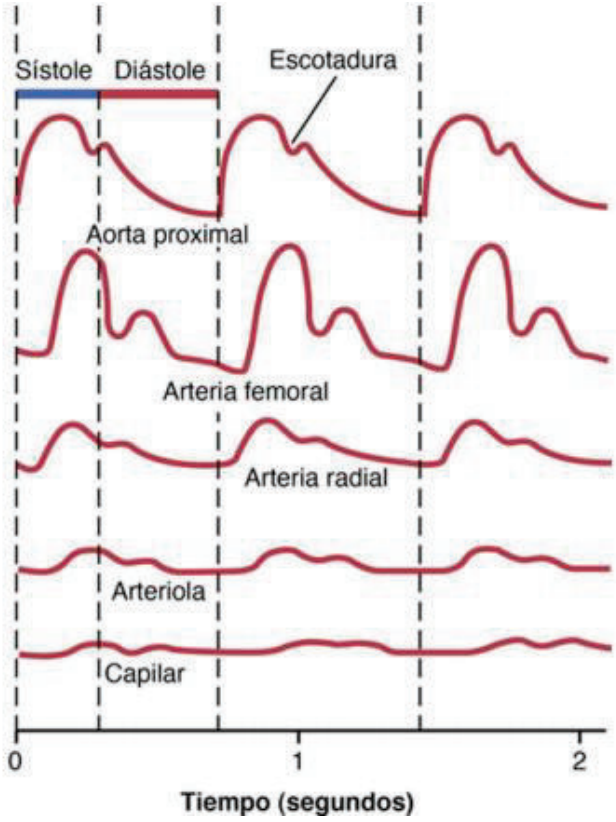


Figura 1.6 Cambios del perfil del impulso de presión a medida que la onda de pulso viaja hacia vasos más pequeños[1].

Los grandes vasos como la aorta poseen poca distensibilidad, pero los vasos más pequeños a medida que se alejan del corazón poseen una distensibilidad mayor, los cambios en el calibre de los vasos pequeños son más sutiles, pero mantienen la periodicidad de contracción del corazón.

[7] **Distensibilidad:** Capacidad de los vasos sanguíneos de expandirse o contraerse ante el paso de sangre.

1.1.11 ALTERACIÓNES FISIOLÓGICAS

Las alteraciones en la musculatura cardíaca, así como anormalidades y bloqueos en la conducción de los impulsos que transitan por el sistema cardionector del corazón dan paso a un ritmo cardiaco anormal.

1.1.11.1 Ritmicidad anormal del marcapasos

Aquí mencionamos a la taquicardia que se define como una frecuencia de pulso superior a 100 latidos por minuto y la bradicardia que se define como una frecuencia de pulso menor a 60 latidos por minutos. Se debe diferenciar la bradicardia patológica de la que se da en los atletas, a causa de su entrenamiento pueden mantener normalmente una frecuencia cardíaca menor a 60 latidos por minuto en reposo.

1.1.11.2 Ritmos anormales en el corazón

La señales que estimulación el ciclo cardiaco normalmente se generan a nivel de las aurículas para pasar a los ventrículos, inician en el nódulo sinusal, pasan al nódulo aurículo-ventricular, seguido al has de hiz para terminar finalmente en sus ramas derecha e izquierda como se observa en la **Figura 1.2**.

Cuando por algún motivo los nódulos resultan dañados o se producen bloqueos en su conducción se producirán ritmos anormales. Los bloqueos interfieren con la comunicación entre las aurículas y los ventrículos clasificándose de la siguiente manera:

- Primer grado: Bloqueo leve de la conducción, existen retrasos en la comunicación entre las aurículas y ventrículos.
- Segundo grado: Bloqueo moderado de la conducción, algunos pulsos generados en las aurículas no llegan a los ventrículos.
- Tercer grado: Bloqueo total de la comunicación entre las aurículas y los ventrículos.

Cuando se producen estos bloqueos el corazón tiene un sistema de escape, es decir, cuando los ventrículos no reciben una señal para iniciar la contracción desde las

aurículas comienzan a latir por sí mismos, en un esfuerzo por impedir que el corazón se detenga. A diferencia de los pulsos que se generan en las aurículas, el pulso generado en los ventrículos puede alcanzar una frecuencia máxima de 60 latidos por minuto y no es periódico. En un bloqueo de segundo grado como el mostrado en la **Figura 1.7** la señal de las aurículas se transmite a los ventrículos, pero en momentos es interrumpida, por su origen en las aurículas las señales que no han sido interrumpidas mantienen un tiempo de separación constante entre ellas.

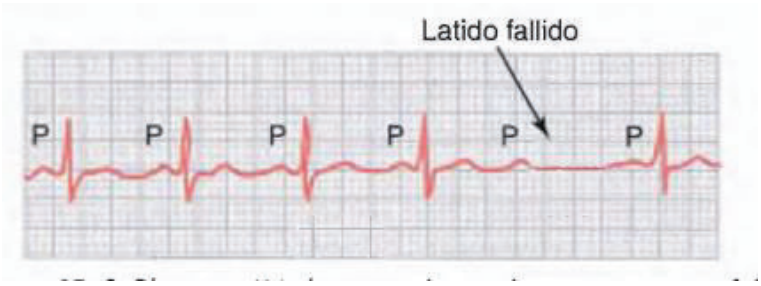


Figura 1.7 Bloqueo de segundo grado en el que se muestra un latido fallido[1].

1.1.11.3 Pulsos anormales del corazón

Otra de las anormalidades que se pueden generar en el corazón es la presencia de vías que aceleran la trasmisión del pulso desde las aurículas a los ventrículos, o la presencia de nódulos que generan un pulso de manera espontánea. Estas alteraciones generan un pulso irregular como el mostrado en la **Figura 1.8**, aquí se aprecia una extrasístole como resultado de un nódulo parásito.

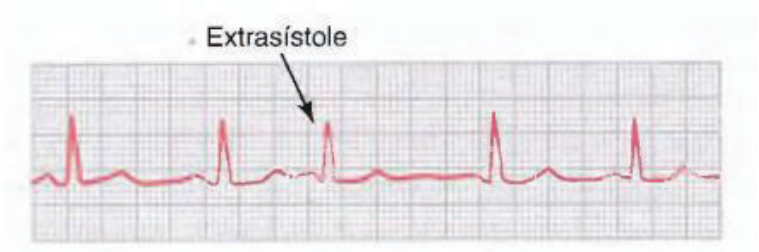


Figura 1.8 Extrasístole [1]

Para la determinación de la frecuencia cardíaca en estas condiciones en la práctica se establece un tiempo en el que se contará el número de pulsos generados, este

tiempo por lo general oscila entre 6 a 20 segundos, para mitigar errores es aconsejable la utilización de tiempos largos.

1.2 COMUNICACIONES INALÁMBRICAS

Una comunicación inalámbrica se compone de dos dispositivos, un emisor y un receptor que comparten información que es propagada por un medio no guiado.

Este tipo de comunicaciones es realizado a través de ondas electromagnéticas, y son de gran utilidad cuando un dispositivo no se encuentra en una posición fija, los costos por la implementación de cableado estructurado son elevados o cuando se desea establecer comunicación en zonas de difícil acceso.

1.2.1 REDES INALÁMBRICAS

Las redes inalámbricas cada día son más aceptadas como se observa en la proyección del número de dispositivos móviles en la **Figura 1.9**, ganando mayor preferencia por parte de los usuarios.

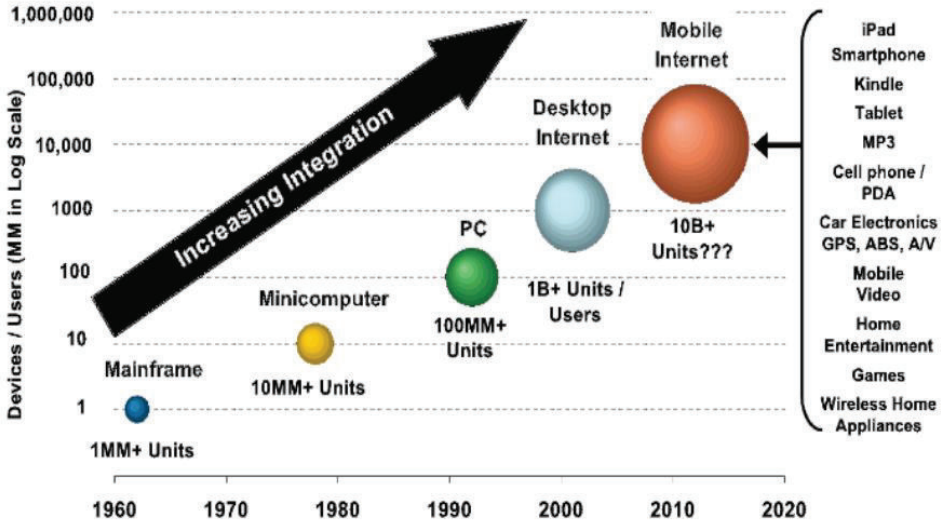


Figura 1.9 Proyección del número de dispositivos móviles [4]

La clasificación de las redes inalámbricas se la realiza en base a la cobertura que ofrecen como se muestra en la **Figura 1.10**.

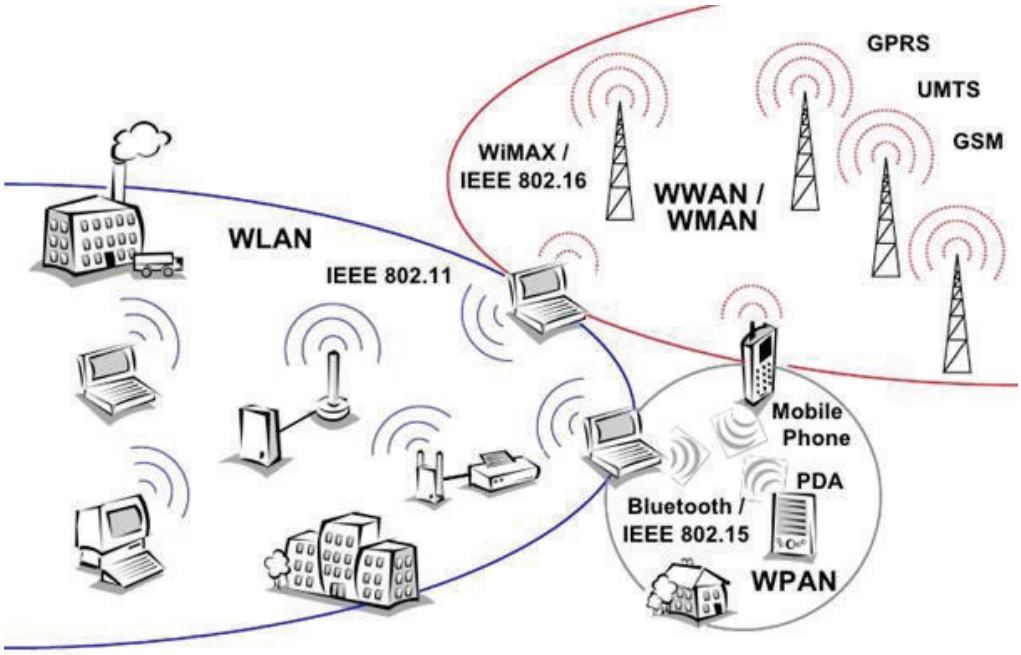


Figura 1.10 Clasificación de las redes inalámbricas por la cobertura [5].

1.2.2 REDES INALÁMBRICAS DE ÁREA PERSONAL: WPAN

El concepto de WPAN es introducido por el comité IEEE 802.15, este comité es responsable de la elaboración de estándares para comunicar dispositivos que se encuentran a distancias cortas.

Las redes WPAN comunican dispositivos que se encuentran separados por distancias no superiores a los 10 metros. Los dispositivos que comúnmente se comunican utilizando WPAN son computadoras, impresoras, televisores, teclados inalámbricos.

Los dispositivos empleados en el campo de las WPAN poseen distintos grados de complejidad abarcando desde pequeños sensores económicos hasta dispositivos con capacidades computacionales y de comunicación con la red avanzadas.

Entre las WPAN más utilizadas se mencionan:

1.2.2.1 Bluetooth

El protocolo Bluetooth posee un alcance de aproximadamente 10 metros y establece conexiones utilizando el mínimo de energía posible.

Se puede obtener alcances mayores pero con la desventaja de un mayor consumo de energía. Este protocolo utiliza la banda ISM[8] de 2,4GHz. Existen varias clases de dispositivos que utilizan este protocolo como se las indica a continuación:

- **Clase 1:** Potencia de 100 mW o 20 dBm y un alcance aproximado de 100 metros.
- **Clase 2:** Potencia de 2,5 mW o 4 dBm y un alcance aproximado de 20 metros.
- **Clase 3:** Son las de mínimo consumo, con una potencia de 1 mW y un alcance aproximado de 1 metro.[6]

1.2.2.2 Zigbee

Es el nombre del estándar que abarca un grupo de protocolos de alto nivel y bajo consumo energético, es desarrollado por la Alianza ZigBee.

El estándar ZigBee [7] define un *stack* para las capas de red, aplicación y seguridad como se muestra en **Figura 1.11**.

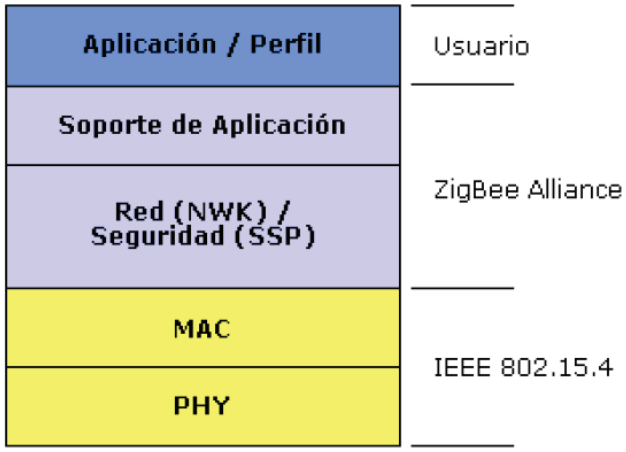


Figura 1.11 Pila de protocolos ZigBee[7].

1.2.3 REDES INALÁMBRICAS DE ÁREA LOCAL: WLAN

Este tipo de redes poseen las mismas ventajas que las redes LAN, pero sin la necesidad de utilizar cables, facilitando la movilidad del usuario. Trabajan en banda

[8] **ISM - *Industrial, Scientific and Medical*:** Industrial, Científica y Médica, son bandas no comerciales de radiofrecuencia utilizadas para aplicaciones industriales, científicas y médicas.

libre lo que reduce gastos y ha permitido su apertura en el mercado, en este tipo de redes se mencionan:

1.2.3.1 IEEE 802.11

Fue desarrollado por el grupo de trabajo IEEE 802.11 para comunicaciones entre redes inalámbricas. Desde su desarrollo en 1990 se han elaborado nuevos estándares como los siguientes:

- 802.11a: 54 Mbps a 5 GHz, ratificado en 1999.
- 802.11b: 11 Mbps 2.4 GHz, ratificado en 1999.
- 802.11d: *World mode*, ratificado en 2001.
- 802.11e: Calidad de servicio, ratificado in 2005.
- 802.11g: 54 Mbps at 2.4Ghz, mayor velocidad de datos que 802.11b, ratificado en 2003.
- 802.11h: Selección de frecuencia dinámica y mecanismo de control de potencia de transmisión, ratificado en 2003.
- 802.11i: Autenticación y seguridad, ratificado en 2005.
- 802.11j: Frecuencias japonesas adicionales, ratificado en 2005.
- 802.11k: Gestión de recursos de radio, ratificado en 2007.
- 802.11n: Alto rendimiento, ratificado en 2007.

En un principio fueron diseñadas para ambientes laborales, pero actualmente son muy comunes y abarcan desde pequeños hogares hasta grandes redes corporativas. Existieron problemas de interoperabilidad que fueron solucionados por *Wireless Ethernet Compatibility Alliance* o WECA por sus siglas en inglés que posteriormente pasó a llamarse *Wireless Fidelity* o Wi-Fi y está encargada de certificar la interoperabilidad entre dispositivos.

1.2.3.2 Wireless fidelity

Wireless Fidelity o Wi-Fi es la tecnología de mayor preferencia para conexiones inalámbricas de computadoras y dispositivos móviles por su gran alcance. Las tecnologías inalámbricas no han desplazado a las tecnologías cableadas por

completo pero los ambientes en los cuales los usuarios tienden a cambiar de sitio se han convertido en una excelente opción.

1.2.4 REDES INALÁMBRICAS DE ÁREA METROPOLITANA: WMAN

Las WMAN son redes con una cobertura geográfica extensa de varios kilómetros, un representante notable de este tipo de redes fue WiMAX, que ofrecía comunicación vía microondas utilizando frecuencias entre 2,5 GHz y 5,8 GHz y una cobertura aproximada de 50Km. Esta tecnología ha dejado de ser utilizada y se ha reemplazado por tecnologías como por ejemplo LTE.

1.2.4.1 Long term evolution

LTE o *Long Term Evolution* es un estándar para la transmisión de datos inalámbricos de alta velocidad en dispositivos móviles, es un estándar desarrollado por 3GPP y basado en GSM[9]/EDGE[10] y UMTS[11]/HSPA[12]. 3GPP es una colaboración de grupos de asociaciones de telecomunicaciones cuyo objetivo principal fue asentar las especificaciones del sistema global de comunicaciones de tercera generación con base en las especificaciones de *Global System for Mobile Comunications* o GSM.

A diferencia de las tecnologías 2G y 3G que se basan en conmutación de circuitos, LTE propone una conmutación de paquetes[8], destaca por su interfaz radioeléctrica basada en OFDMA[13], para el enlace descendente (DL) y SC-FDMA[14] para el enlace ascendente (UL) y facilidad para implementar antenas con tecnología MIMO[15].

---

[9] **GSM –** ***Global System for Mobile Comunications:*** Estándar que describe los protocolos de segunda generación de redes celulares digitales.

[10] **EDGE -** ***Enhanced Data rates for GSM Evolution*****:** Tecnología de telefonía móvil digital que permite mejorar las tasas de transmisión de datos compatible con GSM.

[11] **UMTS -** ***Universal Mobile Telecommunications System*****:** Es un sistema celular móvil de tercera generación para redes basadas en el estándar GSM.

[12] **HSPA -** ***High Speed Packet Access*****:** Amalgama de los protocolos de enlace descendente y ascendente que permite mejorar el rendimiento de redes 3G.

[13] **OFDMA – Multiplexación por división de frecuencias ortogonales:** Es un método de acceso que permite a múltiples usuarios compartan el canal con aplicaciones que utilicen velocidades bajas.

[14] **SC-OFDMA:** Se puede considerar como la versión pre-codificada de OFDMA mediante la transformada discreta de Furier.

[15] **MIMO –** ***Multiple Input/Multiple Output***: Método que utiliza múltiples emisores, receptores o ambos para minimizar la tasa de errores e incrementar la tasa de transmisión.

1.3 IEEE 802.15.4

En los grupos de trabajos de IEEE 802 encontramos a IEEE 802.15 especializado en WPAN. Este grupo de trabajo se divide a su vez en 5 subgrupos como se muestra en la **Figura 1.12** .

En el presente trabajo de titulación se trabajarán las definiciones del grupo de trabajo 4, IEEE 802.15.4, WPAN con velocidades cercanas a los 250 Kbps y un disminuido consumo de energía, muy útil para aplicaciones con sensores que necesiten bajas tasas de transmisión de datos como en el presente trabajo de titulación.

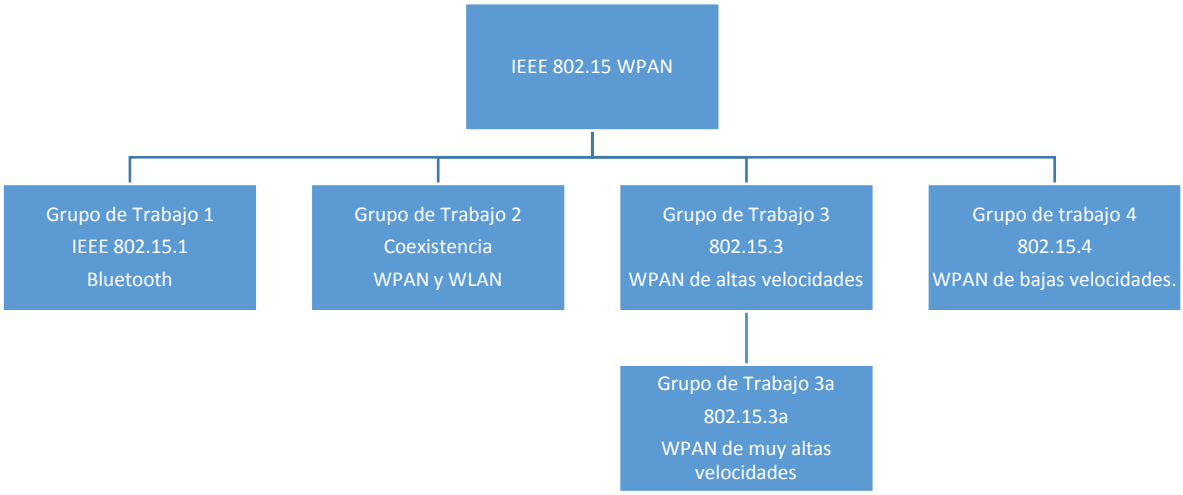


Figura 1.12 Grupos de trabajo IEEE 802.15 [9]

El estándar IEEE 802.15.4 permite la creación de dispositivos que a más de poseer un bajo consumo de energía y una baja tasa de transmisión de datos posean una baja complejidad, esto repercute positivamente en el desarrollo de dispositivos con un costo disminuido.

IEEE 802.15.4 es considerado el estándar por defecto para el desarrollo de las WSN[16][10] y trabaja en las capas PHY (Física) y MAC (*Medium Access Control*) del

[16] **WSN - *Wireless Sensor Network* o Red de sensores inalámbrica:** Pequeños dispositivos conocidos como nodos que poseen sensores y que colaboran entre sí mediante comunicaciones inalámbricas para cumplir un objetivo en común.

modelo de referencia ISO/OSI[17]. Para las capas superiores encontramos la pila de protocolos correspondiente a ZigBee y 6LowPAN[18][11].

IEEE 802.15.4 permite una comunicación confiable mediante el envío de acuse de recibo, detección de errores y retransmisión. Ante la simpleza en el envío de datos, requiere de poco procesamiento para realizar la comunicación, permitiendo un ahorro de energía y prolongando los momentos de inactividad del dispositivo [12].

1.3.1 CAPA FÍSICA

La capa física es la capa inicial del modelo ISO/OSI, IEEE 802.15.4 en la capa física utiliza DSSS[19] permitiendo coexistencia con tecnologías como 802.11 b y g que utilizan modulación similar, en las bandas no licenciadas de 868MHz, 914Mhz y 2.4Ghz.Esta capa es la encargada de las siguientes funciones:

- Encendido y apagado del transceptor[20]
- Acceso al medio por CSMA/CA[21]
- Detectar la energía
- Evaluación del canal
- Selección del canal
- Recepción y transmisión de datos.

En esta capa se crea el enlace entre los dispositivos, se produce la modulación y demodulación y el sincronismo entre emisor y receptor [10].

La banda más utilizada es la de 2.4Ghz y se especifican 16 canales numerados del 11 al 26 separados 5Mhz, abarcan desde 2405Mhz hasta 2480 Mhz. Como se puede

---

[17] **ISO/OSI -** ***International Organization for Standarization/Open System Interconnection*****:** Modelo de referencia de capas para los protocolos de red.
[18] **6LowPAN - IPv6** ***over Low-Power*** **PAN:** Protocolo que permite la implementación de redes IPv6 sobre redes personales de baja potencia.
[19] **DSSS -** ***Direct Sequence Spread Spectrum*****:** Método de codificación del canal en espectro ensanchado.
[20] **Transceptor:** Dispositivo conformado por un emisor y receptor que comparten una misma circuitería.
[21] **CSMA/CA -** ***Carrier Sense Multiple Access with Collision Avoidance*****:** Método de acceso múltiple mediante detección de portadora con el fin de prevenir colisiones.

observar en la **Figura 1.13** los canales 15, 20, 25 y 26 de IEEE 802.15.4 no se superponen con los canales 1, 6 y 11 de 802.11 b/g lo que facilita la coexistencia entre ambas en la configuración norteamericana [12] .

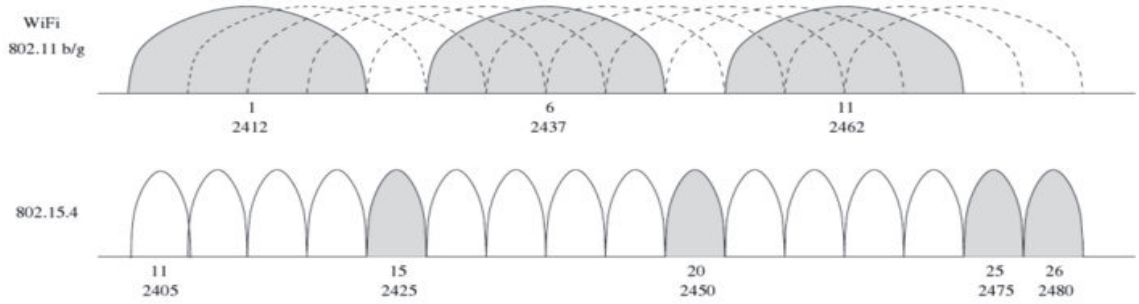


Figura 1.13 Canales 802.15.4 y 802.11 b/g en configuración norteamericana [12]

En la alternativa europea los canales de IEEE 802.15.4 15, 16, 21 y 22 se encuentran entre los canales 1, 7 y 13 de IEEE 802.11 b/g como se muestra en la **Figura 1.14** [12].

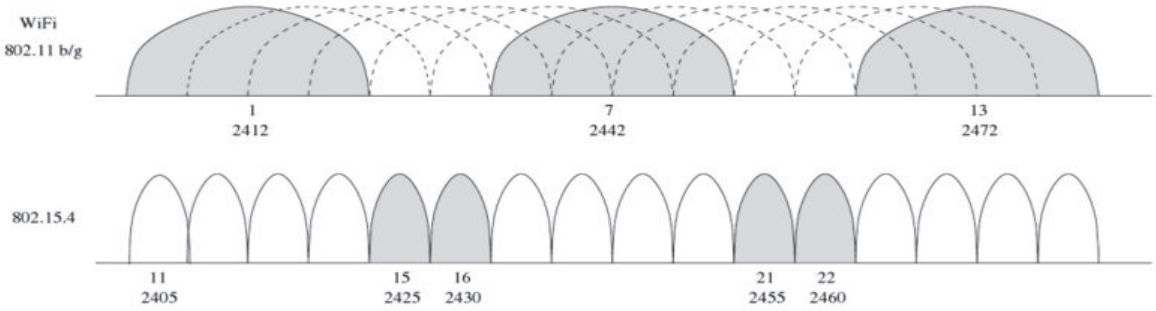


Figura 1.14 Canales 802.15.4 y 802.11 b/g en configuración europea[12].

La velocidad de modulación obtenida es de 62500 símbolos por segundo, cada símbolo posee 4 bits, obteniendo 250Kbps. El ancho de banda efectivo es de 2Mhz en el canal [12].

1.3.1.1 Evaluación del canal

Cuando se va a establecer una comunicación es necesario seleccionar un canal, para evaluar el canal más adecuado y teniendo en cuenta que debe coexistir con otras tecnología como IEEE 802.11 e IEEE 802.15.4, se procede a analizar los

niveles de señal utilizando procedimientos denominados *Energy Detection* (ED) y *Clear Channel Detection* (CCA).

1.3.1.2 Detección de energía

Para obtener una valoración del nivel con que es recibida una señal o RSSI[22] se mide la energía que se encuentra en el canal y se la evalúa como una señal compatible. Esto permite conocer si el canal se encuentra libre u ocupado.

1.3.1.3 Evaluación de canal libre

Es la manera en la que se busca un canal libre, un canal se encontrará ocupado si presenta las siguientes características:

- Un nivel de energía suficiente que ha superado un umbral determinado preestablecido.
- La presencia de una portadora cuando se encuentran señales compatibles con IEEE 802.15.4 en el canal.
- Cuando se presentan señales compatibles con IEEE 802.15.4 y un nivel de energía suficiente.

1.3.1.4 Sincronización

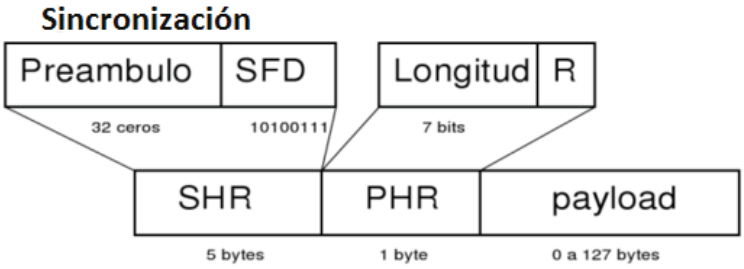
Para que un receptor conozca el inicio de una transmisión se establece un preámbulo que consiste en 32 ceros seguidos de la secuencia 10100111 enviados al inicio de la transmisión.

A continuación se envía un byte que contiene la longitud de la trama tal como se muestra en la **Figura 1.15** [12].

Al enviar la trama ocurre un proceso conocido como *turnaround*, en este proceso cuando el transmisor que envió la trama se apaga para proceder a encender el receptor[12], en el otro extremo de la comunicación ocurre exactamente lo contrario, es decir, una vez que recibió la trama, el receptor se apaga y el transmisor puede encenderse.

---

[22] **RSSI - *Recieve Signal Strength Indication*:** Es una medida de la energía presente en una señal de radio.



SHR: *Sinchronization Header* – Cabecera de sincronización
SFD: *Start of Frame Delimiter* - Delimitador de comienzo de trama
PHR: *PHY Header* – Encabezado de la capa física.

Figura 1.15 Sincronización de trama IEEE 802.15.4[12]

1.3.1 CONTROL DE ACCESO AL MEDIO: MAC

Es especificado por IEEE 802.15.4 para garantizar la correcta transmisión extremo-extremo, en este nivel no está especificado el enrutamiento.

MAC provee retransmisión y detección de errores FCS[23] por CRC[24].

Para cada una de las tramas tenemos un encabezado o *header* en donde se encuentra la identificación de la trama y la dirección a la que se dirige.

En cada trama también encontramos una cola o *footer* que contiene el FCS para el chequeo de errores como se observa en la **Figura 1.16** [12].

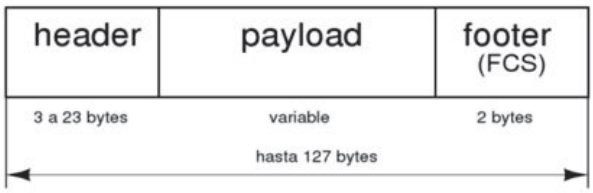


Figura 1.16 Estructura de la trama IEEE 802.15.4 [12]

[23] **FCS:** ***Frame Check Sequence*****:** Conjunto de bits por lo general al final de la trama utilizados para verificar la integridad de la misma.
[24] **CRC:** ***Cyclic Redundancy Check*****:** Código utilizado para la detección de errores durante la trasmisión de datos.

1.3.1.1 Acceso al medio

Para que sea posible la comunicación por un mismo canal, reduciendo el número de colisiones entre los dispositivos IEEE 802.15.4 utiliza el método de CSMA/CA que permite el múltiple acceso con detección de portadora evitando colisiones.

CSMA/CA establece que cuando un dispositivo desea utilizar el medio para transmitir, debe esperar un tiempo aleatorio antes de iniciar la transmisión.

Cada dispositivo que desee transmitir calcula el número de periodos que debe esperar, este valor se selecciona aleatoriamente entre 0 y 2^{BE}-1 periodos, siendo BE el exponente de *backoff*.

Si al terminar el tiempo calculado el dispositivo no puede transmitir se incrementa el valor de BE en uno y se calcula un nuevo número de periodos que debe esperar el dispositivo para intentar trasmitir[12]. El incremento exponencial del intervalo de tiempo de espera para iniciar la transmisión reduce el número de colisiones.

1.3.1.2 Direccionamiento

En IEEE 802.15.4 el direccionamiento no tiene un tamaño fijo y puede ser de un tamaño reducido para minimizar la sobrecarga e incluso puede ser omitido si la dirección se encuentra implícita en el mensaje.

Dentro del encabezado como se aprecia en la **Figura 1.17** existe un campo que permite indicar si se utiliza direccionamiento corto, extendido o si se omitió el direccionamiento tanto en el origen como en el destino. Además aquí encontramos el número de secuencia de trama que se utiliza para relacionarlo con su acuse de recibo o ACK[25].

El direccionamiento extendido posee una dirección conocida como IEEE de 64 bits única para diferenciar a cada dispositivo de los demás. El direccionamiento corto posee una dirección de 16 bits, única para cada dispositivo dentro de la red a la cual pertenece.

---

[25] **ACK: *Acknowledgement* – Acuse de recibo:** Mensaje enviado por el receptor al emisor de confirmación de la recepción de un mensaje.

Encontramos otros campos como el PAN ID que nos permiten identificar las redes origen y destino.

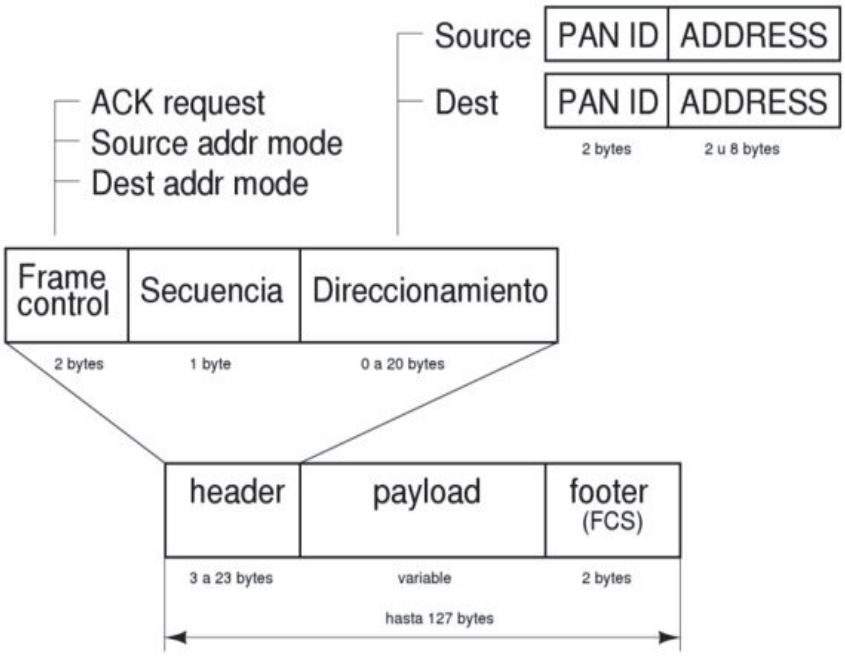


Figura 1.17 Cabecera de IEEE 802.15.4[12]

En la Tabla 1.1 se observan algunas de las características presentes en 802.15.4.

Tabla 1.1 Características de IEEE 802.15.4 [12]

PROPIEDAD	RANGO
Rango de transmisión de datos	868 MHz:20kb/s; 915 MHz:40kb/s; 2.4GHz:250kb/s.
Alcance	10 – 20 m.
Latencia	Menor a 15 mseg.
Canales	868/915 MHz: 11 canales. 2.4 GHz: 16 canales.
Bandas de frecuencia	868/915 MHz y 2.4 GHz.
Direccionamiento	Corto: 16 bits o extendido: 64 bits IEEE.
Canal de acceso	CSMA/CA y CSMA/CA ranurado.
Temperatura	Rango de temperatura industrial: -40° a +85° C.

1.4 ZIGBEE

Desarrollado por la Alianza ZigBee, ZigBee es el nombre que se da a un estándar que abarca un grupo de protocolos de alto nivel y bajo consumo energético, todo esto basado en el estándar 802.15.4.

El estándar ZigBee define protocolos concernientes a las capas de red, aplicación y seguridad, los desarrolladores crean sus aplicaciones o las integran a aplicaciones desarrolladas por Alianza ZigBee[7] .

ZigBee es utilizado en aplicaciones de domótica por tres de sus características:

- Bajo consumo
- Topología de red de malla
- Fácil integración

Estas características permiten crear dispositivos con una electrónica muy simple.

1.5 REDES DE SENSORES INALÁMBRICOS: WSN

La miniaturización, el bajo consumo, uso de bandas no licenciadas han permitido un gran desarrollo de las redes de sensores inalámbricos en los últimos años, ofreciendo un abanico de soluciones en el ámbito industrial y científico.

Las WSN tomarán un rol fundamental en los futuros sistemas de comunicaciones, como en el caso de IoT[26] para la comunicación machine-to-machine.

IEEE 802.15.4 es considerado el estándar de facto para WSN y a pesar de tener bibliografía detallada sobre el desarrollo de redes de sensores usando IEEE 802.15.4 su utilización a gran escala aún es muy baja a pesar de su gran potencialidad.

1.5.1 FUNDAMENTOS DE WSN

Las WSN utilizan dispositivos llamados nodos o motes que permiten medir y comunicar la información obtenida por medio de enlaces inalámbricos.

---

[26] **IoT – Internet de las cosas:** Concepto que hace referencia a la interconexión de objetos cotidianos a internet.

Las WSN comenzaron a estudiarse preliminarmente cuando se trabajaba con redes ah hoc[27] pero su estudio formal comenzó después del año 2000. Los nodos de las WSN son conectados utilizando topologías ad hoc teniendo en cuenta las necesidades de los usuarios [13].

Las redes ad hoc poseen características diferentes a las WSN, las WSN se caracterizan por tener nodos con capacidad reducida, pequeño tamaño, baja complejidad y costo, trabajar con cantidades pequeñas de información, pueden enviar y recibir datos pero no al mismo tiempo y la principal característica, el ahorro de energía. Las ah hoc por su parte trabajan con cantidades grandes de información, pueden enviar y recibir datos al mismo tiempo y son diseñadas para dispositivos inteligentes como computadoras, el ahorro de energía no es una prioridad.

Entre las principales características de las WSN tenemos:

- Gran escalabilidad
- Auto organización
- Auto corrección de errores.
- Eficiencia energética
- Complejidad reducida
- Bajo costo
- Tamaño reducido de nodos.

Cabe resaltar que para la creación de una WSN es importante la arquitectura de protocolos y el desarrollo técnico para cada solución.

Una guía capaz de mostrar los protocolos necesarios para cada solución no existe desafortunadamente por lo que se requiere investigación y desarrollo.

En la **Tabla 1.2** se observa una comparación entre las características de las redes ad-hoc con las WSN.

[27] **Ad-hoc:** Red inalámbrica descentralizada, se trata de una red que no depende para su funcionamiento de una infraestructura establecida previamente.

Tabla 1.2 Comparación de WSN vs Ad Hoc [9]

	AD-HOC	WSN
Nodos	Muchos	Pocos
Vulnerabilidad de los dispositivos	Baja	Alta
Cambios en la topología	Pocos	Muchos
Técnicas de comunicación	Punto a punto	*Broadcast*
Capacidad de energía, memoria y procesamiento	Alta	Baja
Identificación global	Si	No
Finalidad	Tareas complejas	Detección o estimación de eventos.

En las WSN la información obtenida por un nodo llega realizando uno o múltiples saltos a un controlador o servidor que analiza la información recibida y permite la conexión con redes externas. Ante el panorama descrito anteriormente podemos tener dos escenarios, el primero en el que múltiples nodos (círculos verdes) se comunican con un solo controlador (cuadrado azul) y el segundo en el que múltiples nodos se comunican con varios controladores como se observa en la **Figura 1.18**.

En el primer caso al tener un solo controlador se experimenta un problema de escalabilidad, al crecer el número de nodos también aumentará la cantidad de información recibida en el controlador hasta saturarlo. En el segundo caso al poseer varios controladores la probabilidad de poseer nodos que no puedan enviar su información desciende y la escalabilidad ya no es un problema. A pesar de tener varios controladores cada nodo seleccionaran uno de ellos para enviar la información al usuario final.

Los criterios para la selección del controlador pueden darse en base a:

- Número de saltos
- Retardos mínimos

- Máximo *throughput*, etc.

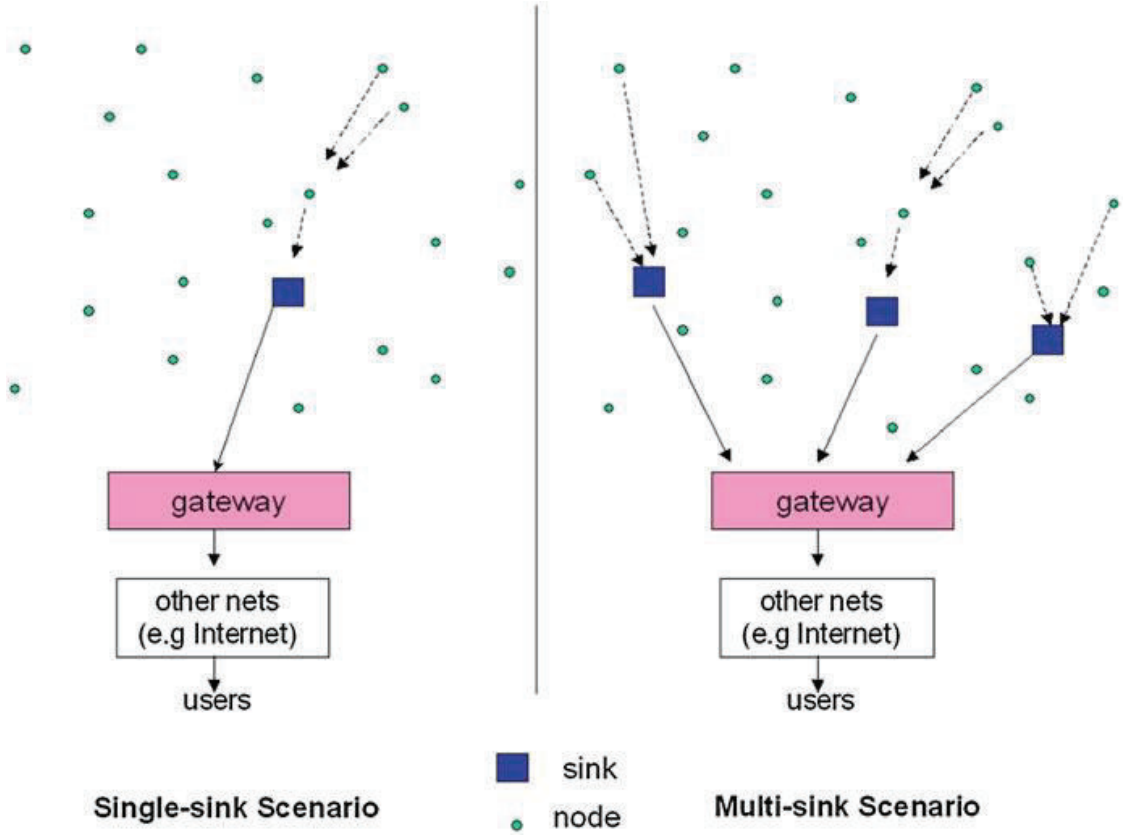


Figura 1.18 WSN con un solo controlador y con varios controladores [10].

Aunque parece ser una ventaja poseer varios controladores uno de los principales problemas al tener en consideración es la mayor complejidad que presentarán los protocolos de comunicación.

1.5.2 GESTIÓN DE ENERGÍA

Las WSN generalmente dependen de baterías, por lo que el tiempo de vida útil de la red estará en función del tiempo que estas sean capaces de proveer energía suficiente al dispositivo para recibir, procesar y transmitir la información.

Un nodo posee generalmente los siguientes elementos:

- Batería
- Memoria

- Procesador
- Sensores
- Transceptor o *transceiver*

Cuando un nodo se encuentra en estado activo, el transceptor demanda mayor corriente que el microprocesador, considerando que la principal ventaja de las redes WSN es su duración, un consumo energético elevado opacaría esta ventaja. El *transceiver* estará activo solo cuando un nodo desee transmitir información, mientras menos veces el nodo necesite realizar esta tarea mayor será la duración de la batería y por ende mayor el tiempo de vida útil del nodo, reducir en lo posible el uso de protocolos complejos que usen *handshakes*[28] y limitar el uso de ACK son importantes para maximizar la duración de las baterías.

1.5.2.1 Aplicaciones

El abanico de aplicaciones en las que pueden intervenir las WSN es sumamente extenso, algunos ejemplos son los siguientes:

- Monitoreo ambiental
- Posicionamiento y rastreo
- Logística
- Cuidados de la salud como en el presente trabajo de titulación.

1.6 NODOS

Los nodos sensores, también conocidos como motas o motes pueden poseer sensores incorporados o permiten agregar sensores acorde a las necesidades del desarrollador. Respecto a la energía que es utilizada por los nodos estos pueden estar en alguno de los siguientes estados:

- ***Sleep*:** en el cual el nodo se encuentra en un estado de hibernación, mientras mayor sea el tiempo que el nodo pasa en este estado menor será el consumo de energía.

[28] ***HandShake*:** Proceso automatizado de negociación dinámico que establece los parámetros del canal antes de iniciar la comunicación.

- ***Wakeup*:** En este estado el nodo pasa del estado *sleep* al estado activo, esto puede ser por un estímulo o interrupción programada.
- ***Active*:** Aquí se produce el transmisión y recepción de información, es el estado en el que mayor batería se consume por ende es aconsejable que el nodo se encuentre en este estado el menor tiempo posible.

En el mercado podemos encontrar una gran variedad de nodos pero el utilizado en el presente trabajo de titulación es el IRIS X2110 del fabricante MEMSIC.

1.6.1 IRIS X2110

El nodo como fue establecido en el plan del presente trabajo de titulación es el IRIS X2110 y se observa en la **Figura 1.19**.

IRIS es la última generación de motes MEMSIC, X2110 trabaja en las frecuencias de 2400Mhz a 2483.5Mhz [14].

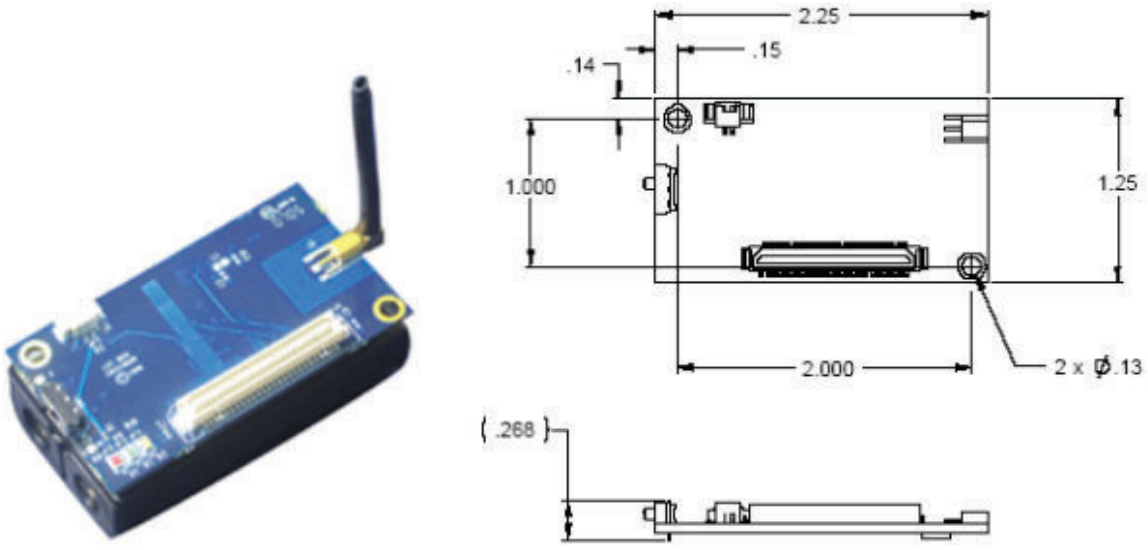


Figura 1.19 X2110 con una antena estándar[14]

En la **Figura 1.20** se observa el diagrama de bloques de la mota IRIS, se utiliza el procesador Atmel-RF230 que es compatible con IEEE 802.15.4 y un transceptor de radio listo para trabajar con ZigBee con un controlador Atmega128. Todas las aplicaciones y software de la familia MICAz son compatibles con X2110 [15].

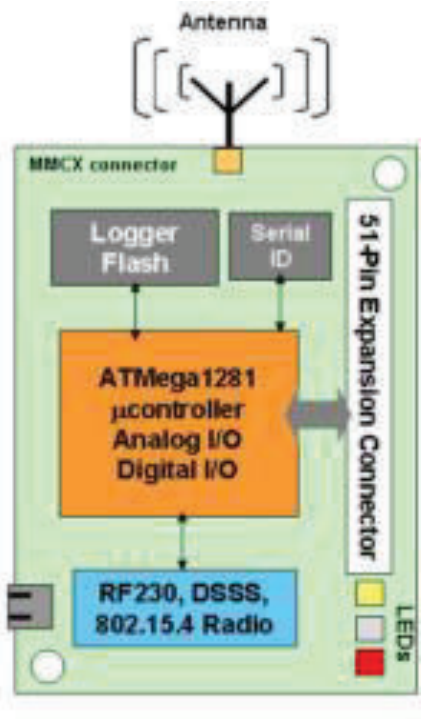


Figura 1.20 Diagrama de bloques X2110[15]

1.6.1.1 Energía

Los mote IRIS han sido diseñados para trabajar utilizando baterías AA, pero prácticamente cualquier combinación capaz de suministrar un voltaje entre 2,7 y 2,6 voltios es suficiente.

Tabla 1.3 Corriente de operación en IRIS [15]

Corriente de operación [mA]	IRIS
Procesador, activo	8 (7,37 Mhz)
Procesador, *Sleep*	0.008
Radio, recibiendo	16
Radio, transmitiendo	17
Radio, *sleep*	0.001
Memoria flash, escritura	15
Memoria flash, lectura	4
Memoria flash, *sleep*	0.002

En la **Tabla 1.3** se muestran la corriente utilizada por el mote en diferentes escenarios. Se puede resaltar que la transmisión de señales de radio es la función que mayor corriente requiere, aquí se observa la importancia de realizar el mínimo de transmisiones posibles cuando se desea prolongar la duración de las baterías.

1.6.1.2 Radio

AT86RF230 utiliza modulación O-QPSK (*offset quadrature phase shift keying*) con una configuración de pulso semi-senoidal, La comunicación de radio IEEE 802.15.4 incluye DSSS (*Digital direct sequence Spread Spectrum*) con una ganancia de propagación de 9 dB y una velocidad efectiva de 250 kbps.

Tabla 1.4 Configuración de potencia de radiofrecuencia[15]

Potencia de RF[29] (dBm)	Registro de potencia (código)
3.0	0
2.6	1
2.1	2
1.6	3
1.1	4
0.5	5
-0.2	6
-1.2	7
-2.2	8
-3.2	9
-4.2	10
-5.2	11
-7.2	12
-9.2	13
-12.2	14
-17.2	15

[29] **RF - Radio Frecuencia:** Conocida también como espectro de radiofrecuencia es la porción menos energética del espectro electromagnético.

La comunicación de radio trabaja en la banda ISM. El canal de radio puede ser seleccionado entre los canales 11 (2.405 GHz) y 26 (2.480 Ghz) cada uno separado 5 Mhz. La potencia de la comunicación de radio como se observa en la Tabla 1.4 es programable de 3 dBm a -17.2 dBm.

1.6.1.3 Tarjetas sensoras y conectores expansibles

MEMSIC provee una amplia variedad de tarjetas para la adquisición de datos utilizando el conector de 51 pines como se observa en la **Figura 1.21**, el mismo que será acoplado a la tarjeta MDA300CA utilizada en este trabajo de titulación.

Figura 1.21 Conector de 51 pines [14]

1.6.2 MIB520 TARJETA DE INTEFAZ USB

La tarjeta MIB520 provee comunicación a la familia de motes IRIS y MICAz para transmisión de datos y programación de los motes. Recibe energía a través de la interfaz USB. La tarjeta a utilizar en este trabajo de titulación será la MIB520CB que posee un conector USB macho como se muestra en la **Figura 1.22**.

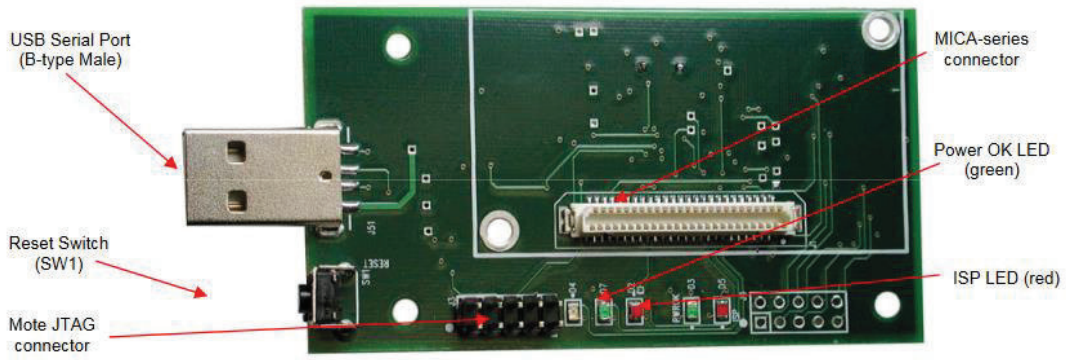


Figura 1.22 MIB520CB[15]

La tarjeta MIB520CB posee un procesador Atmega 16L denominado ISP[30] y se encarga de cargar el código desarrollado por el programador al nodo sensor mediante una conexión USB. Para la programación el fabricante recomienda tener instalado TinyOS [15].

1.6.2.1 Ftdi usb

MIB520CB utiliza los drivers FTDI USB para utilizar el puerto USB como un puerto virtual COM, es necesario instalar el driver FT2232C VCP, los 2 puertos virtuales que utiliza MIB520CB son para Linux el ttyUSB(n) y el ttyUSB(n+1), el puerto ttyUSB(n) es utilizado para la programación del mote y el ttyUSB(n+1) para la comunicación. El manual de instalación de los drivers de acuerdo al fabricante se encuentran en el Anexo A.

1.6.2.2 Reset

El botón RESET reinicia el ISP y el mote, además reinicia el software que se ejecute en el computador.

1.6.2.3 Energía

La energía la obtiene de la interfaz USB del host, es necesario apagar el mote antes y mientras esté conectado a la tarjeta MIB520.

1.6.2.4 Interfaz usb

MIB520 ofrece 2 puertos separados, el primero de ellos dedicado a la programación y el segundo a la comunicación con el mote[15].

1.6.2.5 Mda300ca

Está diseñada como una plataforma de propósito general en la que se pueden agregar diferentes sensores analógicos, en la **Figura 1.23** se puede apreciar la tarjeta que será utilizada en el desarrollo del presente trabajo de titulación.

Entre las características de esta tarjeta tenemos 7 canales ADC de 12 bits que detectan voltajes entre 0 y 2.5V con una precisión de 0.6mV, un voltaje máximo en

[30] **ISP - *in-system processor***: Procesador utilizado para un sistema embebido, es decir, es un procesador utilizado para cumplir funciones dedicadas dentro de un sistema más complejo.

las líneas analógicas de entrada correspondiente al valor de Vcc+0.5V y drivers disponibles para TinyOS[16].

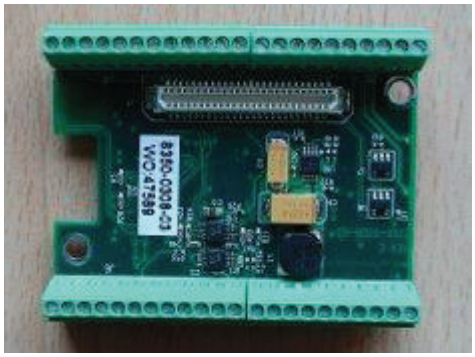
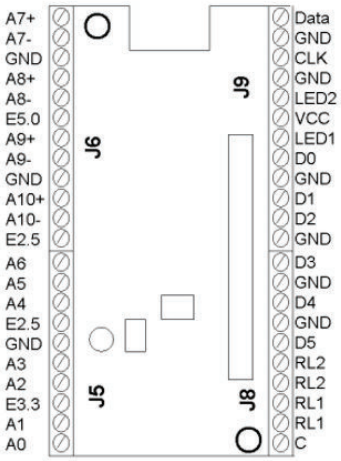


Figura 1.23 Tarjeta de adquisición de datos MDA300CA

En la **Figura 1.24** se muestra los pines de esta tarjeta.



PIN	Descripción
A0-A6	Entradas analógicas
A7-A11	Entradas diferenciales analógicas
DATA	I2C Datos
CLK	I2C Reloj
D0-D6	Líneas digitales
C	Canal de conteo
LED	1: Led Rojo 2:Led Verde
E	Excitación 2.5V, 3.3V y 5 V
Vcc	Vcc del mote
RL1	Relé normalmente abierto
RL2	Relé normalmente cerrado

Figura 1.24 Pines MDA300CA [17]

1.6.3 SENSOR DE PULSO

El sensor de pulso para el presente trabajo de titulación fue adquirido de la página web pulsesensor.com y se muestra en la **Figura 1.25**. Este sensor posee 3 terminales; 2 para alimentación y uno para la salida de la señal.

Fue seleccionado por presentar un bajo consumo de corriente en relación con un sensor fabricado a partir de dispositivos electrónicos, un tamaño pequeño, trabajar

con voltajes que pueden ser suministrados por el nodo, un precio de 12 dólares americanos y por su disponibilidad para la realización de este trabajo de titulación.

Figura 1.25 Sensor de pulso cardiaco[18]

Las características de este sensor se muestran en la **Tabla 1.5**.

Tabla 1.5 Especificaciones del sensor de pulso[18]

Parámetro	Valor
Diámetro	~ 16 mm
Ancho	~ 3 mm
Longitud del cable	~609 mm
Voltaje	~ 3 [V] a 5 [V]
Corriente	~4 mA

1.7 TINYOS

TinyOS es un sistema operativo de peso ligero diseñado específicamente para sensores inalámbricos de bajo consumo de energía, para el desarrollo de aplicaciones TinyOS utiliza una programación orientada al componente y se utilizará en el desarrollo de este trabajo de titulación.

TinyOS tiene mecanismos enfocados en el ahorro de energía y permite un desarrollo fácil de aplicaciones, provee un conjunto de servicios y abstracciones como: censado, comunicación, almacenamiento y temporizadores. Define un modelo de ejecución concurrente para que los desarrolladores puedan construir aplicaciones sin preocuparse por interacciones imprevistas.

TinyOS es ejecutado sobre una docena de plataformas genéricas, muchas de ellas fácilmente soportan agregar nuevos sensores. La estructura de TinyOS hace que sea razonablemente fácil de portar a nuevas plataformas.

Las aplicaciones de TinyOS y el mismo sistema operativo son escritos en nesC. El lenguaje nesC es un lenguaje derivado de C con algunas características para reducir RAM[31], el tamaño del código y optimizarlo, es ideal para sistemas embebidos[32], este lenguaje permite optimizaciones significativas y ayuda a prevenir errores de bajo nivel como las condiciones de carrera[33][19].

1.7.1 SISTEMAS Y APLICACIONES CON TINYOS

En un alto nivel TinyOS posee tres características para escribir sistemas y aplicaciones fácilmente:

1.7.1.1 Componente modelo

El componente modelo define como escribir pequeñas y reusables piezas de código para componerlas en abstracciones más grandes.

El componente modelo es basado en nesC. Esto permite escribir piezas de código reutilizable declarando explícitamente sus dependencias. Por ejemplo, un componente botón genérico de usuario que nos dice cuando un botón es presionado se encuentra en la parte superior de un manejador de interrupciones. El componente modelo permite que la implementación del botón sea independiente de la interrupción, de esta manera puede ser utilizado en diferentes plataformas de hardware[19].

1.7.1.2 Modelo de ejecución concurrente

El modelo de ejecución concurrente define como los componentes intercalan sus cálculos y como interrumpen o no interrumpen la interacción del código, permite a

---

[31] **RAM - Memoria de acceso aleatorio ó *Random Access Memory***: Memoria de tipo volátil que contiene las instrucciones ejecutadas en la unidad central de procesamiento.
[32] **Sistema Embebido:** Sistema de computación diseñado para realizar unas pocas funciones dedicas
[33] **Condiciones de carrera**: Estado en el que los resultados dependen del tiempo de espera para acceder a un recurso.

TinyOS soportar muchos componentes que necesitan actuar al mismo tiempo que requieren poco RAM. Primero, cada llamada de entrada/salida en TinyOS es en fase dividida: en lugar de esperar la ejecución de un bloque hasta su finalización, una respuesta regresa inmediatamente y quien realizo la llamada obtiene una devolución de llamada cuando completa las entradas/salidas. Dado que la pila no está atada a la espera de Entrada/Salida para completar las llamadas, TinyOS sólo requiere una pila, y no tiene hilos.

Las tareas, que son de peso ligero difieren procedimientos de llamada [19]. Cualquier componente puede publicar una tarea, la cual TinyOS ejecutará tiempo después. Debido a que los dispositivos de baja energía deben pasar la mayor parte de su tiempo durmiendo, tienen una baja utilización de CPU y en la práctica las tareas suelen ejecutarse muy pronto después de ser publicadas (en unos pocos milisegundos). Además, debido a que las tareas no pueden adelantarse unas a otras, el código de tarea no necesita estar preocupado por carreras de datos.

1.7.1.3 Simplificación de la escritura

TinyOS posee API[34], servicios, librerías del componente y una estructura de componentes en general que simplifican la escritura de nuevas aplicaciones y servicios. TinyOS posee un conjunto de API para la funcionalidad común como enviar paquetes, leer sensores y responder a eventos.

Para la programación de interfaces, TinyOS también prosee una estructura de componentes y librerías de componentes. Parte de la estructura de componentes incluye cerraduras de recursos, los cuales permiten el funcionamiento automático en baja potencia, y las librerías de componentes simplifican la escritura de las cerraduras.

1.8 NESC

Es un lenguaje deriva del lenguaje C y utiliza componentes, estos componentes deben ser enlazados para el desarrollo de aplicaciones.

[34] **API:** Interfaz de programación de aplicaciones

1.8.1 ELEMENTOS BÁSICOS

1.8.1.1 Componente

Son segmentos de código encargados de realizar tareas determinadas y pueden ser de 2 tipos, primitivos o complejos.

Los componentes primitivos son intrínsecos a TinyOS. Cuando el programados crea nuevos componentes a partir los componentes primitivos, estos nuevos componentes se conocen como complejos.

Para acceder a los distintos componentes se utilizan interfaces.

Las aplicaciones son el resultado de la organización de varios componentes. La función que tiene cada componente es especificada en un fichero de código fuente, como sugerencia este fichero lleva el mismo nombre que el componente.

Cuando se nombran los componentes estos no pueden repetirse, el compilador de nesC utiliza estos nombres para acceder a los componentes y en caso de repetirse se generaría errores de compilación.

Por su comportamiento los componentes se clasifican en módulos y configuraciones.

Los componentes se dividen en bloques como son la signatura y la implementación, la implementación es la parte del componente que lo distingue en módulos y configuraciones[19].

1.8.1.2 Interfaz

Por ellas se accede a los componentes, permiten también la recepción de eventos para generar respuestas específicas por parte del componente[19].

1.8.1.3 Cableado

Las aplicaciones desarrolladas en nesC son el resultado de la unión de varios componentes. *Wiring* en inglés, o cableado es la operación de relacionar las interfaces de los distintos componentes para generar aplicaciones con un nivel superior[19].

1.8.2 NOMBRES Y ESTRUCTURA DEL PROGRAMA

La estructura del programa es la parte más esencial y una obvia diferencia entre C y nesC. Los programas en C están compuestos de variables, tipos y funciones definidas en archivos que son compiladas separadamente y después se la junta. Los programas de nesC se los construye en base a componentes que son conectados (*wired*) por declaraciones explícitas en el programa.

Para demostrar las diferencias entre C y nesC, así como algunas características de nesC se implementarán 2 programas simples a manera de “Hola Mundo” como aplicaciones para el nodo, Encendido que iniciará el nodo y encenderá un LED y Parpadeo que encenderá el nodo y comenzará a parpadear un LED de manera repetida.

1.8.2.1 Aplicación: hola mundo

El equivalente más cercano de “Hola Mundo” para un nodo es el encendido de un diodo emisor de luz o LED posterior al encendido del nodo y después que proceda a dormir. Una implementación de este tipo en C como se observa en el Segmento de código 1.1 es relativamente simple:

```
#include “mote.h”
int main()
{
    mote_init();
    led0_on();
    sleep  ();
}
```

Segmento de código 1.1 Encendido de LED en C

La aplicación de Encendido del LED en C es compilada y unida con una librería “mote” que provee las funciones para la inicialización del hardware (mote_init) y coloca al mote en modo *sleep* de bajo consumo de energía. La cabecera “mote.h” provee declaraciones de estas y otras funciones básicas. La función *main* es llamada automáticamente cuando el mote es encendido.

```
module EncendidoC {
    uses interface Boot;
    uses interface Leds;
}
implementation {
    event void Boot.booted (){
      call Leds.led0On();
    }
}
```

Segmento de código 1.2 module EncendidoC en nesC

La aplicación de nesC de encendido está separada en 2 partes. La primera parte, módulo EncendidoC, contiene la lógica ejecutable del encendido[19], en el Segmento de código 1.2 se puede observar la implementación.

El código del módulo de Encendido del LED en nesC menciona que EncendidoC interactúa con el resto del sistema por dos interfaces, *Boot* y LEDs, y provee una implementación para el evento *booted* del encendido del mote de la interfaz *Boot* que llama al comando led0On de la interfaz “Leds”. Si comparamos con el código en C podemos observar que la implementación del evento *booted* toma lugar en la función *main*, y la llamada del comando led0On toma lugar en la función led0_on.

En este código se puede contrastar las diferencias entre C y nesC, donde el programa en C está compuesto de funciones, los programas en nesC son construidos en base a componentes que implementan un servicio en particular. Además, las funciones en C típicamente interactúan llamándose directamente, mientras que las interacciones entre componentes son especificadas por interfaces, es se observa en el Segmento de código 1.3: La interfaz de usuario hace un pedido o *request* (llama comandos) en la interfaz del proveedor, el proveedor por su parte devuelve las llamadas (señal de evento) a la interfaz de usuario.

Los comandos y eventos son por sí mismos como funciones regulares (pueden contener arbitrariamente código C), llamar a un comando o una señal de evento es

solo una función de llamada. EncendidoC es un usuario de ambos *Boot* y Leds; el evento *booted* es una devolución de llamada cuando el nodo es encendido, mientras que led0On es un pedido que llama al comando que enciende el LED 0.

```
interface Boot {
    event void booted ();
}

interface Leds {
    command void led0On();
    command void led0Off();
    command void led0Toggle();
    ...
}
```

Segmento de código 1.3 Interfaces simples de nesC

Las interfaces de nesC son similares a las interfaces de Java, con la adición de la palabra *command* o *event* para distinguir las llamadas de sus devoluciones (*request from callbacks*).

La segunda parte del encendido, *configuration* EncendidoAppC en el Segmento de código 1.4 específica como EncendidoC es conectado a los servicios de TinyOS encargados de controlar la funcionalidad de los LEDs.

```
configuration EncendidoAppC { }
implementation {
    components MainC, LedsC, EncendidoC;
    Main.Boot -> EncendidoC.Boot;
    EncendidoC.Leds -> LedsC.Leds;
}
```

Segmento de código 1.4 *configuration* EncendidoAppC en nesC

Esto dice que la aplicación EncendidoAppC es conformada de tres componentes (módulos o configuraciones), MainC (sistema de inicio), LedsC (control de LED) y EncendidoC (nuestro módulo de encendido).

EncendidoAppC explícitamente especifica las conexiones o *wiring* en ingles entre las interfaces provistas y las usadas por estos componentes. Cuando MainC ha terminado de inicializar el sistema, este señala al evento *booted* en EncendidoC. Este evento entonces llama al comando led0On de esta interfaz, la cual está conectada por EncendidoAppC a la interfaz Leds que la provee LedsC. De esta manera la llamada enciende el LED 0. El diagrama de componentes y sus conexiones se muestra en la **Figura 1.26**, este diagrama se genera de manera automática de EncendidoAppC por nesdoc[35].

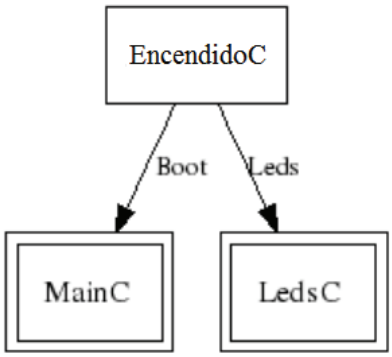


Figura 1.26 Diagrama de conexión de la aplicación de Encendido

EncendidoAppC ilustra la tercera mayor diferencia entre C y nesC: Conexiones explícitas expresadas por el enlace de la versión en C de Encendido con su librearía “mote”. En la versión C, Encendido llama a una función global llamada led0_on que es conectada a cualquier librería que provee una función con el mismo nombre, si dos librerías proveen la misma función, la primera en ser nombrada en el comando de líneas “gana”.

Usando una configuración de nesC, el programa explícitamente selecciona que implementación de componente de función se utilizará.

[35] **nesdoc:** Herramienta de generación automática de documentación que provee TinyOS

El compilador nesC puede tomar ventaja de esta conexión explicita para construir binarios altamente optimizados. La implementación del compilador nesC toma los archivos nesC describiendo componentes como entradas o salidas a un archivo C, como se muestra en la **Figura 1.27**. El archivo en C es pasado a un compilador C nativo que puede compilar al procesador[36] o microcontrolador[37] deseado.

El compilador nesC cuidadosamente construye el archivo C para maximizar la optimización del compilador C. Por ejemplo, desde que se le da un solo archivo, el compilador C puede libremente optimizar a través de los límites de su llamada. El compilador nesC también elimina código muerto que nunca es llamado y variables que nunca son utilizadas. Esto acelera la compilación C y reduce el tamaño del programa en código y requerimientos de RAM.

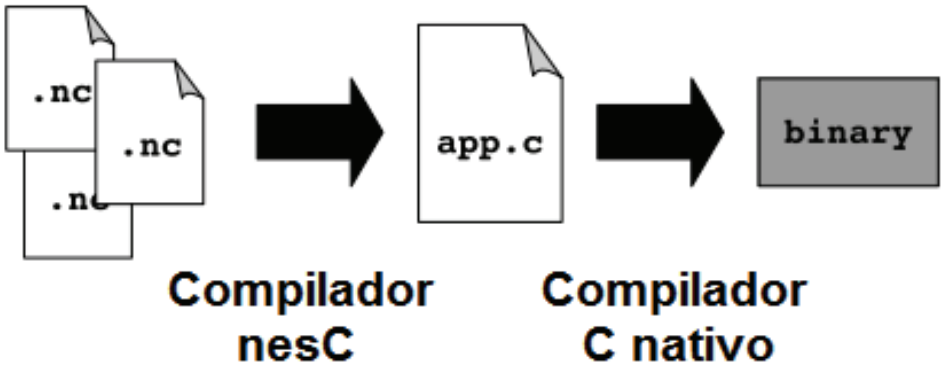


Figura 1.27 Modelo de compilación de nesC[19]

1.8.3 MIG – MESSAGE INTERFACE GENERATOR

El programa *Listen* que viene incluido en la instalación de TinyOS es un programa básico para la comunicación con el mote, este programa imprime los paquetes en hexadecimal en pantalla.

Para obtener los resultados de la **Figura 1.28** se cargó un programa en un mote encargado de enviar un valor autoincrementado de manera periódica.

---

[36] **Procesador:** Elemento que interpreta las instrucciones y procesa los datos de los programas de computadora.
[37] **Microcontrolador:** Circuito integrado programable que ejecuta tareas almacenadas en su memoria.

```
00 FF FF 00 00 04 22 06 00 02 00 01
00 FF FF 00 00 04 22 06 00 02 00 02
00 FF FF 00 00 04 22 06 00 02 00 03
00 FF FF 00 00 04 22 06 00 02 00 04
```

- Dirección destino (2 bytes) = FF FF
- Enlace de origen (2 bytes) = 00 00
- Longitud del mensaje (1 byte) = 04
- ID de grupo (1 byte) = 22
- Tipo de controlador de mensaje activo (1 byte) = 06
- Carga (up to 28 bytes):
    - ID mote origen (2 bytes) = 00 02
    - Contador (2 bytes) = XX XX

Figura 1.28 Paquete recibido por *Listen*

Como los datos se reciben en hexadecimal no es fácil interpretarlos utilizando este programa por lo cual necesitamos otro programa que nos permita observar e interpretar los datos que provienen del mote.

TinyOS *toolchain*[38] hace este brinda ayuda en este proceso al proveer de herramientas para la generación automática de objetos de mensajes a partir de la descripción del paquete.

En lugar de analizar varios paquetes de manera manual TinyOS posee la herramienta MIG o *Message Interface Generator* para construir una interfaz Java, Python o C del mensaje.

Con la secuencia de bytes obtenidos del mote, el código generado por MIG los analiza separando los campos de los mensajes, y proporciona un conjunto estándar de accesorios y modificadores para imprimir paquetes recibidos o generar nuevos.

[38] ***Toolchain***: Conjunto de herramientas

Esta herramienta utilizada para la generación de código que procesa los mensajes de TinyOS posee la estructura indicada en el **Segmento de código 1.5** :

```
mig [any ncc option] [-o output-file] [-java-classname=full-
class-name] [-java-extends=class-name] tool msg-format-file
message-type
```

Segmento de código 1.5 Estructura MIG [20]

El argumento *tool* indica que herramienta debe ser generada, *msg-format-file* especifica que cual es el archivo nesC principal de la aplicación, *message-type* especifica el tipo de mensaje C que se desea procesar. El tipo de mensaje C se debe definir con un "struct message-type" o "*unión message-type"* en un archivo .h con la aplicación nesC 'incluida'. Si el archivo .h no depende de ningún otro archive entonces se puede especificar el archivo .h directamente como el *msg-format-file*.

Solo hay una herramienta, java, que genera una clase java para codificar y decodificar mensajes. Las opciones –java-* son específicas de esta herramienta y se describen a continuación:

- **o:** Se denomina output-file y especifica el archivo de salida del código generado.

Mig trabaja invocando ncc[39] que es una extensión de gcc[40], por lo general será necesario pasar algunas opciones a ncc para que se pueda compilar el archivo.

Algunas opciones comúnmente necesarias son:

- ***target*:** Es la plataforma de tinyos y especifica la arquitectura de la aplicación de TinyOS que está generando o recibiendo mensajes. En caso de utilizar tossim[41] se especifica –target=pc.
- **I:** Especifica un directorio adicional para buscar componentes nesC.

[39] **ncc**: Es una extensión a gcc que conoce como compilar aplicaciones nesC
[40] **gcc:** GNU *Compiler Collection* - colección de compiladores GNU
[41] **Tossim:** Simulador de TinyOS

- ***board***: Especifica una o más tarjetas de sensores. Esto actúa para buscar el camino o *path* y los símbolos del procesador que pueden ser necesarios al generar el archivo MIG.

1.8.3.1 Java tool

Esta herramienta genera una clase java que puede codificar y decodificar paquetes TinyOS, basándose en la infraestructura ofrecida por net.tinyos.message. Por cada campo Cnombre de la estructura *message-type,* se generan los siguientes métodos:

- ***get***_Cnombre: devuelve el valor del campo
- ***set***_Cnombre: modifica el valor del campo
- ***offsetBits***_Cnombre: devuelve el *offset*[42] de bits del campo en *message-type*
- ***offset***_Cnombre: devuelve el offset de bytes del campo en message-type
- ***sizeBits***_Cnombre: devuelve el número de bits del campo (no para *arrays*)
- ***size***_Cnombre: devuelve el número de bytes del campo (no para *arrays*) (ausente si Cnombre es un campo de bits[43])
- ***isSigned***_Cnombre: devuelve *true* si Cnombre tiene signo
- ***is*Array**_Cnombre: devuelve *true* si Cnombre es un *array*

La herramienta java *tool* acepta las siguientes opciones:

- **java-*classname*=*full-class-name***

Esta opción es requerida y especifica el paquete y el nombre de la clase que lo genera. Si el nombre de clase completo no tiene '.', entonces no se incluye ninguna directiva de paquete en la salida.

- **java-*extends*=*class-name***:

Especifica la clase que la clase generada extenderá. La clase por defecto es net.tinyos.*message.Message*.

---

[42] **Offset:** entero que indica la distancia desde que un objeto inicia hasta un punto dado del objeto
[43] **Campo de bits:** Número de ubicaciones de memoria asignadas para contener una secuencia de bits.

En el **Segmento de código 1.6** se ve un ejemplo de utilización de la clase MIG, fue extraído del presente trabajo de titulación.

```
CFLAGS += -I$(TOSDIR)/sensorboards/mda300ca/ -I
mig java -target=null $(CFLAGS) -java-classname
=NodoSensorMensaje NodoSensor.h NodoSensorMensaje -o
NodoSensorMensaje.java
```

Segmento de código 1.6 Nodo sensor MIG

1.8.4 GCC – MSP430

Como parte de la instalación de TinyOS uno de los 5 pasos principales es la instalación de los compiladores nativos, como estamos compilado código para microcontroladores de bajo consumo es necesario compiladores que puedan generar el código de montaje apropiado, para este trabajo de titulación necesitamos MSP430 *toolchain* [21]. MSP430™ y MSP432™ son compiladores de código abierto que incluyen depuradores de C / C ++ utilizados para de aplicaciones que utilizan microcontroladores MSP430 y MSP432 [22], son propiedad de Texas *Instrument* y mantenidos por la empresa Somnium. Las características de este compilador son:

- Código fuente gratuito
- CC[44] y binarios de GDB[45] para Windows, Linux y Mac OS X
- Soporte de depuración de MSP430
- Archivos de cabecera y enlazador MSP430
- Sin limitación de tamaño de código

1.8.5 YETI

En la página oficial de TinyOS un editor sugerido para trabajar con TinyOS es Yeti[21], este editor de nesC es implementado como un *plugin* para Eclipse. La lista de características que provee Yeti incluye:

---

[44] **CC - *Compiler Collection*:** Es una colección de compiladores desarrollados por GNU
[45] **GDB - GNU *Debugger*:** Depurador estándar que es utilizado para códigos compilados con GNU

- Resaltado de sintaxis
- Validación de código en tiempo real
- Finalización de código
- Herramientas de búsqueda

Incluye un depurador experimental que trabaja con avr-dbg y avarice pero con capacidad de soportar otros GDB. Está diseñado para trabajar con TinyOS 2.x y el soporte para TinyOS 1.x es limitado.

1.9 JDBC – JAVA DATABASE CONNECTINITY

En el presente trabajo de titulación se pretende conectar una aplicación desarrollada en Java con una base de datos MySQL, la opción sugerida en la página oficial de MySQL es la utilización de JDBC[46].

JDBC es una interfaz de programación de aplicaciones o API que permite desde Java ejecutar operaciones con bases de datos utilizando un lenguaje SQL[47]. JDBC es presentada como un grupo de interfaces Java y métodos que gestionan los manejadores de conexión hacia cada modelo de base de datos. Estos manejadores consisten en clases con un grupo de interfaces Java que utilizan métodos de registro para declarar los tipos de localizadores a base de datos o URL que pueden manejar

Cuando se desea utilizar una base de datos en un programa, se ejecuta la biblioteca que corresponda al modelo de base de datos utilizado. Se accede a la base de datos estableciendo una conexión, después se puede realizar cualquier tipo de consultas en la base de datos.

1.10 JAVA

Java es un lenguaje orientado a objetos diseñado para poseer el mínimo número de dependencias posible permitiendo que un programa sea ejecutado fácilmente en cualquier dispositivo, es decir que el código no dependa de la plataforma en la que

[46] **JDBC - JAVA *DataBase Connectivity*:** Driver que permite la comunicación entre una base de datos de MySQL con una aplicación desarrollada en JAVA

[47] **SQL - *Structured Query Language*:** Es un lenguaje de consulta estructurada utilizada para modificar y realizar consultas en una base de datos.

se ejecute. Cabe diferencia Java de JavaScript, este último es una tecnología utilizada para el desarrollo de páginas web y se ejecuta en el explorador.

Java fue originalmente desarrollado por *Sun Microsystems*, compañía que posteriormente fue adquirida por Oracle. El lenguaje utilizado en Java deriva de los lenguajes C y C++.

Las aplicaciones desarrolladas en Java se compilan a *bytecodes* (clase Java) que son ejecutados por la máquina virtual de Java (JVM) por sus siglas en inglés, independientemente de la arquitectura del computador.

1.10.1 PARADIGMA ORIENTADO A OBJETOS

Orientado a objetos se refiere a un método de programación en el cual funciones y métodos se combinan en entidades denominadas objetos, en este objeto se encuentra el comportamiento y el estado de una entidad o paquete. Esta separación en objetos facilita la gestión de trabajos y permite la realización de proyectos mediante la reutilización de código. De igual manera se produce una menor cantidad de fallos.

1.10.2 MULTIPLES SISTEMAS OPERATIVOS

Para que el código generado tenga independencia de plataforma, se compila un código llamado Java *bytecode* que comprende una serie de instrucciones simplificadas específicas de la plataforma Java. El destino final es el lenguaje de máquina del dispositivo destino, esto se lo realiza con la ayuda de la máquina virtual de java que consta de un código escrito utilizando código del dispositivo destino. La conversión del *bytecode* a código de máquina es realizada por el compilador JIT.

1.10.3 RECOLECTOR DE BASURA

Durante el desarrollo de una aplicación en Java el programador establece la creación de los objetos y no debe preocuparse por la liberación de los mismos, esta tarea está a cargo del entorno de ejecución de Java o Java *runtime* que se encarga de la gestión del ciclo de vida de los objetos. Cuando las referencias a los objetos desaparecen entra en acción el recolector de basura eliminando el objeto.

1.10.4 JAVA DEVELOPMENT KIT - JDK

Comprende el software necesario para el desarrollo de aplicaciones Java, se puede instalar en una unidad de red con las distintas herramientas distribuidas o en su defecto en una computadora.

En el seno del JDK suele encontrarse el Java *Runtime Environment* incluyendo herramientas como el compilador, Javadoc o el depurador.

1.10.5 JAVA RUNTIME ENVIRONMENT - JRE

El JRE es el entorno de ejecución de la aplicación, contiene el software necesario para que el usuario final pueda ejecutar cualquier aplicación desarrollada en Java. Se puede encontrar en el JDK o de manera independiente.

1.11 MYSQL

MySQL fue desarrollado por Oracle *Corporation* y es un sistema utilizado para la gestión de base de datos relacional[48], muy popular para el trabajo de desarrollo web ya que se asocia con varias aplicaciones que se basan en la web y publicación en línea, una pila de aplicaciones empresariales de código abierto denominados LAMP[49] que utilizan Linux como sistema operativo.

Su código es abierto y se basa en el lenguaje de consulta estructurado SQL. Surgieron varias ramificaciones durante la creación de MySQL denominadas *forks* como MariaDB, Drizzle o Percona Server con XtraDB.

1.12 SCRUM

Scrum es el nombre utilizado para describir a marcos para el desarrollo y mantenimiento de productos complejos.

Scrum mediante la creación de equipos auto-organizados impulsando la comunicación verbal entre todos los miembros y disciplinas involucradas en el trabajo

---

[48] **Base de datos relacional:** Es una colección de elementos de datos organizados en un conjunto de tablas formalmente descritas.

[49] **LAMP:** Acrónimo que describe una infraestructura de internet que utiliza las siguientes herramientas Linux, Apache, MySQL y Python

de titulación, es de utilidad en entornos de trabajo inestables, que ameritan rapidez y flexibilidad.

El desarrollo de un producto se lo realiza en iteraciones denominadas *Sprint* con una duración máxima de 4 semanas obteniendo un incremento ejecutable del producto para el dueño del producto. Se realizan reuniones durante el trabajo de titulación destacando una reunión diaria de 15 minutos para coordinar objetivos y actividades.

1.12.1 BASES DE SCRUM

Scrum se basa en que el conocimiento proviene de la experiencia y las decisiones en base a lo que se conoce con un enfoque iterativo e incremental, ayudando a predecir resultados y calcular riesgos. Para el control del proceso se aplica:

- Transparencia: Requiere que todos los aspectos relevantes se definan en base a un estándar común, como por ejemplo un mismo lenguaje para quienes realizan el trabajo y aquellos que reciben el producto.
- Inspección: Inspecciones realizadas por el cliente y deben ser de manera periódica para no interferir con el trabajo.
- Adaptación: Es el ajuste de los procesos que se desvían de límites tolerables y pueden ocasionar que se incumpla con el objetivo.

Se recomiendan cuatro eventos formales en cada Sprint para la Inspección y Adaptación, explicados en Eventos de Scrum:

- Reunión de planificación del Sprint – Sprint *Planning Meeting*
- Scrum diario – *Daily* Scrum
- Revisión del Sprint – Sprint *Review*
- Retrospectiva del Sprint – Sprint *Retrospective*

1.12.2 EQUIPO SCRUM – SCRUM TEAM

Scrum sugiere la conformación de equipos llamados Scrum *Teams* y están conformados por el *Product Owner* (Dueño del producto), el *Development Team* (Equipo de desarrollo) y un Scrum *Master* (Facilitador). Los equipos deciden como

llevar a cabo el trabajo, son auto organizados y multifuncionales. La entrega de los productos es de manera iterativa e incremental generando la posibilidad de obtener experiencia en base a retroalimentación. Se garantiza la disposición de un producto potencialmente útil y funcional. En la página 118 se presenta el equipo de trabajo y los roles asignados en el presente trabajo de titulación.

1.12.2.1 Dueño del producto - product owner

Es el encargado de maximizar el valor del producto y del Equipo de Desarrollo, es el encargado del *Porduct Backlog* o Lista del Producto, esta gestión incluye:

- Expresar los elementos de la lista
- Ordenar los elementos para alcanzar los objetivos de manera óptima
- Optimizar el desempeño del Equipo de Desarrollo
- Asegura la transparencia, visibilidad y claridad de la lista para todos indicando lo siguiente en lo que debe trabajar el equipo
- Asegura que el equipo entiende los elementos de la lista

1.12.2.2 Equipo de desarrollo - development team

Tienen por misión entregar un incremento o adelanto del producto terminado que se pueda poner en producción al final de cada Sprint. El equipo gestiona y organiza su trabajo de manera independiente, entre las características del equipo están:

- Son auto organizados a un nivel que ni el Scrum *Master* puede interferir
- Son multifuncionales al incluir miembros de varias disciplinas
- Todos los miembros sin reconocimiento de su título son desarrolladores sin excepción.
- No existen sub-equipos, sin excepción
- La responsabilidad recae sobre el equipo y no sobre sus miembros.

Los equipos pequeños reducen la interactividad mientras que los grupos grandes requieren demasiada coordinación, este es un aspecto a tener en cuenta al diseñar los equipos de desarrollo.

1.12.2.3 Scrum master - facilitador

Es el responsable de que el Scrum ha sido comprendido y adoptado por los participantes. Es el líder y ayuda a los involucrados externos al equipo a comprender las interacciones que pueden ser con el equipo beneficiosas y cuáles de no. Ayuda a modificar las interacciones para optimizar el valor generado por el equipo. El Scrum *Master* ofrece servicios a:

- El dueño del producto: Gestionando la lista del producto adecuadamente de manera que sea entendible la necesidad de contar con los elementos lista, y asegurarse de que el dueño del producto sepa ordenar la lista para optimizar su valor.
- El Equipo de Desarrollo: Guiándolo para la creación de productos de alto valor, reduciendo impedimentos y guiarlo en entornos en los cuales Scrum no se ha entendido por completo o no ha sido implementado.
- La Organización: Guiando la adopción de Scrum, ayudando en su comprensión a los interesados y trabajando con otros Scrum *Master* en la organización para ser más efectivos.

1.12.3 FASES DE SCRUM

Las fases de Scrum son las siguientes:

1.12.3.1 Planificación

Esta fase comprende:

- Planeación: Scrum establece que se debe crear el *Product Backlog* o lista del producto que posee los requerimientos y sugiere incluir las prioridades y el esfuerzo estimado para llevarlas a cabo. También se crea el equipo y se establecen las herramientas y el sistema de desarrollo.
- Diseño Arquitectónico: Se realiza la arquitectura del producto que será utilizada para definir e implementar requerimientos.

1.12.3.2 Desarrollo

Corresponde a la ejecución de los *Sprint*.

1.12.3.3 Finalización

Corresponde a la integración, pruebas y documentación del producto terminado. El *Product Backlog* queda vacío en este punto.

1.12.4 FUNCIONAMIENTO DE SCRUM

Cabe resaltar que Scrum es ligero, fácil de entender y muy difícil de dominar. Scrum no se debe seguir al pie de la letra en la creación de productos, por el contrario crea un marco de trabajo en el que se puede utilizar diversas técnicas y procesos.

En la **Figura 1.29** se puede apreciar el funcionamiento de Scrum.

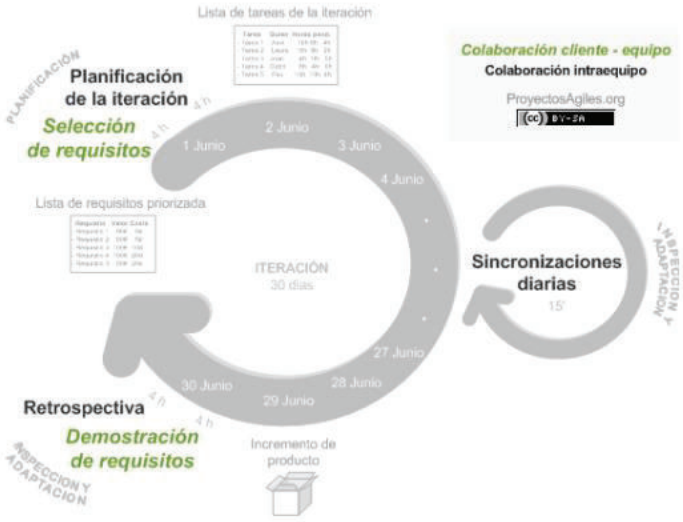


Figura 1.29 Funcionamiento de Scrum [23]

Las reglas de Scrum tienen relación con los eventos, roles y artefactos, siendo principales las relaciones que existen entre ellos. Cada uno de los elementos de Scrum ayuda a un objeto específico necesario para una finalización exitosa.

1.12.5 EVENTOS DE SCRUM

Los eventos son propuestos por Scrum para dar cierta estabilidad evitando reuniones imprevistas, son bloques de tiempo de corta máxima (time-boxes). La duración del Sprint es fija pero los demás eventos poder terminar cuando su objetivo se culmine. Los eventos brindan la oportunidad de retroalimentación, sin ellos la transparencia puede verse afectada.

1.12.5.1 Sprint

Es un bloque de tiempo con una duración máxima de 1 mes para la creación de un producto potencialmente terminado, es el corazón de Scrum, finalizado uno comienza inmediatamente el siguiente, con tiempos mayores un cambio puede aumentar la complejidad y riesgo. Los *Sprint* poseen una reunión de planificación del *Sprint* o *Sprint Planning Meeting*, los Scrum Diarios o *Daily* Scrums, el desarrollo del trabajo, Revisión del *Sprint* o *Sprint Review* y por último la Retrospectiva del *Sprint* o *Sprint Retrospective*. En cada *Sprint* no se debe afectar el Objetivo del *Sprint* o *Sprint Goal*, tampoco disminuir la calidad pero el alcance puede ser renegociado con el dueño del producto.

El único capaz de cancelar un *Sprint* es el dueño del producto y esto puede ocurrir por ejemplo si cambios en la organización conducen a que el *Sprint* quede obsoleto, no tenga sentido o si el equipo subestimo el esfuerzo puede influenciar en el dueño del producto para declarar su cancelación. Los elementos no terminados al cancelar el *Sprint* se vuelven a introducir en la Lista del producto con nuevas estimaciones.

1.12.5.2 Reunión de planificación del sprint – sprint Planning Meeting

Con una duración de ocho horas para un *Sprint* de un mes, esta reunión involucra a todos los miembros del Equipo de Desarrollo y al Scrum *Master* que se asegura de que todos los asistentes comprendan el objetivo. Esta reunión consta de 2 partes:

- Primera: El dueño presenta los requerimientos, prioriza los principales y otorga un nombre a la meta. El equipo despeja dudas con el dueño y selecciona los requerimientos más prioritarios que pueden cumplir en la iteración.
- Segunda: El equipo planifica la iteración estableciendo el *Sprint Backlog* o lista de tareas de la iteración, se estima el esfuerzo en conjunto para cada tarea y cada miembro selecciona las tareas que puede realizar.

1.12.5.3 Objetivo del sprint – sprint goal

Es la meta del *Sprint* y sirve de guía para que el Equipo de Desarrollo comprenda que se está construyendo. Los elementos seleccionados de la Lista brindan una función coherente que puede ser interpretada como el objetivo del *Sprint*[24].

1.12.5.4 Scrum diario – daily scrum

Comprende una reunión de 15 minutos al inicio del día en los que se planifica las actividades a realizarse, teniendo en cuenta el último Scrum Diario y proyectando el trabajo que puede ser completado. Se evalúa la tendencia hacia la finalización del trabajo que se encuentra en la Lista del *Sprint*. Para reducir la complejidad se lo realiza a la misma hora en el mismo lugar, y cada miembro del equipo explica:

- ¿Qué realizó ayer que ayudó al equipo a lograr el objetivo del *Sprint*?
- ¿Qué hará hoy para ayudar al equipo a lograr el objetivo del *Sprint*?
- ¿Existe algún impedimento para lograr el objetivo del *Sprint*?

1.12.5.5 Revisión del sprint – sprint review

Es una reunión informal con el fin de realizar retroalimentación finalizado el *Sprint*. Se revisa el *Sprint* analizando el incremento y se adapta la Lista en caso de ser necesario. Es una reunión que se limita a 4 horas al final de un *Sprint* que ha durado un mes. El Scrum *Master* garantiza que se realice y se comprenda su propósito por parte de los asistentes.

En la revisión se incluye al dueño del producto que explica los elementos de la Lista que se han terminado, el estado actual de la Lista y proyectando fechas de culminación en caso de ser necesario. El Equipo explica si existieron problemas y como se solucionaron, además despeja dudas sobre el incremento realizado. El grupo completo revisa como cambio en el mercado el uso potencial del producto, presupuesto, línea de tiempo, capacidades y decide que hacer a continuación. Esta reunión también busca que la información obtenida en la revisión sea de utilidad para las siguientes revisiones.

Como resultado de la revisión se obtiene una Lista de Producto revisada y define los elementos de la Lista de Producto posibles para el siguiente *Sprint*.

1.12.5.6 Retrospectiva del sprint – sprint retrospective

Se realiza posterior a la Revisión del *Sprint* y antes de la Reunión de Planificación de *Sprint* con un tiempo de 3 horas para un *Sprint* de un mes, el Scrum *Master* debe asegurarse de que se lleve a cabo y la responsabilidad recae sobre él.

En este punto se analiza como resulto el último *Sprint* en relación a procesos, herramientas, personas y relaciones. Se identifica los elementos que resultaron exitosos y sus posibles mejoras. Por último se crea un plan de mejoras que será implementado en el Equipo Scrum para optimizar su trabajo.

Aquí el equipo tiene la oportunidad de revisar su productividad y rendimiento en el último *Sprint* para tomar medidas correctivas en caso de ser necesario antes de la ejecución del siguiente *Sprint*.

1.12.6 ARTEFACTOS DE SCRUM

Son representaciones del trabajo o valor que proporcionan transparencia y facilitan la inspección y adaptación[24].

1.12.6.1 Product backlog

Aquí se encuentran los requerimientos del trabajo de titulación que en parte son las expectativas del dueño del producto. Se encuentra en evolución constante, considerando los requerimientos por prioridad, con la intervención de todos los miembros del equipo. El *Product Backlog* es establecido únicamente por el dueño del producto.

1.12.6.2 Sprint backlog

Se especifican las tareas que se realizarán en cada *Sprint* por parte del equipo, se detallan las tareas con estimaciones de tiempo y recursos asignados.

1.12.6.3 Incremento

Se trata del resultado esperado de cada *Sprint*, se encuentra potencialmente finalizado y puede ser utilizado.

CAPÍTULO II

DISEÑO DEL PROTOTIPO INALÁMBRICO

En este capítulo se iniciará explicando el diseño general del prototipo inalámbrico para el registro de la frecuencia cardíaca, para posteriormente realizar una breve descripción de los diferentes elementos que lo conforman.

2.1 ESCENARIO Y MOTIVACIÓN

La función básica de un sistema de monitoreo de signos vitales es la obtención de información relevante para la toma de decisiones clínicas, la saturación del sistema de salud hace difícil la medición de estas variables de manera continua. A pesar de la existencia de tecnologías inalámbricas como la que se presenta en este trabajo de titulación, con la en capacidad de dar solución al problema, no ha sido posible una implementación masiva en parte por la poca cantidad de personal capacitado y la escaza vinculación entre ciencias de la salud y tecnologías de la información en nuestro medio.

El presente trabajo de titulación pretende demostrar cómo estas tecnologías inalámbricas pueden ser utilizadas, fortaleciendo el vínculo entre ciencias de la salud y las tecnologías de la información, motivando su desarrollo e implementación.

El ambiente en una casa de salud como se muestra en la **Figura 2.1**, consiste en habitaciones con paciente representados por un punto rojo acostados o caminando en ambientes pequeños, esto permite la implementación rápida y ágil de un sistema de monitoreo mediante la instalación de varios nodos Gateway para dar cobertura, reduciendo al mínimo gastos de cableado estructurado.

2.1.1 REQUERIMIENTOS DEL PROTOTIPO

A partir del escenario y motivación explicados anteriormente, el presente trabajo de titulación debe cumplir con las siguientes características:

- Cada nodo debe ser capaz de obtener el valor de frecuencia cardíaca de una persona y enviarlo a al nodo Gateway.

- El nodo Gateway debe conectarse a una red LAN a través de un nodo Gateway.
- El prototipo debe poseer una base de datos que permita el almacenamiento de los datos de cada nodo sensor.
- El prototipo debe soportar por lo menos la transmisión de datos de dos nodos sensores.
- Los datos obtenidos por cada nodo sensor debe almacenarse en una base de datos.
- El prototipo debe permitir por lo menos un equipo con una aplicación desarrollada en Java conectada a la red LAN.
- Una aplicación en Java debe permitir a un usuario visualizar la información de la frecuencia cardíaca en función del tiempo.
- La base de datos, la aplicación en Java y los nodos sensores a través de un Gateway deben tener capacidad de conectarse con una red LAN

Figura 2.1 Esquema de una casa de salud

Los requerimientos anteriormente expuestos están planteados en relación al plan de trabajo de titulación elaborado para el presente trabajo de titulación.

2.2 DESCRIPCIÓN GENERAL DEL PROTOTIPO INALÁMBRICO PARA EL REGISTRO DE LA FRECUENCIA CARDÍACA.

El esquema del prototipo inalámbrico que se implementará será el que se muestra en la **Figura 2.2** compuesto por dos redes interconectadas por medio de un computador Gateway, la primera de ellas es una red LAN y la segunda una red IEEE 802.15.4.

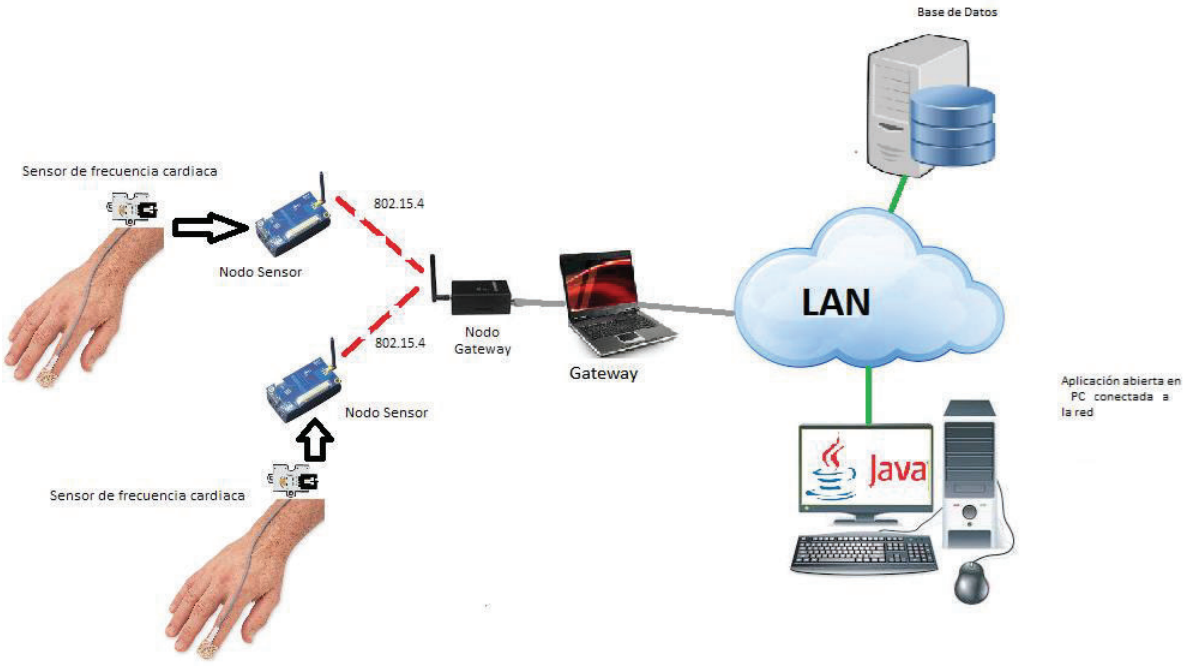


Figura 2.2 Esquema del prototipo inalámbrico para el registro de la frecuencia cardíaca

2.2.1 RED IEEE 802.15.4

Los nodos de la red de sensores o motes que forman parte de este trabajo de titulación serán dos Nodos Sensores y un Nodo Gateway, los nodos son IRIS XM2110 proporcionados por el fabricante Memsic.

Con la ayuda de la tarjeta MDA300CA se realiza la conexión entre el sensor de pulso y el nodo IRIS X2110. Una vez conectado el sensor de pulso el nodo sensor estará

en condiciones de detectar las variaciones de la presión sanguínea en los capilares de una persona utilizando un método no invasivo como es la espectrofotometría[50], por medio del código determinar el valor de la frecuencia cardíaca para posteriormente enviar ese valor a un nodo Gateway.

El Nodo Gateway recibe la información de cada nodo sensor y la envía al computador Gateway.

2.2.2 RED LAN[51]

La red LAN está conectada al computador Gateway y la conforma una Base de Datos encargada de almacenar la información recibida de los nodos sensores y una computadora que contiene una aplicación desarrollada en Java para permitir a un usuario visualizar la información almacenada en la Base de Datos como fue propuesto en el plan del presente trabajo de titulación.

2.2.3 GATEWAY

Se trata de una computadora encargada de comunicar la red LAN con la red IEEE 802.15.4. La comunicación con la red IEEE 802.15.4 se realiza mediante un nodo Gateway conectado por USB.

2.3 DISEÑO DE LA TOPOLOGÍA LÓGICA DE RED.

La disposición y direccionamiento lógicos de los elementos que conforman la red LAN e IEEE 802.15.4 se muestra a continuación.

2.3.1 RED LAN

Para el presente trabajo de titulación se decidió utilizar subredes que facilitarían la administración de la red actual así como de una posible expansión de la misma.

El prototipo trabajará en una red LAN subdividida en 3 subredes (SR) nombradas de la siguiente manera:

---

[50] **Espectrofotometría:** Medición de la energía radiante consumida por un sistema en función de su longitud de onda.

[51] **LAN - *Local Area Network*:** Red de computadoras pequeña que puede abarcar desde una casa hasta un edificio.

- SR_Usuario
- SR_BDD
- SR_Gateway

La base de datos se comunica con el computador Gateway, por lo que la topología lógica utilizada es de tipo estrella como se muestra en la **Figura 2.3**.

La SR_Usuario está conformada por el computador que posee la aplicación de usuario en Java, la SR_BDD está conformada por el servidor de base de datos y la SR_Gateway está conectada el computador Gateway.

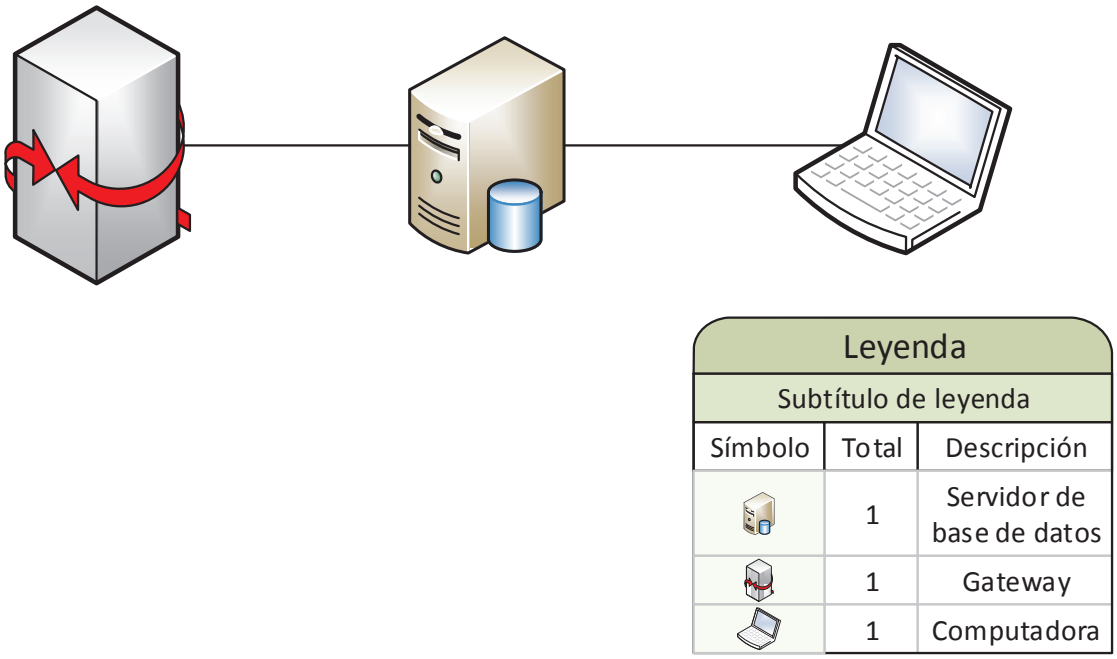


Figura 2.3 Topología Lógica LAN

Cada una de las subredes requiere de dos direcciones IP para su funcionamiento, se utilizarán 3 direcciones IP por cada subred con el objetivo de garantizar una escalabilidad aunque sea mínima, a causa de esto se requiere un total de 9 direcciones IP para la red LAN.

Al ser una red pequeña se utilizará una dirección privada 192.168.1.0/28 de clase C, en la **Tabla 2.1** se encuentra el direccionamiento para cada subred LAN.

Tabla 2.1 Direccionamiento LAN

Subred	Dirección de Red	Dirección de *Broadcast*	Máscara
SR_Usuario	192.168.1.16	192.168.1.23	29
SR_BDD	192.168.1.0	192.168.1.7	29
SR_Gateway	192.168.1.8	192.168.1.15	29

En la **Figura 2.4** se muestra el direccionamiento que será utilizado en el presente trabajo de titulación.

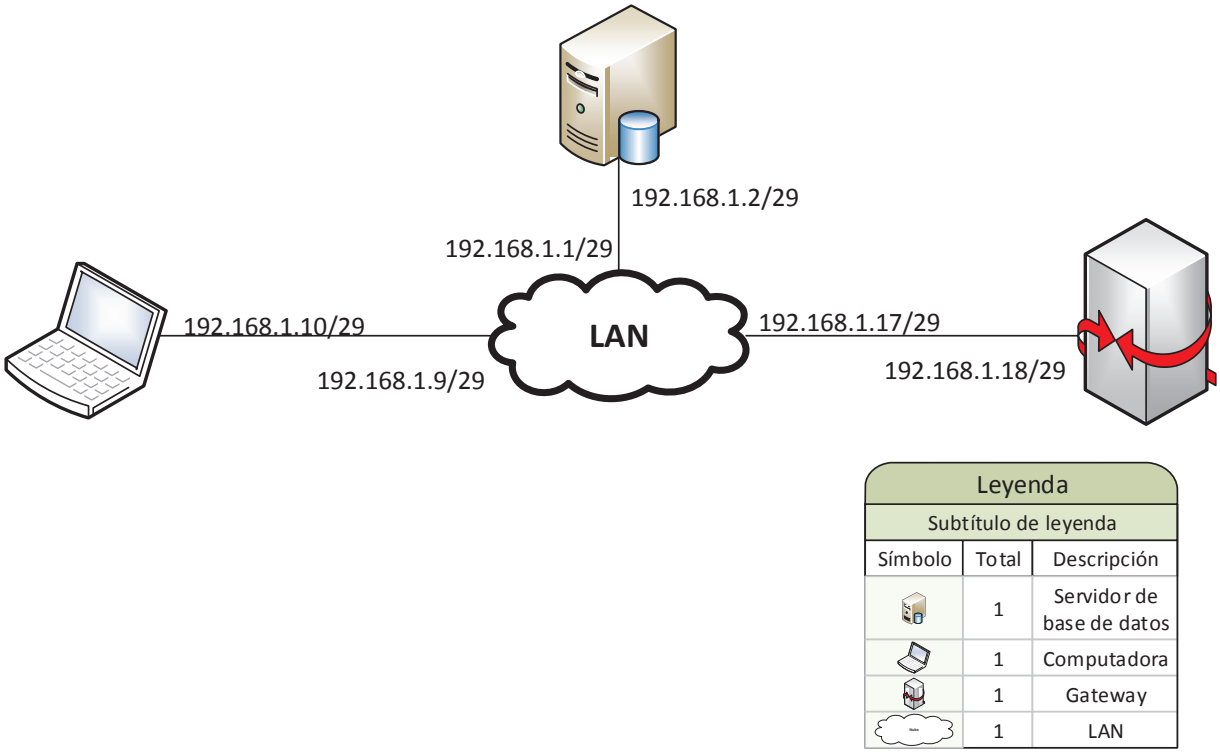


Figura 2.4 Asignación de direcciones en red LAN

2.3.2 RED IEEE 802.15.4

La red IEEE 802.15.4 utiliza un nodo Gateway y dos nodos sensores que serán nombrados como nodo A y nodo B.

El identificador de cada nodo se lo asigna cuando se instala la aplicación, el nodo que será utilizado como Gateway siguiendo las instrucciones del fabricantes deberá poseer el valor de cero [14], los identificadores de cada nodo pueden tomar cualquier valor entre 1 y 65535 como se muestran en la **Tabla 2.2**.

Tabla 2.2 Identificadores IEEE 802.15.4

Nodo	Identificador
Gateway	0
Nodo A	10
Nodo B	15

Para la comunicación entre el nodo Gateway y el computador Gateway se utilizará la aplicación *BaseStation* desarrollado en la universidad de Berkeley California.

2.4 DISEÑO DEL ALGORITMO PARA LOS NODOS SENSORES

Las siguientes consideraciones de diseño permitirán elaborar el algoritmo encargado de recibir las señales de pulso del cuerpo humano, digitalizarlas, calcular la frecuencia cardíaca y enviar el valor al nodo Gateway. Para el presente trabajo de titulación se ocupará fotopletismografía[52] con la ayuda del sensor de pulso cardiaco "Pulse Sensor" que es un proyecto de hardware abierto desarrollado por Joel Murphy y Yury Gitman[18], este sensor permite la detección de las señales de pulso de manera no invasiva y trabajar con voltajes que puede ser suministrado por el nodo sensor.

Para tener una mejor idea de la señal que se ingresa a la tarjeta MDA300CA, se procedió a conectar el nodo sensor y el sensor de pulso a la mencionada tarjeta y con la ayuda de un osciloscopio "Tektronix" TBS 1102B-EDU en la **Figura 2.5** se puede apreciar la señal de voltaje que ingresa a la placa MDA300CA.[25]. En base a

[52] **Fotopletismografía:** Determinación de los cambio de volumen y presión de parámetros orientados al diagnóstico de enfermedades pulmonares o cardiovasculares en base a la medición de la intensidad de la luz reflejada por la superficie de la piel y de los hematíes que están debajo.

las observaciones obtenidas con el osciloscopio y a los fundamentos teóricos adquiridos se procedió a la elaboración de un algoritmo general sobre el funcionamiento que deberá tener el nodo sensor.

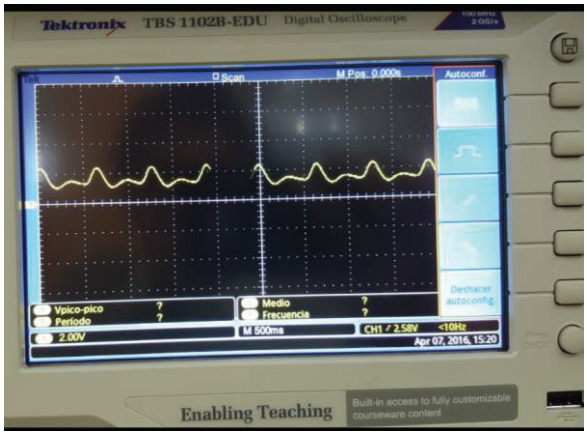


Figura 2.5 Señal de voltaje del sensor de pulso conectado a la placa MDA300CA

El algoritmo para el proceso de obtención y envió del valor de frecuencia cardíaca para cada nodo sensor se muestra en **Figura 2.6**.

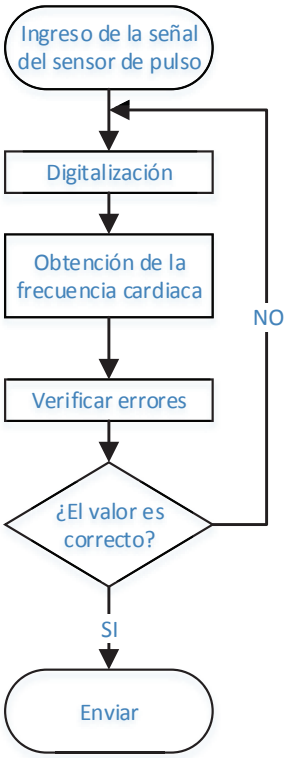


Figura 2.6 Obtención de la frecuencia cardíaca

Cuando la señal de pulso cardíaca ingresa al nodo sensor, como un primer paso se "digitaliza" la señal con 2 estados, uno máximo y uno mínimo que permita contar el número de pulsos de una manera fácil. Se deben establecer 2 umbrales como se muestra en **Figura 2.7** a partir del valor máximo y mínimo de la señal ingresada. Un valor de voltaje que supere el umbral superior será positivo, un valor que sea menor al umbral inferior será negativo y un valor entre ambos umbrales será indeterminado, de las observaciones con el osciloscopio se determinó que el valor mínimo no necesariamente coincide con el voltaje de referencia.

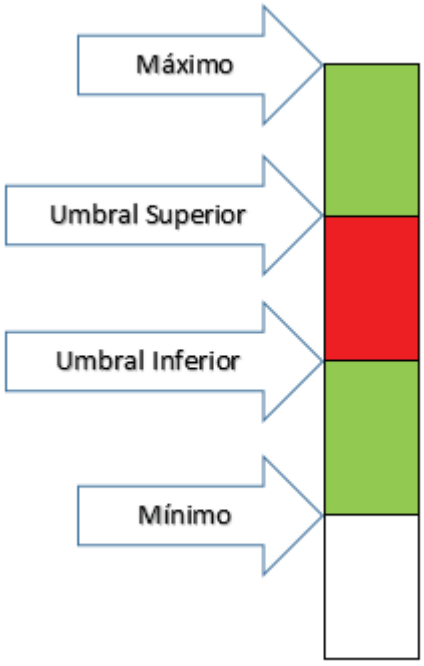


Figura 2.7 Umbrales de muestreo

Para establecer los umbrales se decidió dividir la diferencia de voltaje entre el máximo y el mínimo en 3 partes iguales, con esta consideración la porción superior (verde) será un valor positivo y la porción inferior (verde) un valor negativo. La porción media (rojo) corresponderá a la zona de incertidumbre, mientras que su unión con las porciones verdes corresponderán a los umbrales superior e inferior respectivamente.

Las **Ecuación 2.1** fue creada para ser utilizada durante la programación de los nodos sensores.

$$\text{Promedio} = \frac{\text{Máximo}}{2} + \frac{\text{Mínimo}}{2}$$

$$\text{Umbral Superior} = \frac{\text{Promedio}}{2} + \frac{\text{Máximo}}{2}$$

$$\text{Umbral Inferior} = \frac{\text{Promedio}}{2} + \frac{\text{Mínimo}}{2}$$

Ecuación 2.1 Umbrales

Un pulso ocurrirá cuando la señal pase de positiva a negativa. Una vez establecidos los umbrales se debe establecer si el valor de los umbrales cambiara con el tiempo o no y la frecuencia de muestreo.

Existen varios factores que pueden alterar los resultados de una lectura fiable como es el ajuste del sensor y características propias de la fisiología de cada sujeto como la perfusión de la piel [26], para mitigar estos errores se optará por una calibración constante mediante la actualización periódica de los valores máximos y mínimos. Para la obtención de los máximos y mínimos una señal de pulso debe ser ingresada completamente, una frecuencia mínima de 20 latidos por minuto requiere de 3 segundos para que una señal ingrese totalmente al nodo sensor, deben utilizarse 2 periodos de 3 segundos para asegurarse que una señal de pulso completa a ingresado, tomando en cuenta esto al final de los 6 segundos se calcularán los umbrales superior e inferior y se reiniciarán el máximo y mínimo.

La frecuencia cardíaca máxima es de 250 latidos por minuto, cada señal de pulso requerirá de dos muestras como mínimo para realizar la digitalización. Considerando esto la frecuencia de muestreo debe ser superior a 8,333 muestras por segundo, esto significa que cada muestra de la señal debe obtenerse como máximo cada 120 milisegundos.

Para la obtención de la frecuencia cardíaca la semiología[53] establece que la frecuencia cardíaca se determina contando los latidos en un minuto completo[27], sin embargo de las entrevistas realizadas a profesionales de la salud y considerando que la señal de pulso puede ser irregular se establece que un tiempo de 15 segundos es suficiente. Tomando como referencia el valor de la frecuencia cardíaca

[53] Semiología: Parte de la clínica que se ocupa de buscar e identificar los signos y síntomas de un paciente.

máxima teórica de 250 y mínimo de 20 latidos por minuto para una persona, se desea conocer cuantos segundos son necesarios para que el sensor cuente el número de pulsos de una persona y obtener el valor de pulsos por minutos con un error pequeño.

La **Figura 2.8** muestra los valores de tiempo de muestreo en segundos que se pueden tomar, cuenta el número de pulsos para ese intervalo de tiempo y lo transforma un valor de pulsos por minutos.

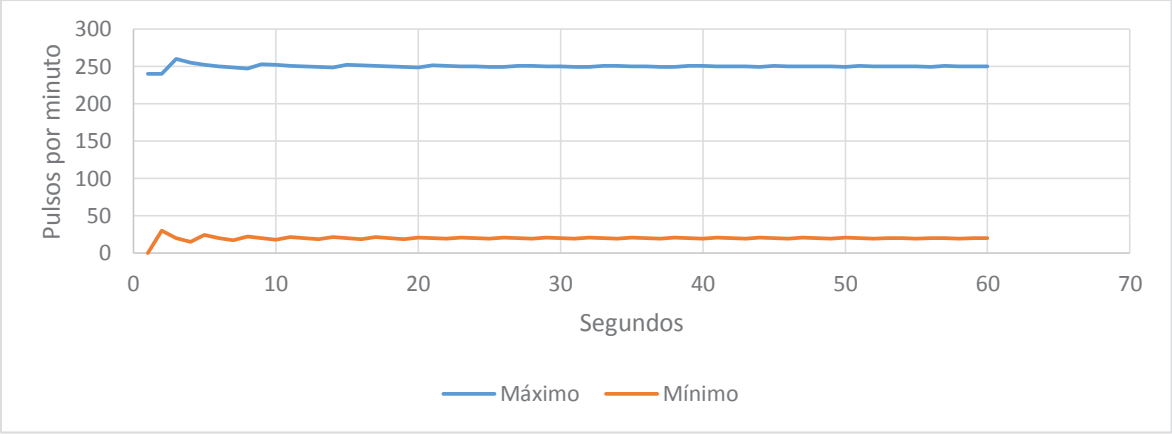


Figura 2.8 Tiempos de conteo de pulsos

En caso una frecuencia máxima (línea azul), las variaciones no son superiores al 4% del valor real, pero en el caso de registrar una frecuencia mínima (línea naranja) las variaciones menores a un 10% se obtienen con tiempos de muestreo superiores a 10 segundos, mientras que con tiempos superiores a los 15 segundos la variación no supera el 7% para valores extremadamente bajos de frecuencia.

Con esta consideración se establece un intervalo de muestreo de 15 segundos es ideal, esto permite en un minuto obtener de 3 a 4 valores que se pueden contrastar entre sí para minimizar errores que es el último punto del diagrama de flujo de la **Figura 2.6** antes de enviar el valor al nodo Gateway.

Con todas estas consideraciones en la **Figura 2.9** se muestra el diagrama de flujo que será implementado en cada uno de los nodos sensores.

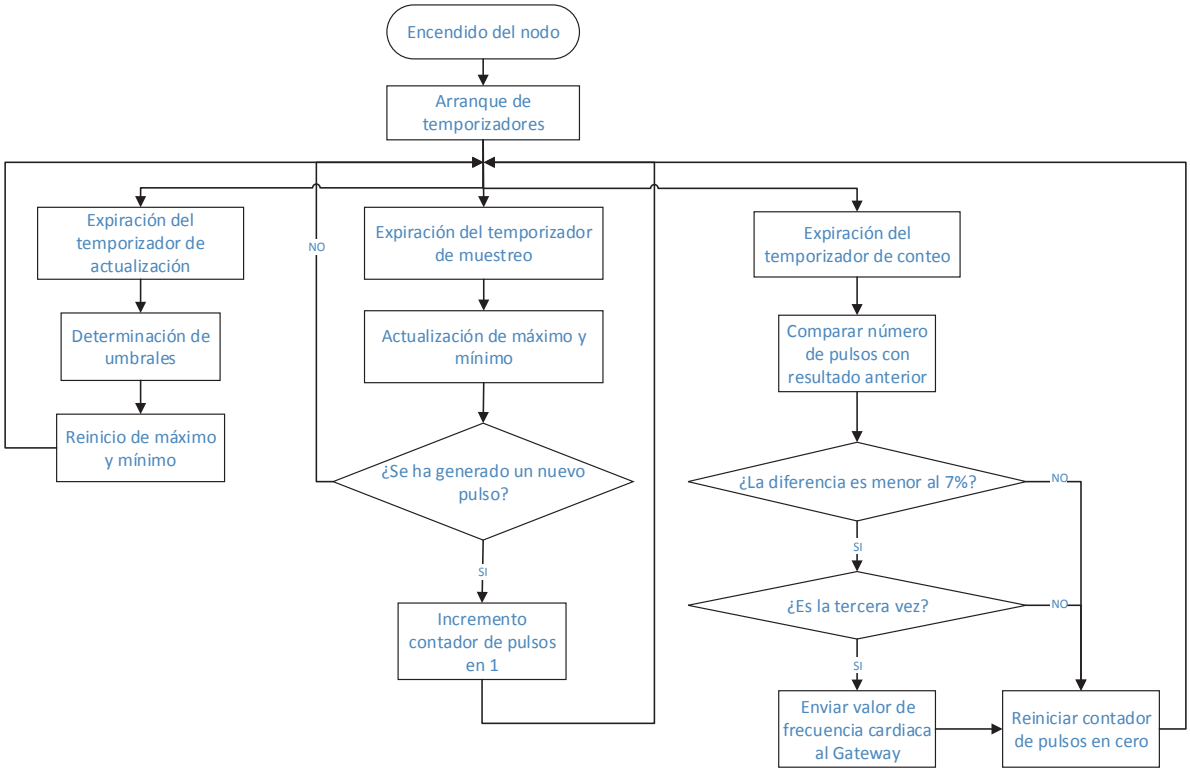


Figura 2.9 Diagrama inicial de flujo de nodo sensor

Al ser implementado el código poseerá 3 temporizadores periódicos de la siguiente manera:

- Temporizador de actualización: Al expirar este temporizador como mínimo cada 6 segundos se calcularán los umbrales y se reiniciarán los valores de máximo y mínimo.
- Temporizador de muestro: Al expirar el tiempo de muestreo debe realizarse una medición como máximo cada 120 milisegundos. Al realizar la medición se intenta detectar cuando se genera un nuevo pulso para aumentar el contador en uno y actualizar el valor de máximo y mínimo.
- Temporizador de conteo: Al expirar este temporizador como mínimo cada 15 segundos se analiza cuantos pulsos se produjeron, este resultado es comparando con el resultado anterior entre 3 y 4 veces buscando similitud en los valores. Si los valores son similares, con una variación máxima de un 7%

el error es pequeño y por lo tanto su valor es enviado al Gateway, si el error es superior no se envía ningún valor al Gateway.

Se programó un LED de una mota para encenderse y apagarse ante variaciones del pulso pero no funcionó por lo que se optó por diseñar una mini aplicación bajo el nombre de Tiempo Real que haría las veces de un osciloscopio para entender que estaba pasando con el censado.

2.4.1 APLICACIÓN TIEMPO REAL

Esta aplicación consta de 2 partes: un código para el nodo sensor con la única funcionalidad de leer el canal un analógico y enviar su valor al Gateway, y una aplicación en Java que grafica los valores obtenidos.

Se agregó un campo al paquete que envían los nodos y se llamó comando que le permite al Gateway determinar que aplicación envió el dato, con el valor de 0 para el prototipo y 1 para la aplicación de tiempo real.

La aplicación en Java permite ingresar el identificador del nodo del cual se desea observar los datos, como resultado de esto en la **Figura 2.10** se puede observar los datos enviados por el nodo sensor cuando se coloca un dedo sobre el sensor de pulso, de estas observaciones se concluye:

- Los niveles de máximo y mínimo suben y bajan lentamente.
- La escotadura mostrada en la **Figura 1.6** se presentaban como una segunda onda más pequeña.
- Existe la presencia de un desnivel con un valor cercano de 500.
- Los movimientos de la persona producen variaciones de la presión sobre el sensor y tienden a generar valores extremos, por ejemplo al presionar el sensor los valores registrados llegan a un valor constante máximo y bajan a un valor de 0 cuando no se coloca nada sobre el sensor, esto dificulta el cálculo de los umbrales.

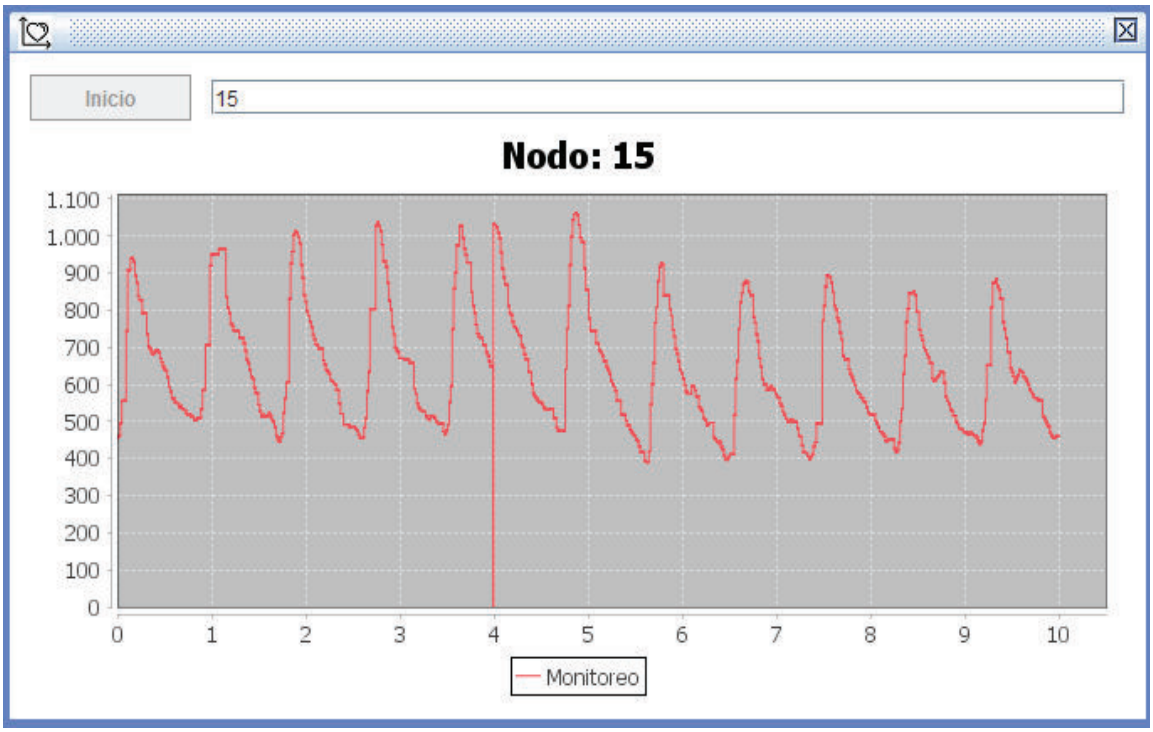


Figura 2.10 Aplicación de Tiempo Real en Java

Para obtener resultados más confiables se decidió amplificar la señal, aumentar la diferencia entre la onda principal y la generada por la por la presencia de la escotadura y reducir ruido. Para conseguir lo anteriormente mencionado se elevó los valores al cuadrado y se restó el desnivel que se mostraba en la **Figura 2.10** en la medida de lo posible como se observa en la **¡Error! No se encuentra el origen de la eferencia.**.

Se divide para 20 con el fin de no tener valores sumamente grandes y poder utilizar variables de menor tamaño en la implementación de la aplicación del prototipo. De la formula anteriormente elaborada se procede a generar un nuevo código para el nodo sensor y su resultado se muestra en **Figura 2.11**, como se observa hay una mayor amplitud y después de observaciones continuas se pudo constatar que el valor de 12000 era el más estable, así que se tomó la decisión de usar solo ese valor como umbral para determinar el pulso.

$$\text{Valor Nuevo} = \left(\frac{\text{ValorRecibido} - \text{Desnivel}}{20}\right)^2$$

$$\text{Valor Nuevo} = \left(\frac{\text{ValorRecibido} - 500}{20}\right)^2$$

Ecuación 2.2 Amplificación de la señal

Con el establecimiento de un umbral se pudo concluir:

- Un valor máximo por excesiva presión sobre el sensor no interfería con la detección del pulso.
- Los valores máximo y mínimo presentaron menores variaciones.
- La señal contenía menor cantidad de ruido.

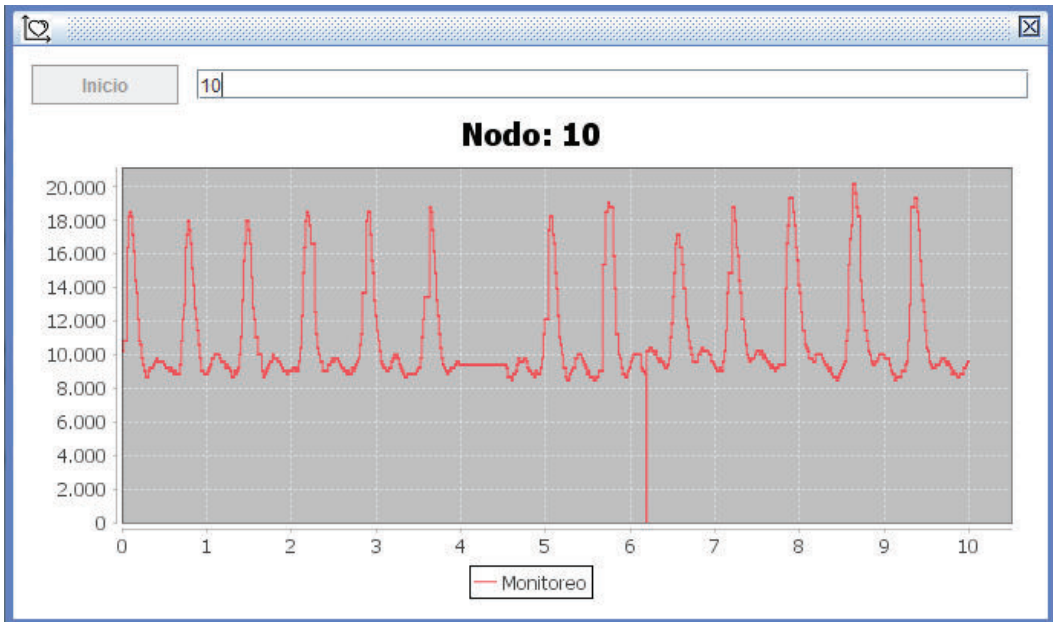


Figura 2.11 Señal amplificada

De lo anteriormente expuesto, el algoritmo del nodo sensor sufre un cambio que se muestra en **Figura 2.12**, en este caso se ocupa un temporizador para el muestro, otro para contar los pulsos y un tercero para descansar. A diferencia del diagrama inicial, el diagrama final no se ve obligado ajustar constantemente los valores de máximo y mínimo, posee un valor de umbral constante permitiendo tolerar movimientos involuntarios pequeños.

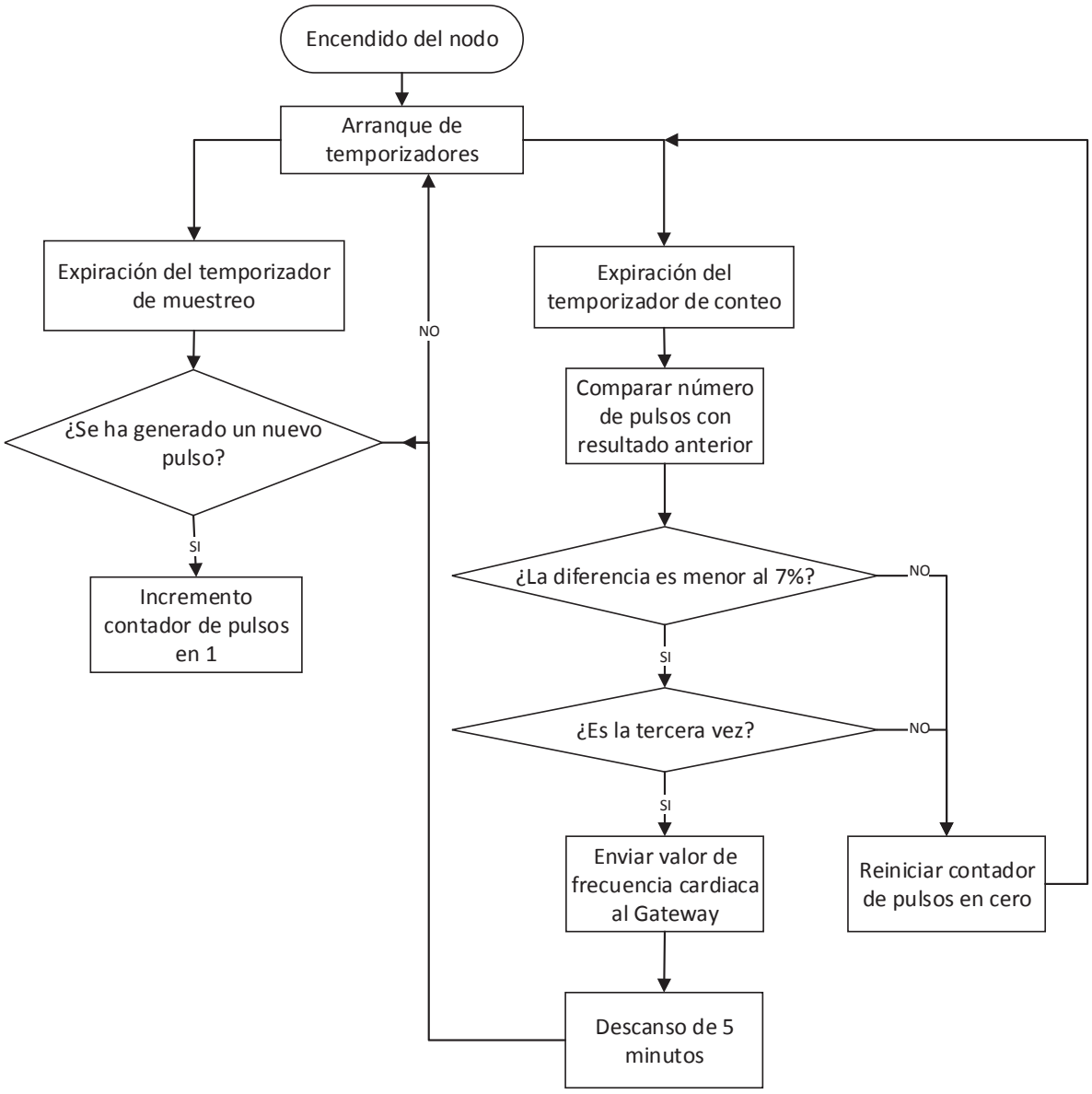


Figura 2.12 Diagrama final del Nodo Sensor

Por la sencillez en el diseño de estos diagramas no se vio la necesidad de implementar algún tipo de ingeniería de software para su desarrollo.

2.5 DISEÑO DE LA APLICACIÓN DE USUARIO

El diseño de la aplicación fue utilizando la metodología SCRUM como se indicó en el plan del presente trabajo de titulación, la implementación con las historias de usuario se muestran en el siguiente capítulo.

La aplicación de usuario tiene por objetivo facilitar la visualización de registros de la frecuencia cardíaca de una persona de una manera gráfica utilizando el lenguaje de programación Java.

2.5.1 NAVEGACIÓN POR APLICACIÓN.

Después de realizadas las entrevistas un primer paso para el desarrollo de la aplicación consisten en el establecer la navegación por la aplicación.

En la **Figura 2.13** se muestra la navegación que tendrá la aplicación de usuario en este trabajo de titulación.

Figura 2.13 Diseño de la navegación de la aplicación de usuario

2.5.2 LÓGICA DEL NEGOCIO

Para evitar un código caótico y que posea cierto orden se decidió utilizar el patrón modelo, vista, controlador para representar la lógica del negocio, como se muestra en la **Figura 2.14**.

Figura 2.14 Modelo vista controlador

Al hacer esto permitimos separar la lógica de datos de nuestra aplicación de la interfaz de usuario y del módulo encargado de gestionar los eventos y comunicaciones, para esto necesitamos de los 3 componentes:

- El componente Modelo que será el encargado de la comunicación con la base de datos, y de archivos de configuración. Este componente utilizará el patrón DAO[54] que define que por cada tabla en la base de datos en la aplicación deben existir 2 clases:
  - Clase de transporte de datos para encapsular los datos
  - Clase de acceso a datos para el manejo de datos
- El componente Vista que es el que interactúa con el usuario.
- Por último el componente Controlador que es un intermediario entre el componente vista y el componente Modelo.

2.5.3 PERDIDA DE NODO

En este punto se vio prudente colocar un indicador que alerte de la pérdida de un nodo. Considerando que la transmisión de la frecuencia cardíaca de cada nodo sensor en condiciones ideales se realizará aproximadamente cada 45 segundos para posteriormente descansar 5 minutos, un fallo en la transmisión no podrá detectarse después de 5 minutos con 45 segundos como mínimo. En base a entrevistas con profesionales de la salud y pruebas realizadas se estima que si un nodo deja de transmitir por 20 minutos será considerado un nodo perdido.

[54] **DAO - Objeto con acceso a datos:** Objeto que permite una interacción entre la base de datos y la aplicación.

2.5.4 INTERFAZ "PRINCIPAL"

En la **Figura 2.15** se muestra la interfaz principal de la aplicación mediante la cual el usuario puede ingresar, eliminar o editar una historia clínica, crear o eliminar un nodo sensor y lo más importante visualizar los registros de la frecuencia cardíaca de cualquier usuario de una manera fácil y sencilla.

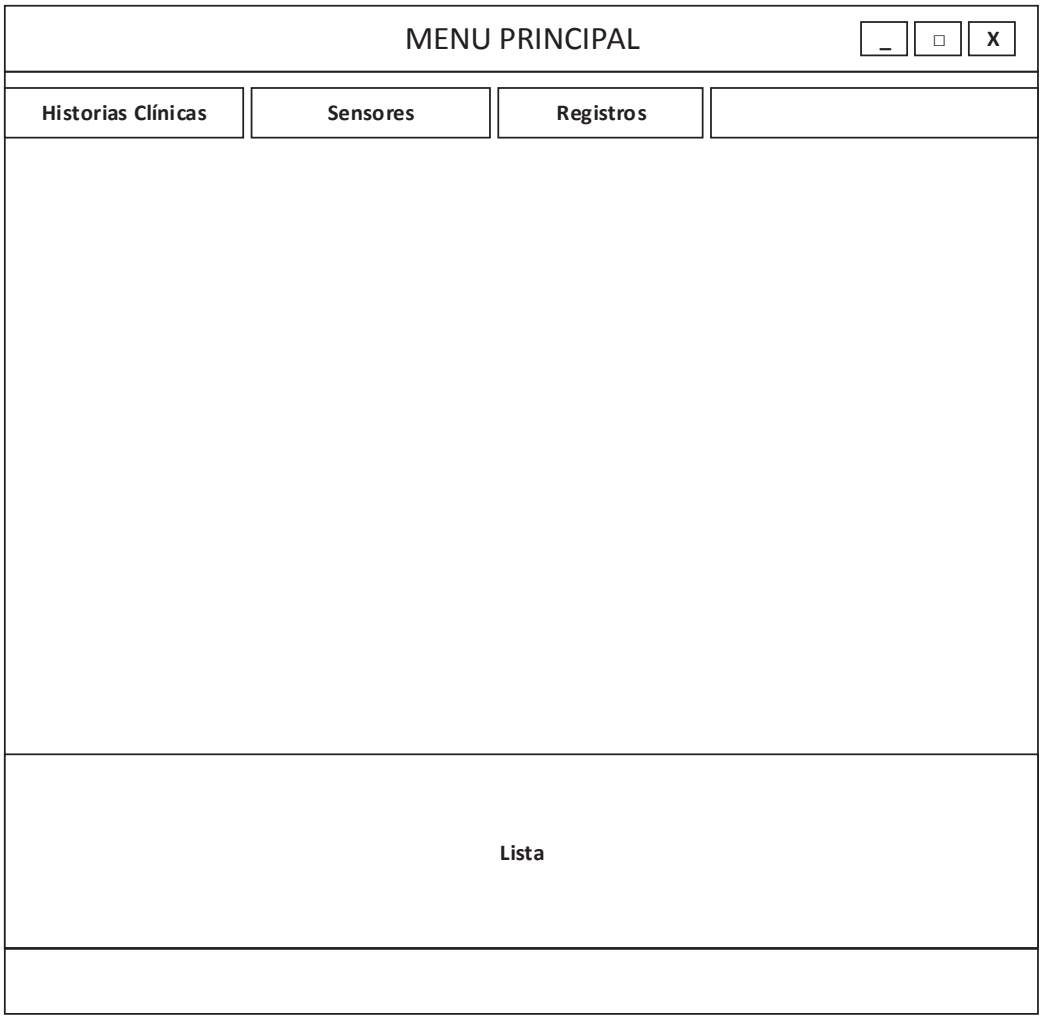


Figura 2.15 Interfaz principal

A continuación se describe cada uno de los elementos que conforman esta interfaz:

- **Botón "Historias Clínicas":** Permite desplegar el menú para crear, eliminar o modificar una historia clínica.

- **Botón "Sensores":** Permite desplegar el menú para agregar o eliminar un nodo sensor.
- **Botón "Registros":** Permite desplegar el menú para visualizar la frecuencia cardíaca de una persona.
- **Lista:** Muestra los nodos que han dejado de transmitir, perdidos.

2.5.5 INTERFAZ "NUEVO"

En la **Figura 2.16** se muestra la apertura del formulario para el ingreso de una nueva historia clínica.

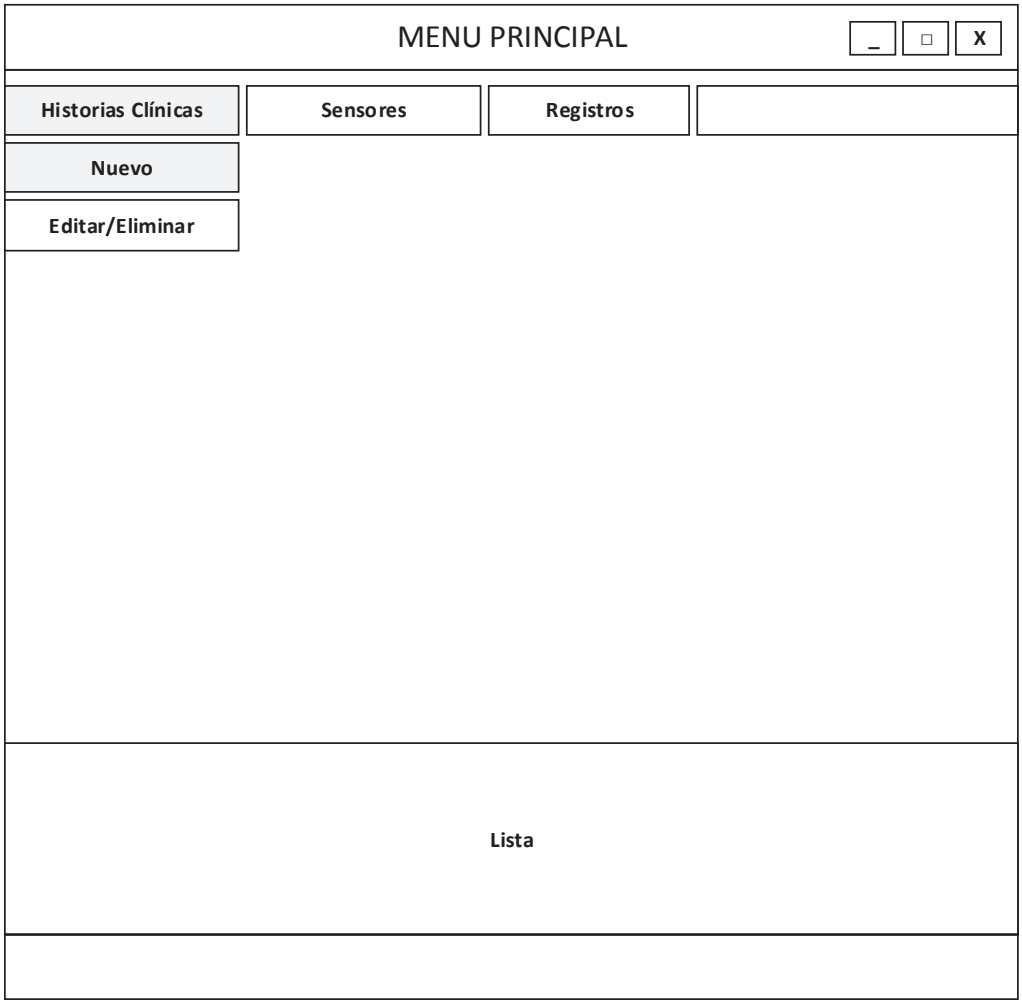


Figura 2.16 Interfaz Principal: Nuevo

En la **Figura 2.17** se observa la interfaz que permitirá el ingreso de una nueva historia clínica. El formulario consta de campos de texto para el ingreso de los datos de una historia clínica, como condición la historia clínica consta de caracteres numéricos y jamás deberá poseer un valor vacío, dos botones de opción para seleccionar el sexo, una lista desplegable para seleccionar uno de los distintos nodos que se encuentra sin utilizar y finalmente posee el botón “Guardar” que permite almacenar la historia clínica en la base de datos.

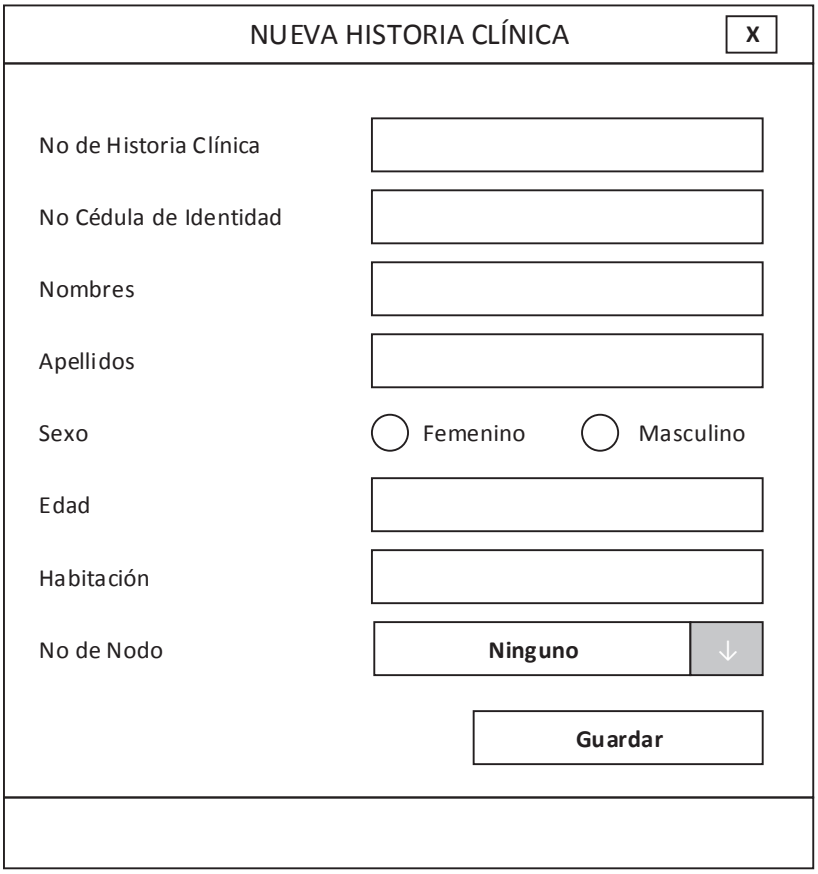


Figura 2.17 Interfaz “Nueva Historia Clínica”

2.5.6 INTERFAZ “EDITAR/ELIMINAR HISTORIA CLÍNICA”

La **Figura 2.18** muestra la interfaz que permite editar o eliminar una historia clínica. Esta interfaz consta de una tabla en donde se muestran todas las historias clínicas almacenadas. Los campos de texto y botones de selección que permiten editar los valores de los campos que conforman la historia clínica de manera fácil.

Pensando en el beneficio del usuario se incorporó una barra de búsqueda de historias clínicas. Para buscar una historia clínica se cuenta con un lista desplegable con el parámetro de búsqueda, un campo de texto en donde se ingresa el valor a buscar y un botón llamado “Buscar” que muestra los resultados de la búsqueda en la tabla. Al seleccionar una historia clínica de la tabla se pueden editar sus valores y guardarlos utilizando el botón “Actualizar”, también se puede eliminar la historia clínica seleccionada presionando el botón “Eliminar”.

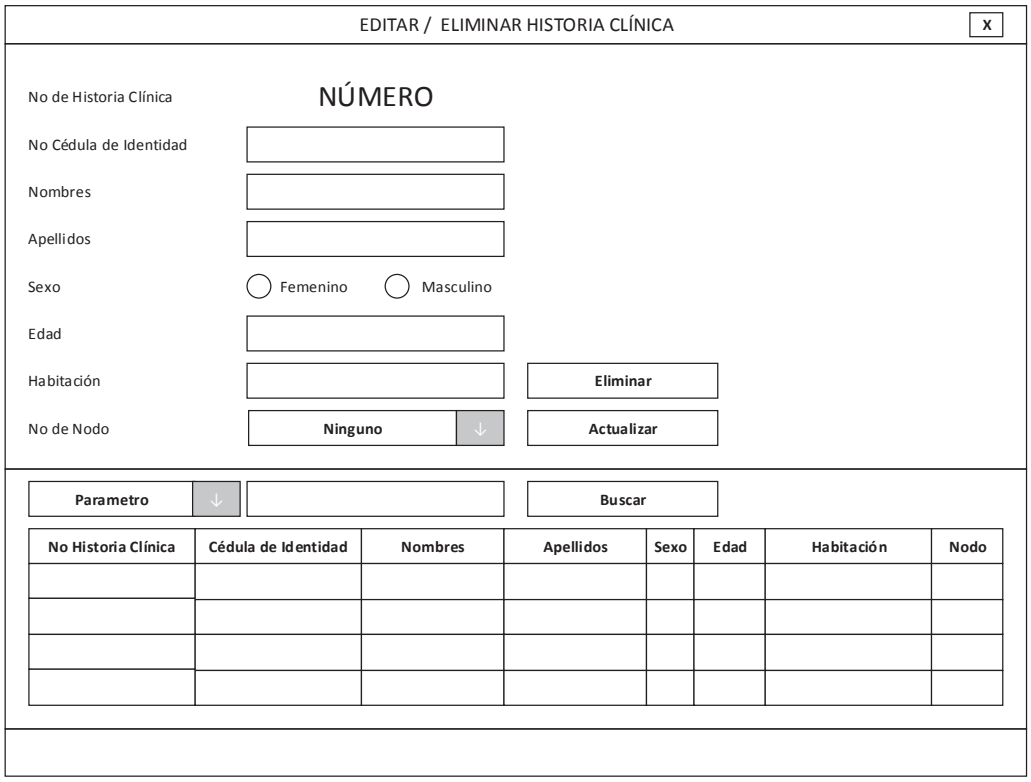


Figura 2.18 Interfaz "Editar/Eliminar historia clínica"

2.5.7 INTERFAZ “NUEVO/ELIMINAR NODO SENSOR”

En la **Figura 2.19** se muestra la apertura para el formulario que permite ingresar un nuevo nodo sensor o eliminar un nodo existente. En la **Figura 2.20** se observa la interfaz que permitirá ingresar un nuevo nodo sensor o eliminar un nodo sensor existente.

Esta interfaz cuenta con una tabla en donde se mostrarán los nodos sensores junto con un mensaje; si el nodo sensor se encuentra asignado se mostrará el mensaje de

“Ocupado” y si no se encuentra asignado a ningún usuario se mostrará junto a la etiqueta de “Disponible”.

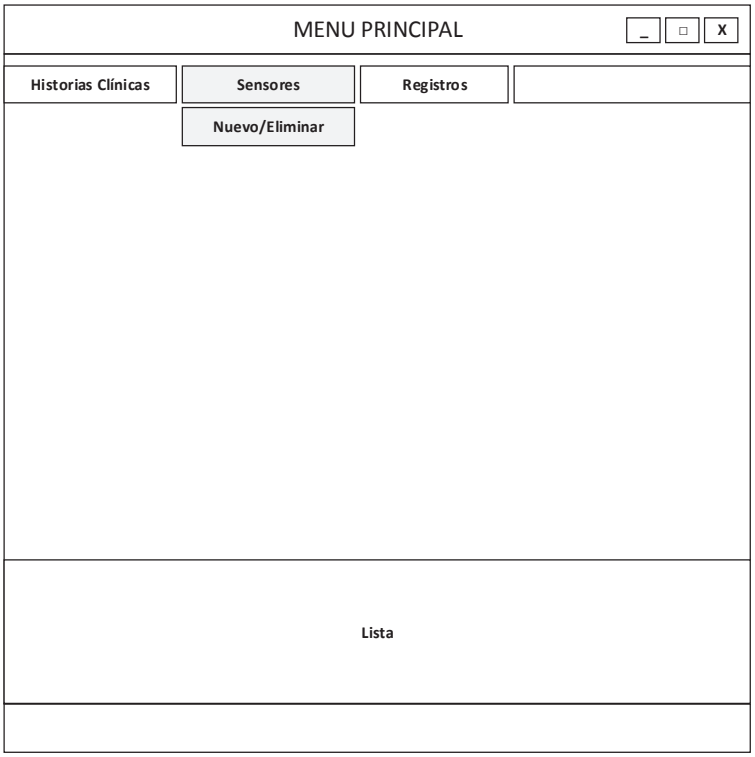


Figura 2.19 Interfaz Principal: "Nuevo/Eliminar nodo sensor"

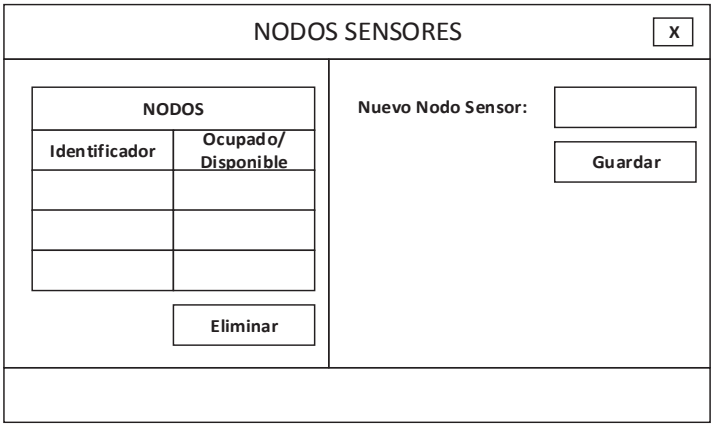


Figura 2.20 Interfaz "Nuevo/Eliminar nodo sensor"

Para eliminar un nodo sensor, en la interfaz “Nuevo/Eliminar nodo sensor” se debe seleccionar de la tabla el nodo a eliminar y a continuación se debe presionar el botón “Eliminar”.

Para ingresar un nuevo sensor se ingresa su identificador en el campo de texto y se presionar el botón “Guardar”

2.5.8 INTERFAZ “MOSTRAR REGISTRO”

En la **Figura 2.21** se muestra la apertura para el formulario que permite mostrar un registro de la frecuencia cardíaca. En la **Figura 2.22** se observa la interfaz que permite mostrar los registros de la frecuencia cardíaca correspondiente a una persona. Esta interfaz consta de una tabla donde se encuentran todas las historias clínicas.

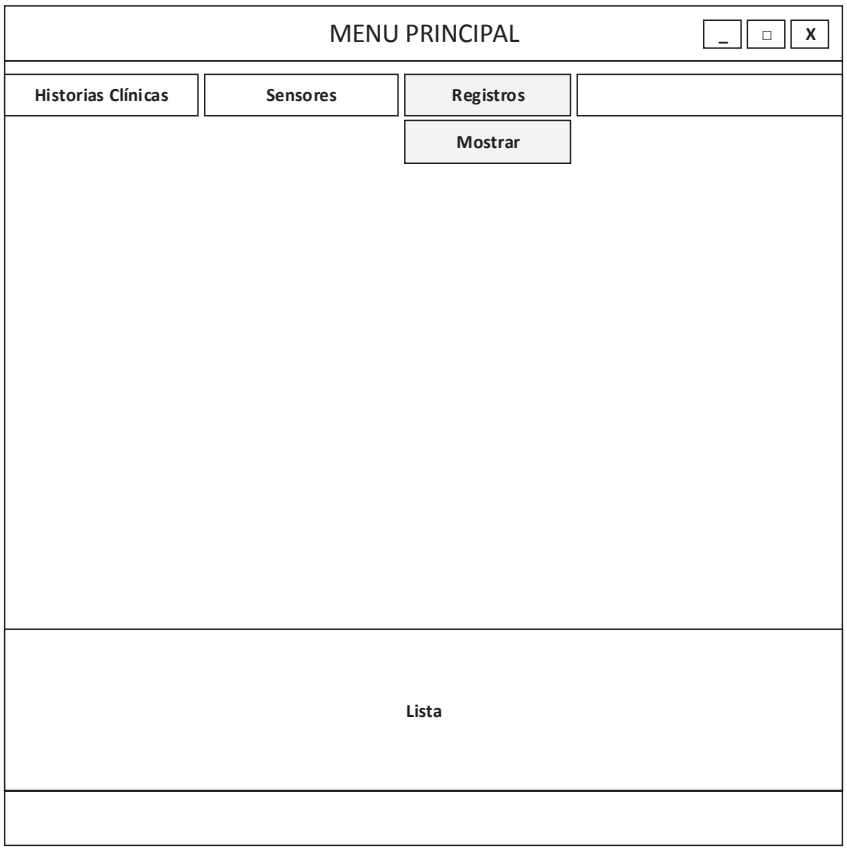


Figura 2.21 Interfaz principal "Mostrar"

A este formulario, pensando en beneficio del usuario, se le agrego una barra de búsqueda para encontrar una historia clínica en particular, se encuentra formada por una lista desplegable que permite seleccionar el parámetro de búsqueda, un campo

de texto en donde se ingresa el valor a buscar y el botón “Buscar” que al presionarlo muestra los resultados en la tabla.

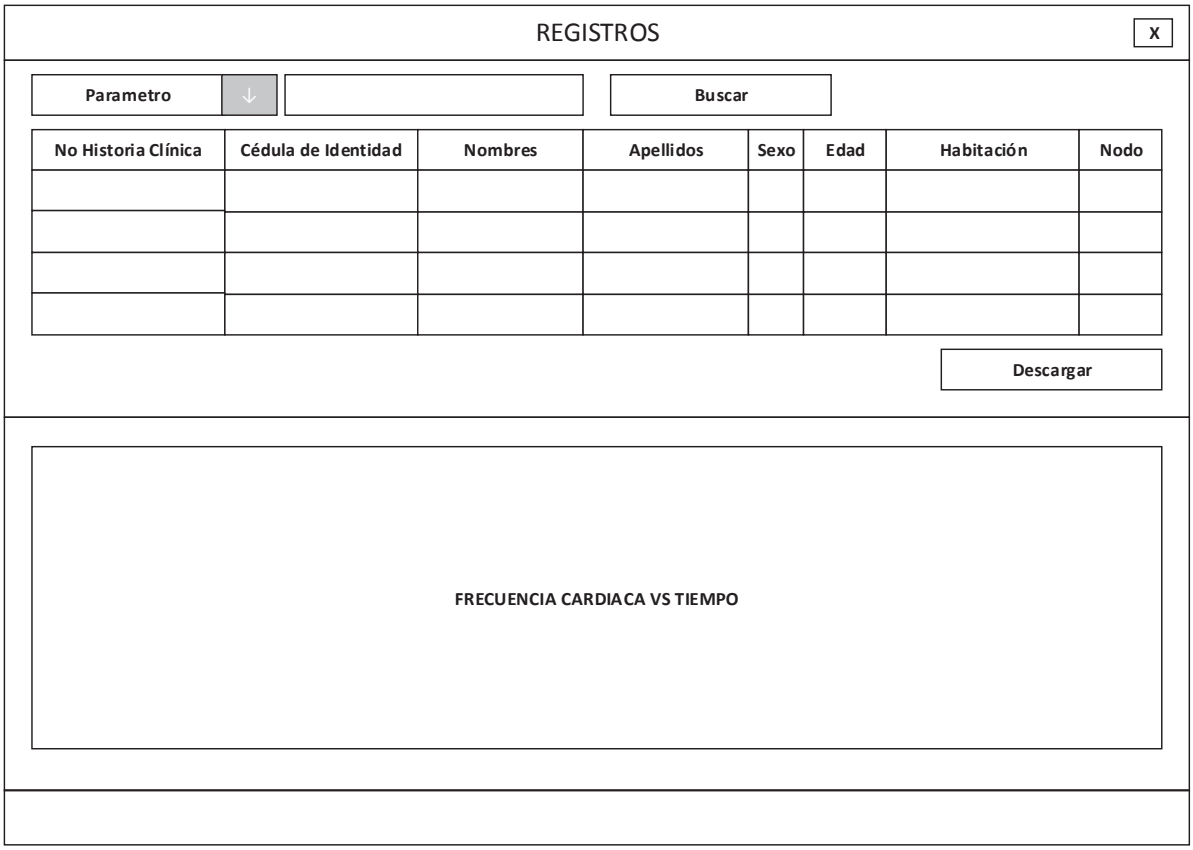


Figura 2.22 Interfaz "Registros"

Para mostrar los registros de la frecuencia cardíaca se dispone de un “panel” donde se muestra en forma gráfica la “Frecuencia Cardíaca vs Tiempo”. Para descargar la gráfica se debe presionar el botón “Descargar”. Durante las entrevistas se observó que los formatos para el almacenamiento de imágenes más utilizados fueron .png y .jpeg, que serán los formatos incluidos en la aplicación.

2.5.9 INTERFAZ “INGRESO”

Para acceder a la aplicación se vio prudente la implementación de un sistema de ingreso. Ante esto se decidió implementar un formulario como el mostrado en **Figura 2.23**. El usuario debe ingresar los parámetros indicados y presionar el botón “Ingresar” para obtener acceso a la aplicación.

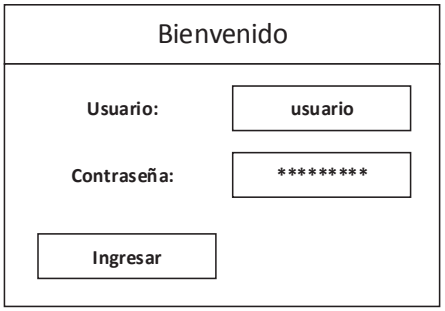


Figura 2.23 Interfaz "Ingreso a la aplicación"

2.6 DISEÑO DEL PAQUETE DEL NODO SENSOR

A medida que se avanzaba en el diseño se vio conveniente que 3 datos deben estar en el paquete enviado por los nodos sensores.

- Comando: Este campo tendrá la extensión mínima de 8 bits y tendrá por función indicar como debe ser tratado el paquete por el Gateway.
  - 0 Envía el valor de la frecuencia cardíaca
  - 1 Utilizado durante la elaboración del prototipo para Tiempo Real.
- Identificador del Nodo: Posee una extensión de 16 bits que es entregada por el sistema operativo, utilizado para el reconocimiento del nodo.
- Dato: Es el valor enviado por el nodo, posee 16 bits y puede ser el valor de la frecuencia cardíaca o el valor obtenido por el sensor utilizado durante la elaboración del prototipo.

2.7 DISEÑO DEL ALGORITMO A INCORPORAR EN EL COMPUTADOR GATEWAY

Para el diseño del algoritmo que se encontrará en el computador Gateway, será implementado en Java teniendo en cuenta las siguientes consideraciones:

- Los datos que provienen del nodo Gateway
- El paquete recibido de los nodos
- El computador Gateway debe indicar en que tabla almacenará los datos en la base de datos.

- El computador Gateway debe envía a la base de datos el valor y registrar la fecha de valor.

En la **Figura 2.24** se muestra el algoritmo a utilizar en el Gateway para el presente trabajo de titulación.

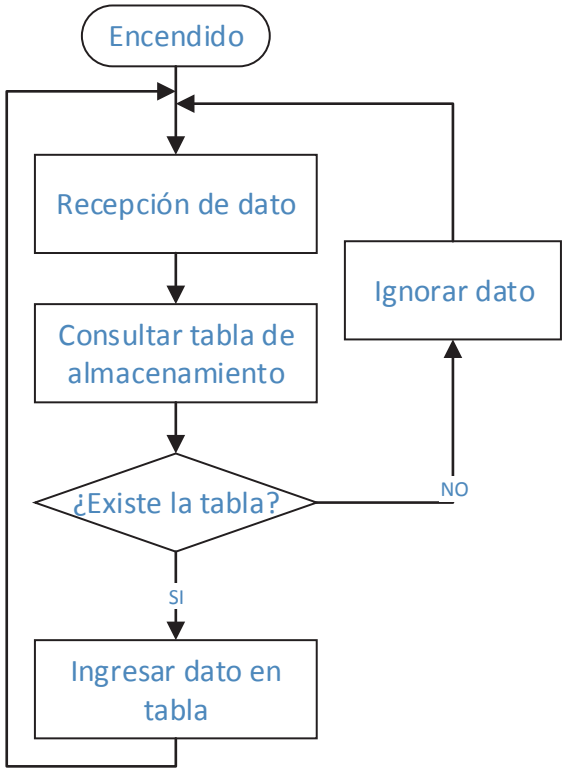


Figura 2.24 Diagrama de flujo Gateway

Una vez iniciado el Gateway este debe esperar la recepción de un dato, cuando este ha llegado debe buscar en la base de datos si la tabla existe para almacenar esta información. En caso de no existir la tabla el dato es ignorado y el Gateway espera recibir el siguiente dato.

2.8 DISEÑO DE LA BASE DE DATOS EN MYSQL

El diseño de la base datos que se presentará fue parte de la implementación de la metodología Scrum explicada en el siguiente capítulo. En base a los requisitos de usuario obtenidos de las entrevistas en la **Figura 2.25** se elabora el siguiente modelo entidad relación, como se observa su diseño es sencillo y fueron necesarias 2 tablas

fijas y una tabla generada dinámicamente para realizar el trabajo de titulación. De las 2 tablas fijas la primera de ellas está encargada de la información de las historias clínicas y la segunda de almacenar los nodos sensores en base a su identificador.

La tabla dinámica será utilizada para el almacenamiento de la información correspondiente al pulso cardiaco, se genera una tabla por cada historia clínica almacenada, esto estará a cargo de la aplicación de usuario.

La base de datos se llamará Medicina y sus tablas fijas se llamarán Historias_Clínicas y la Nodos_Sensores.

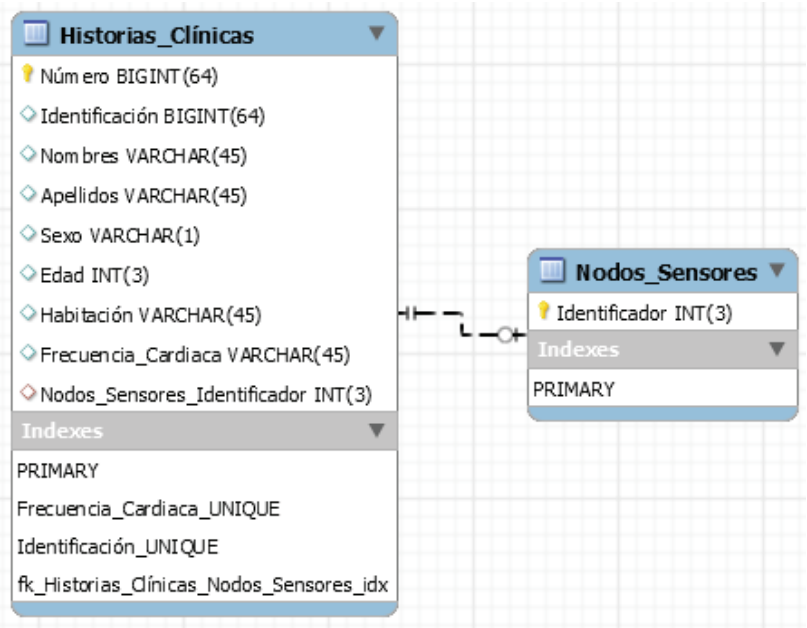


Figura 2.25 Modelo de la base de datos

La tabla Historias_Clínicas contendrá la información correspondiente a cada historia clínica, estos datos fueron obtenidos de las historias de usuario que se encuentran en el siguiente capítulo, la clave principal será el número de Historia Clínica, una clave foránea con la tabla Nodos_Sensores con una relación uno a uno, y un atributo llamado Frecuencia_Cardíaca en donde se almacenará el nombre de la tabla que contendrá los valores de frecuencia cardíaca con su respectivo tiempo, el nombre de esta última tabla estará formado por las siglas FC y el número de historia clínica como en el siguiente ejemplo:

Tabla: Historias_Clínicas

Número de Historia Clínica: **15**

Nombre de la tabla de almacenamiento de la frecuencia cardíaca: **FC15**

La tabla Nodos_Sensores almacenará los nodos sensores mediante su identificador como clave primaria. En la **Figura 2.26** se observa la tabla FC generada por la aplicación. Esta tabla posee como llave primaria la Fecha del Servidor de Base de Datos en el momento en que recibe un valor de frecuencia, y el otro campo es la frecuencia como un valor entero.

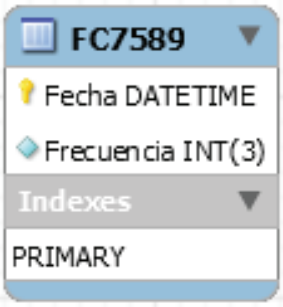


Figura 2.26 Tabla FCXXXX

CAPÍTULO III

IMPLEMENTACIÓN DEL PROTOTIPO PARA EL REGISTRO DE LA FRECUENCIA CARDÍACA

En el presente capítulo se describe la implementación de cada uno de los componentes que conforman el prototipo, primeramente se explicará la instalación de TinyOS y las herramientas necesarias que permitan su utilización.

Seguido se mostrará el proceso de implementación de cada uno de los componentes que conforman el prototipo para finalmente ejecutar pruebas de su funcionamiento.

La **Figura 3.1** muestra el funcionamiento del sistema.

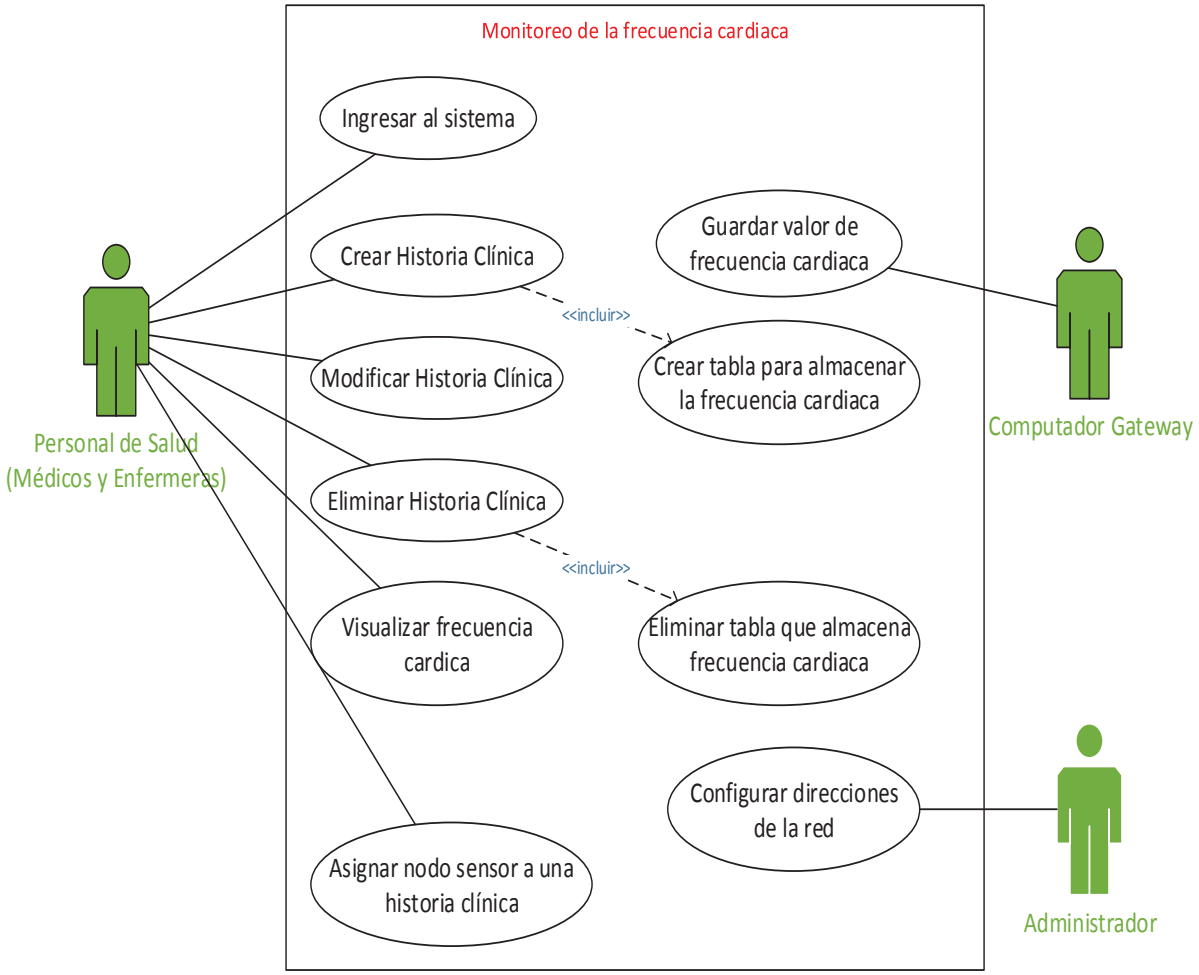


Figura 3.1 Diagrama de casos de uso UML

3.1 INSTALACIÓN DE HERRAMIENTAS

3.1.1 INSTALACIÓN DE TINYOS

El sistema operativo utilizado fue Ubuntu 12.04 de 32 bits. Fue seleccionado por ser la versión más actual hasta la última publicación en la página oficial TinyOS, y se seleccionó la versión de 32 bits por consumir menos recursos del computador.

Para iniciar con la instalación en primer lugar se procede a retirar las versiones antiguas de TinyOS, por ejemplo para retirar la versión 2.1.1 utilizamos el **Segmento de código 3.1**.

```
sudo apt-get remove tinyos-2.1.1
```

Segmento de código 3.1 Remover TinyOS

Seguido se remueve (si existe) el compilador gcc-msp430 con el **Segmento de código 3.2**.

```
sudo apt-get autoremove --purge ms430*
```

Segmento de código 3.2 Remover compilador

A continuación se agregan los links de los paquetes de TinyOS de la Universidad de Stanford en el archivo de la lista de origen de paquetes de Ubuntu que se encuentra en el directorio "/etc/apt/sources.list" con el **Segmento de código 3.3**.

```
sudo gedit /etc/apt/sources.list
```

Segmento de código 3.3 Editar lista de origen

El comando anterior abrirá la ventana del editor de texto, se colocará el **Segmento de código 3.4** y a continuación, se procede a guarda y cierra el editor.

```
#Repositorio de TinyOS

deb http://tinyos.stanford.edu/tinyos/dists/ubuntu lucid main
```

Segmento de código 3.4 Repositorio TinyOS

Después de agregar los links se debe actualizar e instalar la última versión de TinyOS con el **Segmento de código 3.5**:

```
sudo apt-get update

sudo apt-get install tinyos-2.1.2
```

Segmento de código 3.5 Instalar TinyOS

Para continuar con la configuración de TinyOS, es necesario cambiar el propietario de esta carpeta a la nuestra con el **Segmento de código 3.6**.

```
sudo chown nombre_usuario:nombre_usuario –R /opt/tinyos-2.1.2/
```

Segmento de código 3.6 Cambiar propietario

Ahora, usando línea de comandos se debe ubicar el directorio en el que se encuentra la instalación de TinyOS, de no existir ningún cambio sería “/opt/tinyos-2.1.2” y se debe crear o modificar el archivo tinyos.sh con el **Segmento de código 3.7**:

```
#! /usr/bin/env bash
# Configuración de las variables de entorno de tinyos
echo "Configuración para TinyOS-2.1.2"
export TOSROOT=
export TOSDIR=
export MAKERULES=
TOSROOT="/opt/tinyos-2.1.2"
TOSDIR="$TOSROOT/tos"
CLASSPATH=$CLASPATH:$TOSROOT/support/sdk/java
MAKERULES="$TOSROOT/support/make/Makerules"
export TOSROOT
export TOSDIR
export CLASSPATH
export MAKERULES
```

Segmento de código 3.7 TinyOS.sh

Ahora se debe definir las variables en el archivo .bashrc, el archivo se puede editar con el **Segmento de código 3.8**:

```
sudo gedit ~/.bashrc
```

Segmento de código 3.8 Editar bashrc

Se procede a colocar el **Segmento de código 3.9**:

```
#Iniciar encaminamiento del ambiente TinyOS
export TOSROOT=/opt/tinyos-2.1.2
```

```
export TOSDIR=$TOSROOT/tos
export
CLASSPATH=$TOSROOT/support/sdk/java/tinyos.jar:.$CLASSPATH
export MAKERULES=$TOSROOT/support/sdk/make/Makerules
export PATH=/opt/msp430/bin:$PATH
source /opt/tinyos-2.1.2/tinyos.sh
#Fin del encaminamiento de TinyOS
```

Segmento de código 3.9 Código bashrc

Por último se aplican los cambios ejecutando el **Segmento de código 3.10**:

```
source ~/.bashrc
```

Segmento de código 3.10 Aplicar cambios bashrc

3.1.2 INSTALACIÓN DE JAVA TOOLS

Se procede a instalar java tolos que viene con TinyOS. Ejecutar el **Segmento de código 3.11**.

```
cd $TOSROOT/support/sdk/java
sudo tos-install-jni
make
make install
```

Segmento de código 3.11 Java Tools

3.1.3 INSTALACIÓN DE GCC MSP430 TOOLCHAIN

Si la instalación de TinyOS es por primera vez, MSP430 TOOLCHAIN se encontrará instalada, para verificar que es así se puede utilizar el **Segmento de código 3.12**:

```
msp430 -gcc --version
```

Segmento de código 3.12 Verificar instalación de msp430

Si no se encuentra instalada, o si la versión es inferior a 4.6.3 se debe ejecutar el **Segmento de código 3.13**:

```
sudo apt-get autoremove --purge msp430*
```

Segmento de código 3.13 Remover msp430

Ahora es necesario agregar las llaves necesarias para Ubuntu utilizando el **Segmento de código 3.14**:

```
gpg --keyserver keyserver.ubuntu.com --recv-keys 34EC655A

gpg -a --export 34EC655A | sudo apt-key add –
```

Segmento de código 3.14 Llaves

Siguiente, agregar el **Segmento de código 3.15** de orígenes de paquetes a la lista de origen de paquetes de Ubuntu:

```
#TinyOS: Repositorios del compilador MSP430 GCC

deb http://tinyprod.net/repos/debian squeeze main

deb http://tinyprod.net/repos/debian msp430-46 main
```

Segmento de código 3.15 Repositorios msp430

Ejecutar el **Segmento de código 3.16** para la instalación de MSP430 Toolchain, la instalación se realizará en el directorio ”/opt/msp430-46”. Esta instalación no se agregará a las rutas de entorno por defecto por lo que será necesario ejecutar el **Segmento de código 3.17** cada ocasión que el equipo es encendido.

```
sudo pat-get update

sudo apt-get install msp430-46 nesc tinyos-tools
```

Segmento de código 3.16 Instalación msp430

Los compiladores mspgcc por defecto y experimentales son instalados juntos en el mismo sistema. La selección del *toolchain* es realizada por las PATH como la del **Segmento de código 3.17** de las variables de entorno. Esto permite comparar

fácilmente entre *toolchains* experimentales y comunes. También permite evitar colisiones con herramientas de la distribución principal.

```
PATH=/opt/msp430-46/bin:$PATH
```

Segmento de código 3.17 PATH msp430

La línea anterior puede ser agregada al archivo .bashrc para evitar ejecutar el comando cada ocasión que el sistema es encendido.

3.1.3.1 Prueba del compilador

Para comprobar que la instalación del compilador ha sido correcta, se puede utilizar las aplicaciones de muestra que se encuentran en el directorio "$TOSROOT/apps". Como podemos ver en la **Figura 3.2** encontramos listadas las aplicaciones preinstaladas.

```
byron@ubuntu:/opt/tinyos-2.1.2/support/sdk/java$ cd $TOSROOT/apps
byron@ubuntu:/opt/tinyos-2.1.2/apps$ ls
AntiTheft        Makefile               Oscilloscope      Sense       UDPEcho
BaseStation      MultihopOscilloscope   Powerup           TCPEcho
BaseStation15.4  MultihopOscilloscopeLqi PppRouter        tests
Blink            MViz                   RadioCountToLeds  tosthreads
CoapBlip         Null                   RadioSenseToLeds  tutorials
byron@ubuntu:/opt/tinyos-2.1.2/apps$
```

Figura 3.2 Aplicaciones de TinyOS

En la carpeta con la aplicación "*Blink*" vamos a encontrar 3 archivos como se muestra en la **Figura 3.3**, antes de poder cargar la aplicación en un nodo esta debe ser previamente compilada.

```
byron@ubuntu:/opt/tinyos-2.1.2/apps/Blink$ ls
BlinkAppC.nc  BlinkC.nc  Makefile  README.txt
```

Figura 3.3 Aplicación *Blink*

Se ejecuta el **Segmento de código 3.18** para compilar el código de acuerdo al nodo que disponemos, para este trabajo de titulación el nodo es "iris":

```
make iris
```

Segmento de código 3.18 Compilar aplicación

Como se observa en la **Figura 3.4** la compilación se realizó sin problemas.

Al realizar un nuevo listado se puede observar que se ha creado una nueva carpeta llama "*build*", si se abre esta carpeta se encontrarán carpetas para cada una de las plataformas compiladas, en este caso la plataforma "iris" como se puede observar en **Figura 3.5**.

```
byron@ubuntu:/opt/tinyos-2.1.2/apps/Blink$ make iris
mkdir -p build/iris
    compiling BlinkAppC to a iris binary
ncc -o build/iris/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all -t
arget=iris -fnesc-cfile=build/iris/app.c -board=micasb -DDEFINED_TOS_AM_GROUP=0x
22 --param max-inline-insns-single=100000 -DIDENT_APPNAME=\"BlinkAppC\" -DIDENT_
USERNAME=\"byron\" -DIDENT_HOSTNAME=\"ubuntu\" -DIDENT_USERHASH=0x832c629fL -DID
ENT_TIMESTAMP=0x57da38adL -DIDENT_UIDHASH=0x681acaa6L -fnesc-dump=wiring -fnesc-
dump='interfaces(!abstract())' -fnesc-dump='referenced(interfacedefs, components
)' -fnesc-dumpfile=build/iris/wiring-check.xml BlinkAppC.nc -lm
    compiled BlinkAppC to build/iris/main.exe
          2272 bytes in ROM
            51 bytes in RAM
avr-objcopy --output-target=srec build/iris/main.exe build/iris/main.srec
avr-objcopy --output-target=ihex build/iris/main.exe build/iris/main.ihex
    writing TOS image
```

Figura 3.4 Compilación en Iris

```
byron@ubuntu:/opt/tinyos-2.1.2/apps/Blink/build$ ls
iris
byron@ubuntu:/opt/tinyos-2.1.2/apps/Blink/build$ cd iris
byron@ubuntu:/opt/tinyos-2.1.2/apps/Blink/build/iris$ ls
app.c            main.exe   main.srec      wiring-check.xml
ident_flags.txt  main.ihex  tos_image.xml
```

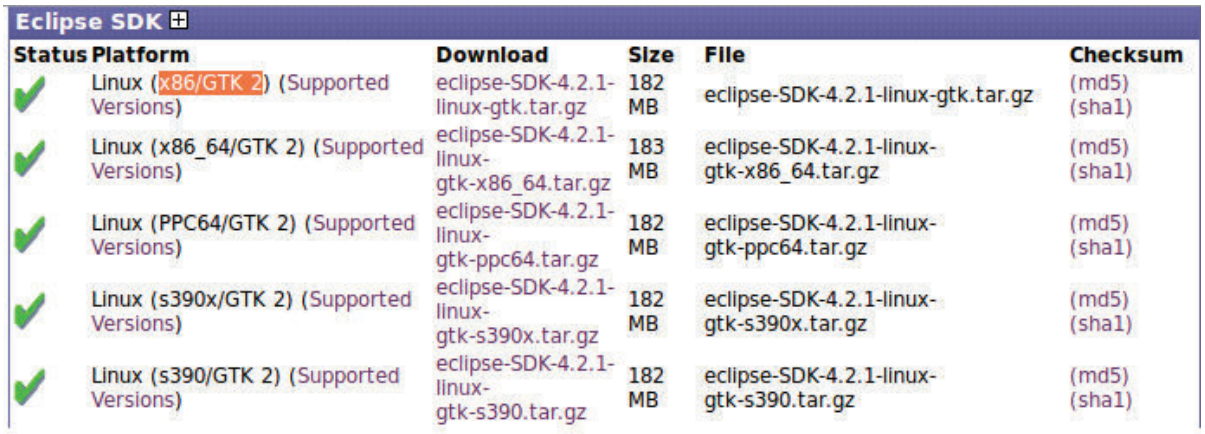
Figura 3.5 Código compilado de la aplicación *Blink*

3.1.4 INTALACIÓN DE ECLIPSE

Para la instalación de Eclipse es necesario descargar Eclipse SDK en la página oficial, su último lanzamiento se encuentra en el siguiente enlace:

http://download.eclipse.org/eclipse/downloads/

Como se observa en la **Figura 3.6**, fue seleccionado Eclipse 4.2.1 que es compatible con el Linux.



Eclipse SDK

Status	Platform	Download	Size	File	Checksum
✔	Linux (x86/GTK 2) (Supported Versions)	eclipse-SDK-4.2.1-linux-gtk.tar.gz	182 MB	eclipse-SDK-4.2.1-linux-gtk.tar.gz	(md5) (sha1)
✔	Linux (x86_64/GTK 2) (Supported Versions)	eclipse-SDK-4.2.1-linux-gtk-x86_64.tar.gz	183 MB	eclipse-SDK-4.2.1-linux-gtk-x86_64.tar.gz	(md5) (sha1)
✔	Linux (PPC64/GTK 2) (Supported Versions)	eclipse-SDK-4.2.1-linux-gtk-ppc64.tar.gz	182 MB	eclipse-SDK-4.2.1-linux-gtk-ppc64.tar.gz	(md5) (sha1)
✔	Linux (s390x/GTK 2) (Supported Versions)	eclipse-SDK-4.2.1-linux-gtk-s390x.tar.gz	182 MB	eclipse-SDK-4.2.1-linux-gtk-s390x.tar.gz	(md5) (sha1)
✔	Linux (s390/GTK 2) (Supported Versions)	eclipse-SDK-4.2.1-linux-gtk-s390.tar.gz	182 MB	eclipse-SDK-4.2.1-linux-gtk-s390.tar.gz	(md5) (sha1)

Figura 3.6 Descarga del SDK

Se procede a extraer el SDK. Se debe dar permisos al directorio "/opt" para poder copiar archivos con el **Segmento de código 3.19**:

```
sudo chown nombre_usuario /opt
```

Segmento de código 3.19 Permisos opt

Antes de obtener los permisos el sistema operativo sobre el que se trabaja, en este caso Ubuntu, pedirá que al usuario que ingrese la contraseña del sistema.

La carpeta de Eclipse que contiene el SDK debe copiarse en el directorio "/opt". Se puede crear un archivo ejecutable ingresando a la carpeta de Eclipse, posteriormente se crea un acceso directo del archivo para moverlo al Escritorio. Una vez creado se ejecuta el acceso directo para iniciar Eclipse.

3.1.5 INSTALACIÓN DE YETI 2

En Eclipse es necesario ir al menú "*Help>Install new software*" para agregar un nuevo repositorio, se debe colocar un nombre y de dirección "http://tos-ide.ethz.ch/*update*/site.xml" como se muestra en la **Figura 3.7**.

Posteriormente se seleccionan los *plugin*[55] a instalar. Se seleccionan todos excepto "Yeti 2-TinyOS 2.x in Cygwin" (debido a que este *plugin* es para Windows) como se muestra en la **Figura 3.8**. Se presiona el botón "*Next*" para revisar los ítems que serán instalados, se presiona el botón "*Next*" nuevamente para revisar las licencias y en caso de aceptar los términos de la licencia, se finaliza la instalación pulsando el botón "*Finish*" en ese orden.

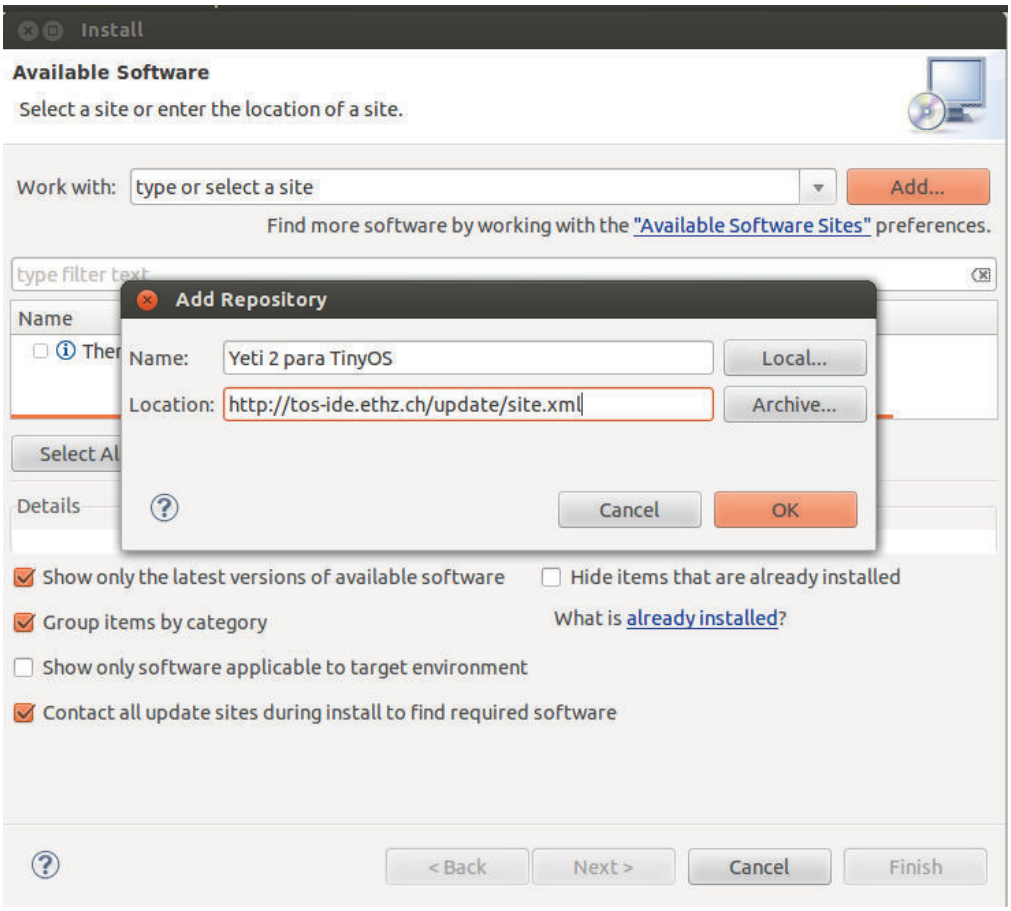


Figura 3.7 Agregar repositorio

[55] ***Plugin***: Aplicación que aporta una funcionalidad específica a una aplicación principal.

Se reinicia Eclipse. En el menú “*Window>Open Perspective>Other*…” se verifica que tengamos la perspectiva de TinyOS ya instalada como se muestra en la **Figura 3.9**.

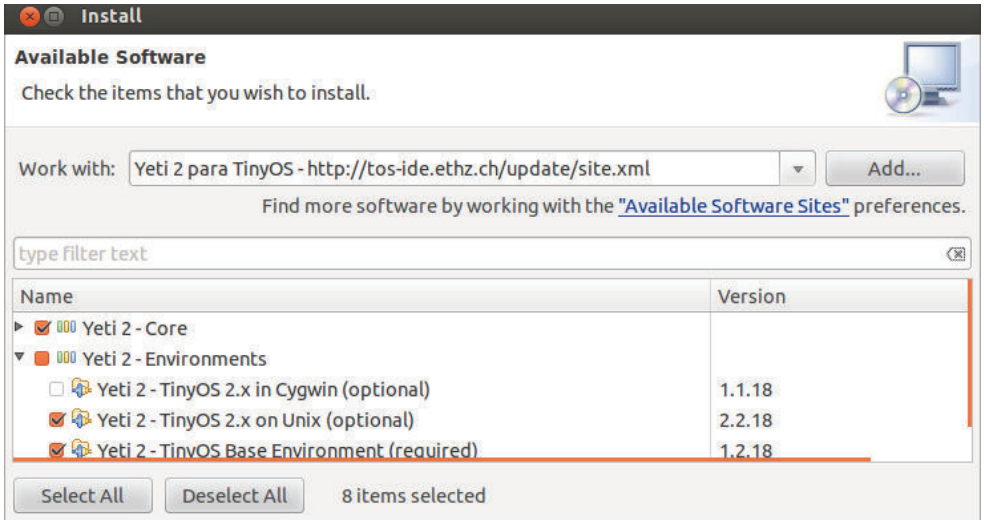


Figura 3.8 Selección de *plugin*

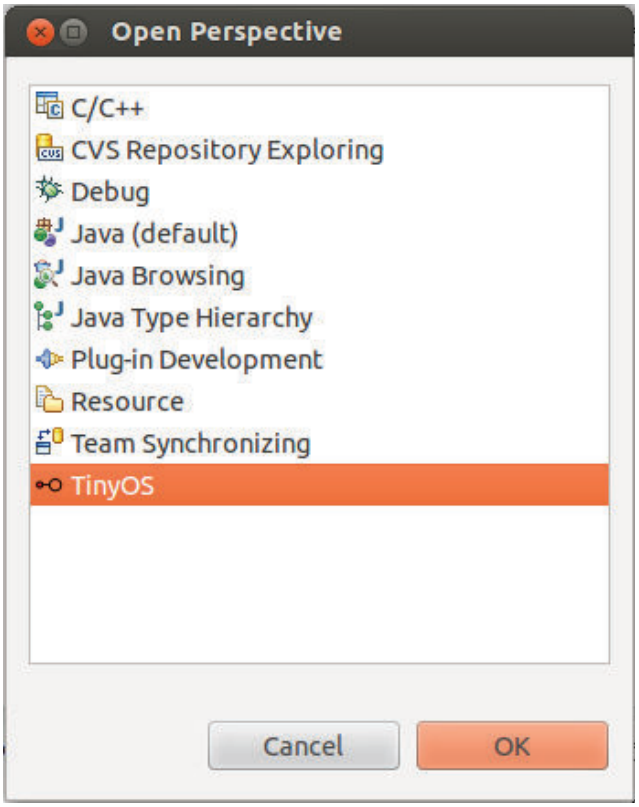


Figura 3.9 Abrir perspectiva

3.1.6 INSTALACIÓN DE JDBC

El driver JDBC que permite la comunicación entre la base de datos y la aplicación de Java es descargable de http://www.mysql.com/products/connector/j/ totalmente gratis. Se extrae el driver JBDC e importa al proyecto como archivo de sistema, en la **Figura 3.10** se observa cómo se importa un archivo un recurso.

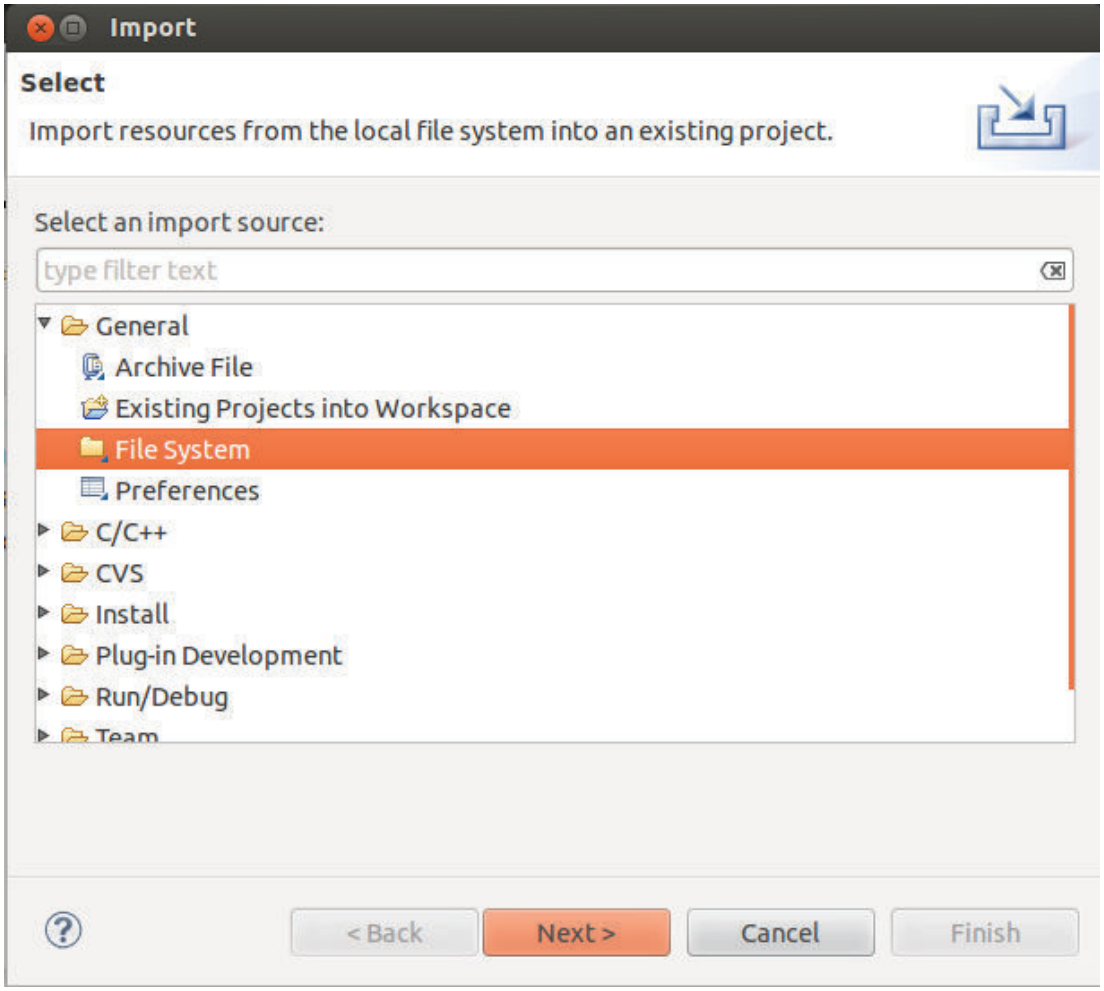


Figura 3.10 Importación de un "Archivo de Sistema"

Una vez seleccionado el directorio se debe marcar el conector como se indica en la **Figura 3.11** y se finaliza el proceso. Finalizada la importación se agrega la librería en “Proyecto>Propiedades”, después se selecciona “Java Build Path”. Una vez aquí se escoge la opción “Agregar JAR…” y se agrega el conector de mysql como se muestra en la **Figura 3.12**.

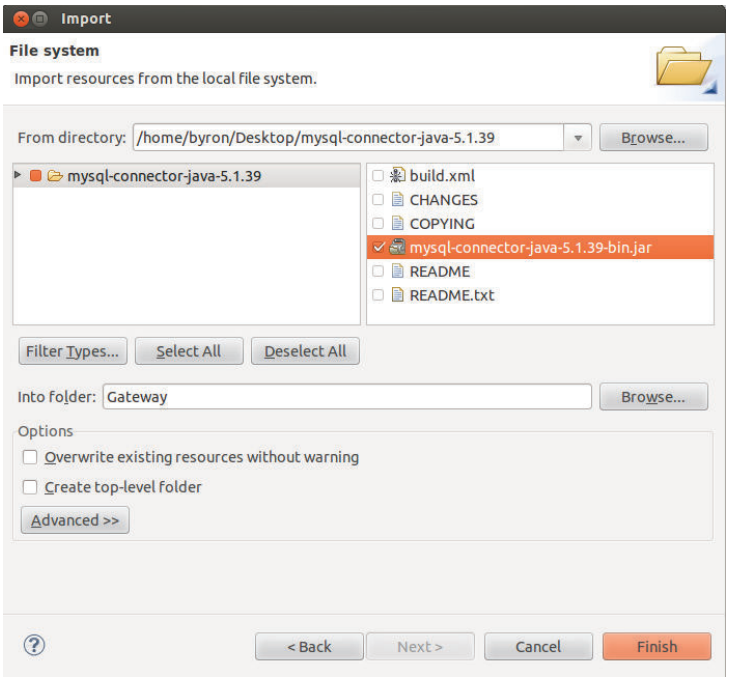


Figura 3.11 Importación de mysql-connector-java

3.1.7 INSTALACIÓN DE NETBEANS

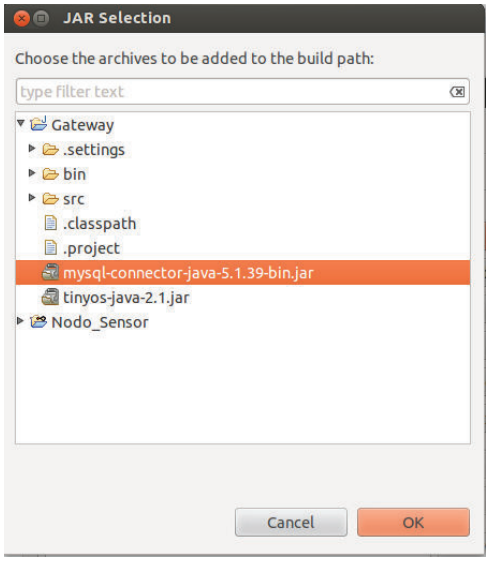


Figura 3.12 Agregar JAR

Para la aplicación de usuario se optó por NetBeans debido a su facilidad para trabajar con herramientas visuales y se seleccionó el sistema operativo Windows 7 Ultímate Pro de 32 bits, la instalación es sumamente fácil.

En la página oficial de NetBeans se descarga el instalador:

```
https://netbeans.org/downloads/
```

3.1.7.1 Instalación de java development kit

Es necesario tener instalado el JDK, de no ser así la instalación también es muy sencilla. El instalador se debe puede descargar de la página oficial de Oracle que se muestra a continuación:

```
http://www.oracle.com/
```

3.1.8 INSTALACIÓN DE MYSQL

Para la instalación de MySQL server se ejecuta el **Segmento de código 3.20** en una ventana de terminal:

```
sudo apt-get install mysql-server
```

Segmento de código 3.20 Instalar MySQL

3.2 IMPLEMENTACIÓN DEL NODO SENSOR

En Eclipse se abre la perspectiva TinyOS. Se crea un nuevo proyecto en "File>New>Project…" de tipo TinyOS como se muestra en la **Figura 3.13**.

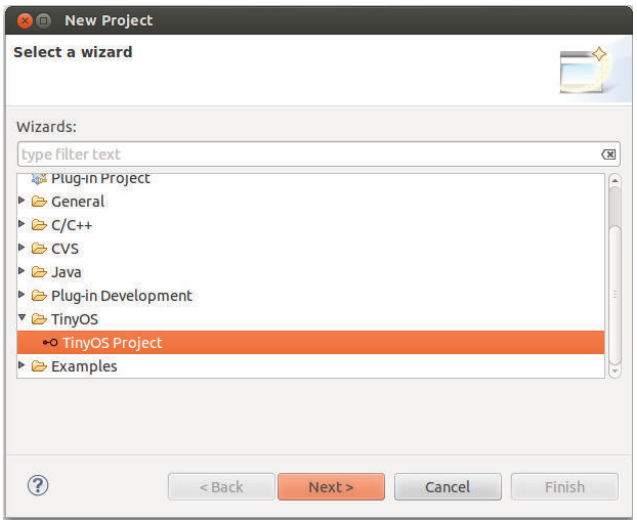


Figura 3.13 Proyectos en Eclipse

El nombre del proyecto para el presente trabajo es Nodo_Sensor y el objetivo es iris como se muestra en la **Figura 3.14**. En la carpeta “src” del nuevo proyecto se necesita crear 4 archivos que se detallan a continuación, el código de cada uno de estos archivos se adjunta en el Anexo C.

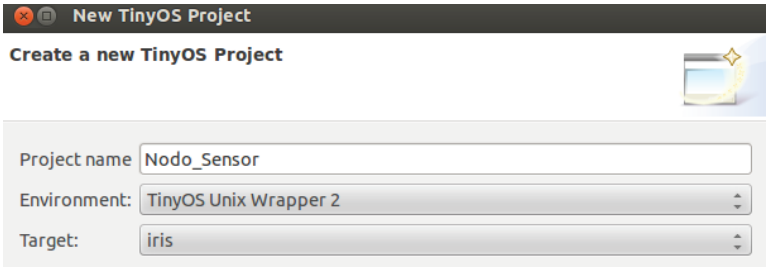


Figura 3.14 Proyecto TinyOS

3.2.1 ARCHIVO DE CONFIGURACIÓN

El archivo se crea en siguiendo el directorio *File>New>Configuration* en el proyecto seleccionado. Este archivo es llamado NodoSensorAppC.nc. Este archivo indica como NodoSensorC.nc está conectado a los servicios de TinyOS, especifica las conexiones entre las interfaces y los componentes. En la **Figura 3.15** se puede observar el diagrama de conexiones de la aplicación, el contenido en los cuadrados entrecortados indica los módulos genéricos.

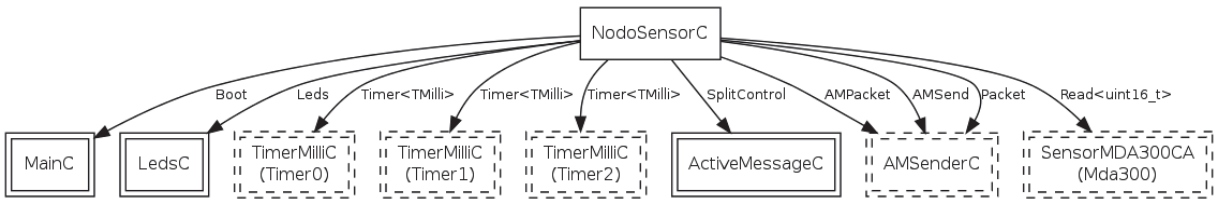
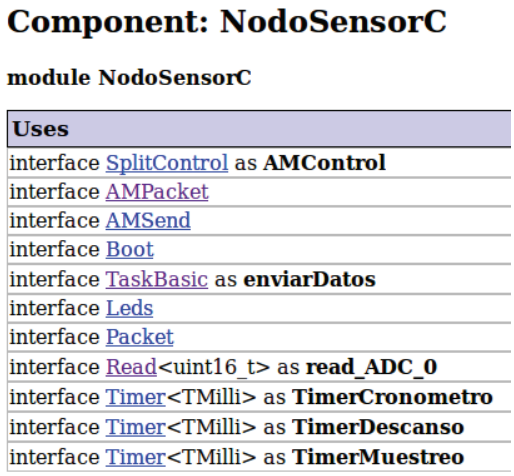


Figura 3.15 Componente NodoSensorAppC

3.2.2 ARCHIVO DE MODULO

El archivo se crea siguiendo el directorio *File>New>Module* con el nombre de NodoSensorC.nc, en este archivo se encuentra la lógica de las interfaces como se observa en la **Figura 3.16**.



Component: NodoSensorC

module NodoSensorC

Uses
interface SplitControl as **AMControl**
interface AMPacket
interface AMSend
interface Boot
interface TaskBasic as **enviarDatos**
interface Leds
interface Packet
interface Read<uint16_t> as **read_ADC_0**
interface Timer<TMilli> as **TimerCronometro**
interface Timer<TMilli> as **TimerDescanso**
interface Timer<TMilli> as **TimerMuestreo**

Figura 3.16 Componente NodoSensorC

Cada interfaz conectada tiene su uso particular en el programa:

- *SplitControl* es usada para cambiar entre el estado encendido y apagado de los componentes.
- AMPacket, AMSend y Packet se utilizan durante las comunicaciones de radio.
- *Boot* utilizada durante el encendido.
- *TaskBasic* utilizada durante la llamada a métodos.
- Leds utilizada durante el control del LED.
- *Read* utilizada para la lectura de datos del canal analógico 0 de MDA300CA.
- *Timer*<Tmilli> utilizada para llamar a cada uno de los temporizadores que se propusieron en la **Figura 2.9** durante el diseño.

Las variables y constantes utilizadas en el presente trabajo de titulación se muestran en el **Segmento de código 3.21**

Durante la secuencia de inicio se procede a inicializar las variables anteriormente mostradas. Una vez encendido el nodo sensor, este comienza a recibir las señales del sensor periódicamente, en cada medición trata de identificar si se ha generado un nuevo pulso.

Al finalizar el temporizador de muestreo con una duración de 15 segundos se procede a comparar el número de pulsos obtenidos, con los que se obtuvieron

anteriormente, si la variación es menor al 10% por lo menos 3 veces se asume que el valor es correcto y se procede con la construcción y envió del paquete al nodo Gateway como se aprecia en el **Segmento de código 3.22**.

```
implementation{
    //Valor recibido desde el sensor cardiaco
    uint16_t valor=0;
    //Variables utilizadas para la conversión del valor recibido por
    //el sensor cardiaco a hexadecimal
    uint16_t ByteMayor,ByteMenor;
    //Variables para contar el número de pulsos
    uint8_t pulsos,pulsosReferencia;
    //Variables para limitar la variación del número de pulsos
    uint16_t pulsosSuperior,pulsosInferior;
    //Resultado: Frecuencia cardíaca
    uint16_t total;
    //Indica si se ha producido un nuevo pulso
    bool pico;
    //Indica si hay un proceso en ejecución
    bool ocupado;
    //Número de veces que la lectura de pulso se repite
    uint8_t veces;
    //Mínimo de veces que la lectura de pulso debe repetirse para
    //ser aceptada como real
    uint8_t minimoVeces;
    //Variable que contiene el mensaje a ser enviado
    message_t paquete;
    …
}
```

Segmento de código 3.21 Constantes y variables de NodoSensorC.nc

```
...
//Método que envía la información al Gateway
task void enviarDatos(){
//Verificar que no este otro proceso en ejecución
if(ocupado == FALSE){
//Preparación del mensaje a enviar
BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)(call
Packet.getPayload (&paquete,sizeof(BlinkToRadioMsg)));
if (btrpkt == NULL) {
return;
}
//Carga del paquete con el identificador y el valor de pulso
btrpkt -> comando=0;
btrpkt -> identificadorNodo = TOS_NODE_ID;
btrpkt -> dato = total;
//Envío del valor de frecuencia al Gateway
if (call AMSend.send(AM_BROADCAST_ADDR,&paquete, sizeof
(BlinkToRadioMsg)) == SUCCESS){
//Indicación que se ejecutara el proceso de envió de datos
ocupado = TRUE;
//Se detienen los temporizadores de muestreo
call TimerCronometro.stop();
call TimerMuestreo.stop();
//Arranca el temporizador de descanso
call TimerDescanso.startPeriodic(DESCANSO);
}
}
}
```

Segmento de código 3.22 Envió de frecuencia al Gateway - NodoSensorC.nc

3.2.3 ARCHIVO DE COMPILACIÓN

Este archivo indica a TinyOS como compilar incluyendo la estructura del paquete y los valores de los temporizadores utilizados, en el **Segmento de código 3.23** se observa el archivo de compilación del nodo sensor utilizado en el presente trabajo de titulación.

```
enum{
//Indicador del mensaje activo, se asocia con la clase .java
AM_NODOSENSORMENSAJE=6,
//La frecuencia de muestreo en milisegundos
FRECUENCIA_DE_MUESTREO=100,
//El tiempo de actualización
ACTUALIZAR=6000,
//El tiempo para el conteo de pulsos
TIEMPO_DE_MUESTREO=15000
};
//Estructura del mensaje
typedef nx_struct NodoSensorMensaje{
//Código prueba
//valor del comando
//0 para enviar pulso
nx_uint8_t comando;
//Identificador del nodo
nx_uint16_t identificadorNodo;
//Dato a enviar
nx_uint16_t dato;
}
BlinkToRadioMsg;
#endif /* NODO_SENSOR_H */
```

Segmento de código 3.23 NodoSensor.h

3.2.4 ARCHIVO MAKEFILE

El archivo *Makefile* es una forma simple de organizar el código de compilación, el código para el presente trabajo de titulación se muestra en el **Segmento de código 3.24**.

```
COMPONENT=NodoSensorAppC
BUILD_EXTRA_DEPS += NodoSensor.class
CLEAN_EXTRA = *.class NodoSensorMensaje.java
CFLAGS += -I$(TOSDIR)/sensorboards/mda300ca/ -I
NodoSensor.class: $(wildcard *.java) NodoSensorMensaje.java
javac *.java
NodoSensorMensaje.java:
mig java -target=null $(CFLAGS) -java-
classname=NodoSensorMensaje NodoSensor.h NodoSensorMensaje -o $@
include $(MAKERULES)
```

Segmento de código 3.24 *Makefile*

La primera línea del **Segmento de código 3.24** informa a TinyOS cuál es el componente principal de la aplicación que para este trabajo de titulación se trata de NodoSensorAppC, las siguientes líneas le indican las dependencias que serán construidas utilizando el compilador MIG.

La línea CFLAGS indica en que directorio se encuentran los drivers para utilizar las interfaces y componentes del programa. La última línea del archivo se refiere a la carga en el sistema de compilación de TinyOS, que tiene todas las reglas para construir e instalar en diferentes plataformas.

3.2.5 INSTALACIÓN DEL CÓDIGO EN EL NODO SENSOR

Para instalar un programa creado en TinyOS en un nodo sensor, se conecta el nodo a la placa MIB520CB para posterior conectar esta placa a la computadora por medio un cable USB como se observa en la **Figura 3.17**.

Antes de cargar el código al nodo sensor es necesario compilarlo previamente, para esto hay abrir una ventana de terminal y dirigirse a la carpeta donde se encuentra el código recién creado. Una vez ahí se ejecuta el **Segmento de código 3.18**.

Figura 3.17 Nodo sensor con MIB520CB

Si la compilación fue exitosa se ejecutará el siguiente comando para realizar la instalación del código en el nodo sensor, se debe ejecutar el **Segmento de código 3.25** considerando que el identificador debe ser distinto de cero para los nodos sensores:

```
$ make iris install,{identificador} mib520, /dev/ttyUSB(n)
```

Segmento de código 3.25 Instalación del código en Nodo Sensor

Ejemplo:

```
$ make iris install,10 mib520, /dev/ttyUSB0;
```

Una vez instalado el programa en el nodo sensor, se procede a conectar la placa MDA300CA, a esta placa se conecta el sensor cardiaco en los terminales Vcc, GND y A0. De acuerdo a la literatura ese sería todo el procedimiento para empezar a recibir las señales que provienen del sensor de pulso, pero en la práctica no es así,

utilizando los LED del nodo sensor se pudo comprobar que la lectura no se realizaba correctamente al conectar el nodo sensor con la placa MDA300CA, el error se lo describe a continuación y como fue solucionado.

Durante la implementación no fue posible encender y apagar un LED de manera automática ante cambios de voltaje en la entrada analógica cuando la placa MDA300CA era conectada al nodo, esto indicaba que la lectura no se estaba realizando. Con diversos experimentos y revisiones del driver de la placa MDA300CA se llegó a la conclusión de que el evento readDone(), encargado de realizar la convertir un valor analógico en digital nunca era llamado, el canal ADC_0 que es donde se recibe la señal del sensor de pulso no podía realizar la lectura. La señal ingresaba a ADC_0 pero el evento readDone() no se ejecutaba.

Mediante una búsqueda exhaustiva se pudo comprobar que era un error común y se probaron varias soluciones sin éxito. Finalmente una solución fue determinante y consistió en la adición de 2 resistencias de 10 kOhms, la primera entre los pines Vcc y DATA, mientras que la segunda entre los pines Vcc y CLK [16], el error se debía a que el driver para la placa MDA300CA no fue completado para la versión TinyOS 2.x.

En la **Figura 3.18** se puede apreciar el diagrama de la placa MDA300CA conectado a las 2 resistencias de 10 kOhm y al sensor cardíaco.

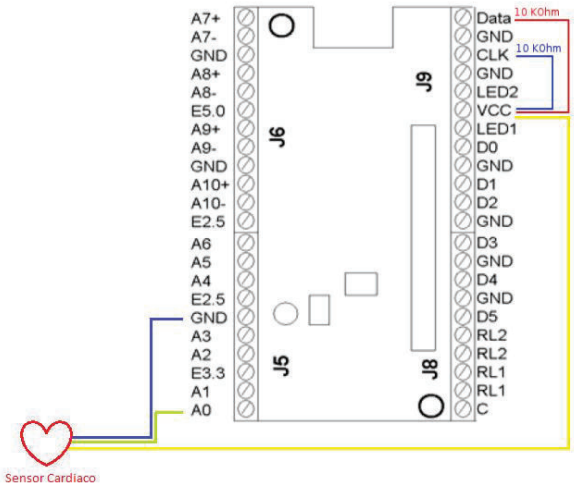


Figura 3.18 Diagrama de conexiones MDA300CA

En la **Figura 3.19** se puede observar como esta implementada la placa MDA300CA.

Figura 3.19 Conexiones MDA300CA

3.3 IMPLEMENTACIÓN DEL NODO GATEWAY

Para la implementación del nodo *Gateway* se hace uso de la aplicación *BaseStation* desarrollado por la universidad de Berkeley en California.

Para instalar la aplicación en el nodo que servirá como nodo Gateway, se debe conectar el nodo a la placa MIB520 y se ejecuta el **Segmento de código 3.25** en el directorio de la aplicación que por defecto es "/opt/tinyos-2.1.2/apps/BaseStation":

```
$ make iris install,0 mib520, /dev/ttyUSB0
```

Segmento de código 3.26 Instalación del Nodo Gateway

3.4 MINIAPLICACIÓN – TIEMPO REAL

Bajo el nombre de TiempoReal se creó una pequeña aplicación que tiene por objetivo enviar los valores tan pronto como son medidos por el nodo sensor a una aplicación que grafica estos datos conforme llegan, esto no forma parte del trabajo

de titulación pero fue creado para entender el trabajo realizado, el código se adjuntará en el Anexo D. Esta aplicación por su carácter mantiene el mayor tiempo posible encendida la interfaz de radio y como se recordará de la teoría, el mayor consumo de energía proviene de la utilización de esta interfaz.

Esta mini aplicación consta de una clase llamada TiempoReal en el Gateway encargada de enviar los datos a una dirección IP establecida en un archivo llamado RedBDD.txt explicada más adelante y de un formulario en la aplicación de usuario que le permite graficar los datos que provienen de un nodo sensor.

3.5 IMPLEMENTACIÓN DEL GATEWAY

Se conecta el nodo Gateway vía USB al computador con la ayuda de la placa MIB520CB. Para la implementación se ha creado una aplicación en Java encargada de recibir los mensajes del nodo Gateway y almacenarlos en la base de datos, esta aplicación necesita una clase para manipular los datos recibidos por los el nodo Gateway, esta clase por fortuna se puede crear de manera automática durante la compilación del código del nodo sensor utilizando el compilador MIG. En el presente trabajo de titulación esta clase se llama NodoSensorMensaje.java.

Es necesario tener permisos para permitir la comunicación por USB, para esto ejecutamos el **Segmento de código 3.27**:

```
$sudo chmod 777 /dev/ttyUSB*;
```

Segmento de código 3.27 Permisos USB

La velocidad de la plataforma y el puerto deben ser especificados con el **Segmento de código 3.28**:

```
$ export MOTECOM=serial@/dev/ttyUSB1:57600 ó

$ export MOTECOM=serial@/dev/ttyUSB1:iris
```

Segmento de código 3.28 Velocidad de la comunicación serial

Posteriormente se ejecuta el **Segmento de código 3.29**:

```
java net.tinyos.sf.SerialForwarder –comm

serial@/dev/ttyUSB1:iris
```

Segmento de código 3.29 Comunicación serial

El **Segmento de código 3.29** es necesario para recibir los paquetes del puerto serial, para simplificar al usuario la apertura del programa se ha creado un pequeño script llamado Inicio y se lo ejecuta con el **Segmento de código 3.30**:

```
./Inicio
```

Segmento de código 3.30 Script Inicio

Para simplificar la configuración de red se ha creado un pequeño documento de texto llamado “RedBDD.txt” que se observa en la **Figura 3.20**, este archivo posee los parámetros de la base de datos. Las opciones de Tiempo Real son utilizadas solo en el desarrollo del prototipo mas no forma parte de este.

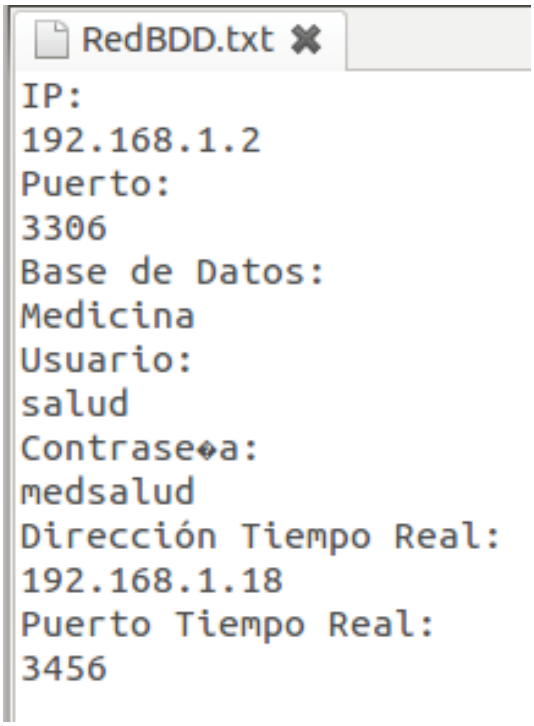


Figura 3.20 RedBDD

En la **Figura 3.21** se observa el diagrama UML de la aplicación ejecutada en el computador Gateway.

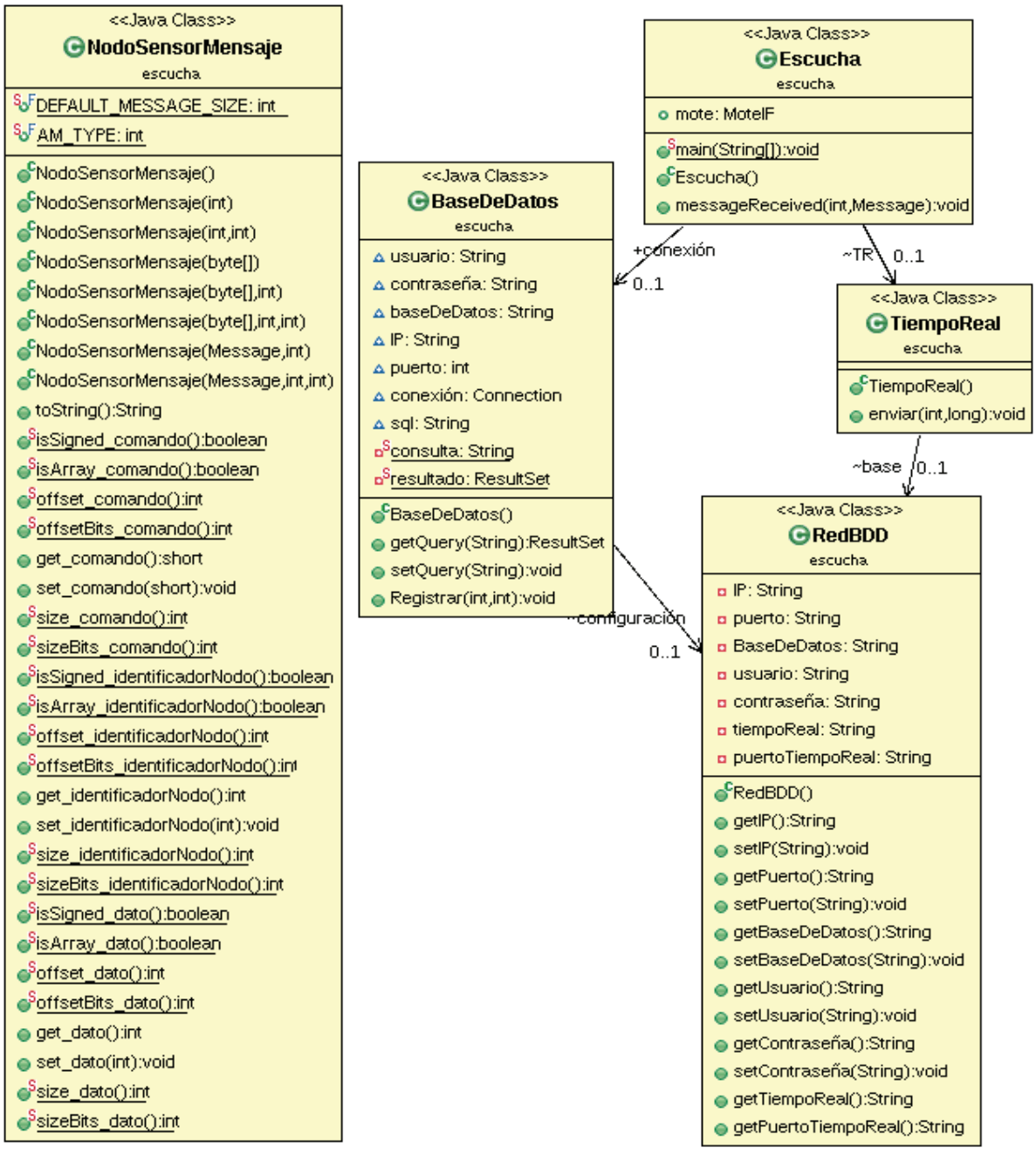


Figura 3.21 Diagrama UML computador Gateway – Eclipse

El código utilizado para el computador Gateway fue implementado utilizando el lenguaje de programación Java, y posee 3 clases propias y una generada por MIG:

- RedBDD encarga de obtener los datos del documento RedBDD.txt
- BaseDeDatos encargada de gestionar la comunicación con la base de datos.
- Escucha, clase principal y encargada de gestionar la llegada de mensajes del nodo Gateway.
- NodoSensorMensaje que es la clase generada por MIG y posee la configuración de los mensajes enviados por los nodos sensores.

En el **Segmento de código 3.31** se muestra el código de la clase escucha, donde se registra el nodos sensor o mote para ser escuchado.

```
public Escucha(){

    try{

    //Crea un objeto de la clase MoteIF

        mote = new MoteIF();
    //Prepara la escucha de mensajes del nuevo mote

        mote.registerListener(new NodoSensorMensaje(), this);
    }
    //Captura la excepción si el registro de un nuevo mote falla
    catch(Exception e){

        System.out.println("Error en registrar nodos");
    }

}
```

Segmento de código 3.31 Registro de mote en la clase Escucha

El método *messageRecieved* que se muestra en el **Segmento de código 3.32** está encargado de manipular la recepción del mensaje, el código implementado obtiene de este el identificador del nodo y el valor de la frecuencia cardíaca para llamar al método “Registrar” cuando el comando es 0, o al método de “Tiempo Real” cuando el comando es 1.

```
@Override
//Método ejecutado al recibir un mensaje
public void messageReceived(int arg0, Message msj) {
    //Variable que almacena el mensaje recibido
    NodoSensorMensaje mensaje = (NodoSensorMensaje)msj;
    //Obtiene el valor del comando para ejecutar el código del
    //gateway
    int comando=mensaje.get_comando();
    //Se obtiene el identificador del nodo
    int nodoID=mensaje.get_identificadorNodo();
    //Se obtiene el valor del pulso
    long pulso= mensaje.get_dato();
    switch(comando){
      case 0:
      //Se registra el valor en la base de datos
      conexión.Registrar(nodoID, (int)pulso);
      BDD.Registrar(nodoID, (int)pulso);
      System.out.println("\nComando: "+comando+"\t\tNodo:
"+nodoID+ "\t\tFrecuencia: "+pulso+" ppm");
      break;
      case 1:
      //Se envían todos los valores
      System.out.println(".");
      TR.enviar(nodoID, pulso);
      break;
      }
    }
```

Segmento de código 3.32 Método *messageReceived*

La clase BaseDeDatos es la encargada del almacenamiento de la frecuencia cardíaca en la base de datos junto con la hora en la que el dato es almacenado.

3.6 IMPLEMENTACIÓN DE LA BASE DE DATOS

Para la base de datos y basados en el diseño realizado en el capítulo anterior, como primer punto debemos permitir el acceso remoto y posteriormente se explicará cada una de las tablas y relaciones entre ellas.

La base de datos para este trabajo de titulación es gestionada por MySQL y se instaló en un sistema operativo con base Linux.

3.6.1 CONEXIÓN REMOTA DE BASES DE DATOS

Se debe establecer la configuración necesaria que permita el acceso remoto a nuestra base de datos.

Para permitir la conexión remota de MySQL se modifica el fichero que se encuentra en la ruta "/etc/mysql/my.cnf" eliminado o colocando símbolo de numeral (#) antes del **Segmento de código 3.33**:

```
#bind-address     = 127.0.0.1
```

Segmento de código 3.33 *bind-address*

Posterior ingresando a mysql desde un terminal, se procede a crear un usuario y asignar privilegios con el **Segmento de código 3.34**:

```
GRANT <Privilegios> ON <BaseDeDatos.tabla> to <Usuario>@XXXX

IDENTYFIED BY <Contraseña>
```

Segmento de código 3.34 Agregar usuario y privilegios de la base de datos

Para conceder todos los privilegios se coloca la palabra "ALL" en privilegios. Para seleccionar todas las tablas se debe colocar el signo "*".

Si la conexión proviene de cualquier host se coloca el "%" después del símbolo "@".

Si se desea remover un usuario el **Segmento de código 3.35**, previamente se debe conocer el nombre del usuario y la contraseña que tiene este usuario en para acceder a la base de datos:

```
REVOKE <Privilegios> ON <BaseDeDatos.tabla> to <Usuario>@XXXX
IDENTYFIED BY <Contraseña>
```

Segmento de código 3.35 Remover usuario y privilegios

Para conocer que usuarios tenemos en nuestra base se ejecuta **Segmento de código 3.36**:

```
SELECT HOST, USER SELECT_PRIV FROM mysql.user;
```

Segmento de código 3.36 Usuarios registrados en la base de datos

Si se desea conocer los privilegios que posee un usuario en particular se procede a ejecutar en **Segmento de código 3.37**:

```
SHOW GRANTS FOR usuario@xxxx;
```

Segmento de código 3.37 Conocer privilegios de usuarios

Para reiniciar el servicio de mysql se utiliza el **Segmento de código 3.38**:

```
service mysql restart
```

Segmento de código 3.38 Restaurar MySQL

3.6.2 IMPLEMENTACIÓN DE LA BASE DE DATOS

La base de datos en base a los requerimientos y el diseño establecido consta de 2 tablas fijas que son Historias_Clínicas y Nodos_Sensores.

En **Tabla 3.1** y **Tabla 3.2** se describe cada uno de los atributos de las tablas mencionadas anteriormente obtenidos de las historias de usuario de la implementación de la metodología SCRUM a mostrarse en este capítulo.

Con cada nueva historia clínica se genera una tabla que almacena los datos de la frecuencia cardíaca, descrita en la **Tabla 3.3**. La información que se encuentra en esta última tabla será la que utilizará el nodo sensor para realizar las gráficas de la frecuencia cardíaca en función del tiempo.

Tabla 3.1 Atributos de la tabla Historias_Clínicas

Historias_Clínicas			
Atributo	**Tamaño**	**Tipo**	**Descripción**
Número	64	BIGINT	Identificador de historia clínica. Clave principal.
Identificación	64	BIGINT	Identificación del paciente (cédula de identidad)
Nombres	45	VARCHAR	Nombres del paciente
Apellidos	45	VARCHAR	Apellidos del paciente
Sexo	1	VARCHAR	Sexo del paciente, masculino o femenino (M o F)
Edad	3	INT	Edad en años del paciente
Habitación	45	VARCHAR	Lugar en el que se encuentra el paciente
Frecuencia_Cardíaca	45	VARCHAR	Nombre de la tabla que almacena la frecuencia cardíaca de esta historia clínica
Nodos_Sensores_Identificador	3	INT	Identificador del nodo sensor asignado a esta historia clínica

Los nodos sensores serán almacenados únicamente por su identificador, el mismo que será asignado durante el proceso de instalación del código en el nodo sensor. Las tablas de frecuencia cardíaca contendrán cada una la frecuencia cardíaca y la hora del servidor de cada dato que es almacenado.

Tabla 3.2 Atributos de la tabla Nodos_Sensores

Nodos_Sensores			
Atributo	**Tamaño**	**Tipo**	**Descripción**
Identificador	3	INT	Identificador de cada nodo sensor

Tabla 3.3 Atributos de la tabla Frecuencia Cardíaca

FC{Número de Historia Clínica}			
Atributo	**Tamaño**	**Tipo**	**Descripción**
Fecha	-	DATETIME	Momento en que es almacenado el valor de frecuencia cardíaca en el servidor de base de datos
Frecuencia	3	INT	Frecuencia cardíaca recibida del nodo sensor

3.7 IMPLEMENTACIÓN DE LA APLICACIÓN DE USUARIO

La aplicación fue desarrollada utilizando la metodología SCRUM, previamente se entrevistó a profesionales de la salud, médicos y enfermeras.

3.7.1 PRESENTACIÓN DEL EQUIPO DE TRABAJO Y ROLES

Para trabajar con la metodología SCRUM se debe definir los roles de cada integrante del equipo, en la **Tabla 3.4** se aprecian los roles asignados para este trabajo de titulación.

Tabla 3.4 Roles de SCRUM

ROL	PERSONA
Dueño del producto	Md. Consuelo Quispe
Scrum *Master*	Ing. Carlos Egas
Equipo de Desarrollo	Mena Acosta Byron

3.7.2 DEFINICIÓN DE LAS HISTORIAS DE USUARIO

Las historias de usuario deben contener un nombre, criterios de aceptación y una narración [28]. El formato para las historias de usuario utilizado en el presente proyecto será el indicado en la **Tabla 3.5**, y contiene lo siguiente:

- **Número:** Descripción secuencial de cada una de las historias de usuario.
- **Usuario:** Es quien indica la funcionalidad.
- **Nombre de Historia:** Una breve descripción de la historia de usuario.

- **Descripción:** Se describe y enfoca los objetivos de la historia de usuario.
- **Criterios de Aceptación:** Indica como el usuario confirma si la implementación ha sido correcta.

Tabla 3.5 Historia de Usuario

Historia de Usuario	
Número:	**Usuario:**
Nombre de Historia:	
Descripción:	
Criterios de Aceptación:	

3.7.3 DESCRIPCIÓN DE LAS HISTORIAS DE USUARIO

Las historias de usuario han sido definidas en base a reuniones de levantamiento de requisitos. En la **Tabla 3.6** se describen los requerimientos que se desean implementar, el usuario y un identificador para numerar las historias de usuario.

Tabla 3.6 Tabla de requerimientos

Ítem	Identificador	Usuario	Requerimiento
1	HU1	Enfermera	Creación de la interfaz de usuario.
2	HU2	Médico general	Consulta de información.
3	HU3	Médico general	Descargar información.

3.7.4 DESCOMPOSICIÓN DE LAS HISTORIAS DE USUARIO

Las historias de usuario deben ser descompuestas hasta que puedan caber en un *Sprint*, esta descomposición permite una retroalimentación entre las partes antes de detallar el *Product Backlog*, esta descomposición se muestra en la **Tabla 3.7** HC01-01 a la **Tabla 3.13** HC03-01.

Estas historias fueron obtenidas de diálogos entablados con médicos y enfermeros que expresaron libremente y a su consideración los requisitos que la aplicación debería tener.

Tabla 3.7 HC01-01

Historia de Usuario	
Número: HC01-01	**Usuario:** Enfermero
Nombre de Historia: Ingresar una nueva historia clínica.	
Descripción: Enfermero desea registrar una nueva historia clínica con: 1. Número de Historia Clínica 2. Número de cédula de identificación 3. Nombres 4. Apellidos 5. Edad 6. Sexo 7. Habitación	
Criterios de Aceptación: La aplicación permite ingresar una historia clínica.	

Tabla 3.8 HC01-02

Historia de Usuario	
Número: HC01-02	**Usuario:** Enfermero
Nombre de Historia: Modificar una historia clínica.	
Descripción: Yo como enfermero deseo modificar los datos de una historia clínica.	
Criterios de Aceptación: La aplicación permite modificar los datos de una historia clínica.	

Tabla 3.9 HC01-03

Historia de Usuario	
Número: HC01-03	**Usuario:** Enfermero
Nombre de Historia: Eliminar una historia clínica.	
Descripción: Enfermero desea eliminar una historia clínica.	
Criterios de Aceptación: La aplicación permite eliminar una historia clínica.	

Tabla 3.10 HC01-04

Historia de Usuario	
Número: HC01-04	**Usuario:** Enfermero
Nombre de Historia: Asignar un sensor a un paciente.	
Descripción: Enfermero desea asignar un sensor a un paciente.	
Criterios de Aceptación: La aplicación permite asignar un sensor a un paciente.	

Tabla 3.11 HC01-05

Historia de Usuario	
Número: HC01-05	**Usuario:** Enfermero
Nombre de Historia: Visualizar el número de sensores disponibles.	
Descripción: Enfermero desea visualizar el número de sensores disponibles.	
Criterios de Aceptación: La aplicación permite visualizar cuantos sensores se encuentran disponibles.	

Tabla 3.12 HC02-01

Historia de Usuario	
Número: HC02-01	**Usuario:** Médico
Nombre de Historia: Visualizar los registros de la frecuencia cardíaca en un gráfico de líneas.	
Descripción: Médico desea visualizar los registros de la frecuencia cardíaca de un paciente en un gráfico de líneas de las últimas 2 horas.	
Criterios de Aceptación: La aplicación permite visualizar los registros de la frecuencia cardíaca de un paciente en un gráfico de líneas.	

Tabla 3.13 HC03-01

Historia de Usuario	
Número: HC03-01	**Usuario:** Médico
Nombre de Historia: Descargar registros de la frecuencia cardíaca en un gráfico de líneas.	
Descripción: Médico desea descargar los registros de la frecuencia cardíaca de un paciente en un gráfico de líneas.	
Criterios de Aceptación: La aplicación permite descargar los registros de la frecuencia cardíaca de un paciente en un gráfico de líneas.	

3.7.5 ELABORACIÓN DEL PRODUCT BACKLOG

En este punto se detallan todos los requerimientos que tendrá el prototipo, estos requerimientos se declaran en las historia de usuario junto con un nivel de prioridad, el *Product Backlog* para este trabajo de titulación es definido acorde a los requerimientos obtenidos de las historias de usuario y requerimientos adicionales definidos por el desarrollador como se muestra en la **Tabla 3.14**.

Tabla 3.14 *Product Backlog*

ID	Prioridad	Historia de Usuario	Descripción del requerimiento
1	Alta	Definido por el autor	Definir la arquitectura de la aplicación
			Construir la base de datos.
2	Alta	HC01-01	Ingresar una nueva historia clínica.
		HC01-02	Modificar una historia clínica.
		HC01-03	Eliminar una historia clínica.
		HC01-04	Asignar un sensor a un paciente.
		HC01-05	Visualizar el número de sensores disponibles.
3	Media	HC02-01	Visualizar los registros de la frecuencia cardíaca en un gráfico de líneas.
4	Media	HC03-01	Descargar registros de la frecuencia cardíaca en un gráfico de líneas.

3.7.6 ELABORACIÓN DE LOS SPRINTS

Después de elaborado el *Product Backlog* y sometido a un análisis para aclarar dudas se procede a planificar la iteración en un lista llamada *Sprint Backlog* procurando realizar una estimación del esfuerzo necesaria para completar cada tarea, definiendo su duración preferentemente en horas **Tabla 3.15** y la **Tabla 3.16** se define el *Sprint Backlog* utilizado en este trabajo de titulación:

Tabla 3.15 *Sprint Backlog* (Parte I)

ID	Requerimiento	Tareas	Esfuerzo	Horas
1	Definir la arquitectura de la aplicación	Definir la navegación de la aplicación.	Alto	4
		Entender la lógica del negocio	Alto	6
	Construir la base de datos.	Diseñar las tablas, y atributos de la base de datos.	Alto	10
2	Ingresar una nueva historia clínica.	Implementar métodos y formulario para ingresar una nueva historia clínica.	Alto	10
	Modificar una historia clínica.	Implementación formulario para actualizar y eliminar una historia clínica	Alto	6

Después de elaborado el *Product Backlog* y sometido a un análisis para aclarar dudas se procede a planificar la iteración en un lista llamada *Sprint Backlog* procurando realizar una estimación del esfuerzo necesaria para completar cada tarea, definiendo su duración preferentemente en horas.

En esta lista se definen las diferentes tareas incluyendo 2 campos extras, en el primero de ellos se detalla la cantidad de esfuerzo considerando entre otras variables el manejo de las herramientas y la experiencia.

En esta lista se ha catalogado el esfuerzo como:

- Alto
- Medio
- Bajo

Tabla 3.16 *Sprint Backlog* (Parte II)

ID	Requerimiento	Tareas	Esfuerzo	Horas
2		Implementar métodos para buscar una historia clínica.	Alto	8
		Implementar método para actualizar una historia clínica	Alto	8
	Eliminar una historia clínica.	Implementar métodos para eliminar una historia clínica.	Alto	8
	Visualizar el número de nodos sensores disponibles	Implementación del formulario para ingresar, eliminar y visualizar nodos.	Alto	3
		Implementar métodos para ingresar un nuevo nodo sensor	Medio	6
		Implementar métodos para eliminar un nuevo nodo sensor	Medio	6
		Implementar métodos para visualizar el número de nodos sensores disponibles.	Medio	6
		Implementar métodos e interfaz de tabla informativa de nodos sensores desconectados	Medio	7
3	Visualizar los registros de la frecuencia cardíaca en un gráfico de líneas.	Implementar formulario para visualizar el registro de la frecuencia cardíaca.	Alto	4
		Implementar métodos para buscar una historia clínica.	Alto	4
		Implementar métodos para mostrar los registros de la frecuencia cardíaca en un gráfico de líneas	Alto	10
4	Descargar registros de la frecuencia cardíaca en un gráfico de líneas.	Implementar métodos e interfaz para descargar registros de la frecuencia cardíaca como imagen .png o .jpeg	Bajo	4
5	Crear la interfaz para el ingreso a la aplicación	Implementar métodos y formulario de menú principal	Alto	4
		Implementar métodos e interfaz para ingresar a la aplicación.	Alto	4

Se define como Alto a todas aquellas tareas que ameriten por un lado de una profunda investigación y/o elevada experiencia en el manejo de una determinada herramienta, también tareas de diseño.

Se define como Bajo a aquellas tareas que en las que el desarrollador posee una amplia experiencia y conocimiento acerca de la tarea a ejecutar, o en su defecto el desarrollador posee la herramienta facilitándole una tarea que de otro modo sería muy extensa de desarrollar.

Se define como Medio a aquellas tareas que si bien ameritan un amplio conocimiento o experiencia, una parte de ese conocimiento y experiencia está en poder del desarrollador por razones ajenas al desarrollo del trabajo de titulación.

Las horas no poseen relación directa con el esfuerzo involucrado, si bien existen tareas con un nivel de esfuerzo alto pueden requerir poco tiempo si existe una gran disponibilidad de fuentes bibliográficas para solucionar el problema pero que poseen un nivel técnico superior al tradicionalmente manejado por el desarrollador.

De la misma manera una tarea con un esfuerzo bajo puede requerir demasiado tiempo como son por ejemplo las tareas de programación que una vez obtenida la experiencia se resuelven fácilmente, es decir poseerían un esfuerzo bajo mientras que el tiempo estaría definido por el tamaño del código que se genera.

En base a las consideraciones anteriormente expuestas se estableció un *Product Backlog* a consideración de los miembros de Scrum.

3.7.7 ANÁLISIS DE LA OBTENCIÓN DE LA LISTA DE TAREAS

Después de elaborado el *Sprint Backlog* se establece que la duración en el desarrollo de la aplicación es de 83 horas. Se planifica trabajar en el trabajo de titulación 6 horas diarias, para lo que se necesita un total de 14 días de trabajo. Se planifica trabajar 5 días a la semana (días laborables). Se toma la decisión de desarrollar la aplicación en 3 *Sprint* con una duración de 6 días por *Sprint*.

3.7.8 EJECUCIÓN, REVISIÓN Y EVALUACIÓN DE LOS SPRINTS

Al terminar cada *Sprint*, Scrum propone realizar una revisión (*Sprint Review*) y modificar el *Product Backlog* de ser necesario. El equipo entrega el trabajo terminado por cada *Sprint* mostrando los problemas encontrados y como se resolvieron.

En este punto el dueño del producto realiza una revisión y posterior a las pruebas de aceptación da su aprobación si se cumplen los requisitos indicados en las historias de usuario.

Las pruebas de aceptación tienen el siguiente formato indicado en la **Tabla 3.17** con los siguientes campos:

- **Número:** Campo que indica el número de la prueba a realizarse asociado a la historia de usuario.
- **Escenario:** Campo que identifica la historia de usuario sobre la que se realiza la prueba.
- **Requisitos:** Descripción de las condiciones que desencadenan el escenario.
- **Evento:** Acción que ejecuta el usuario acorde a los requisitos de la prueba.
- **Resultado esperado:** Es el comportamiento deseado del software acorde a los requisitos y el evento ejecutado por el usuario.
- **Evaluación:** Campo que describe una prueba como exitosa o fallida de acuerdo al resultado esperado.

Tabla 3.17 Formato de pruebas de aceptación

PRUEBAS DE ACEPTACIÓN	
Número:	
Escenario:	
Requisitos:	
Evento:	
Resultado esperado:	
Evaluación:	

3.7.8.1 Sprint 1

En este *Sprint* se pretende realizar el diseño de la aplicación y la base de datos.

3.7.8.1.1 Tareas del Sprint

Las tareas del *Sprint* se muestran en la **Tabla 3.18**.

Tabla 3.18 Tareas: *Sprint* 1

ID	Requerimiento	Tareas	Esfuerzo	Horas
1	Definir la arquitectura de la aplicación.	Definir la navegación de la aplicación.	Alto	4
		Entender la lógica del negocio.	Alto	6
	Construir la base de datos.	Diseñar las tablas, y atributos de la base de datos.	Alto	10
2	Ingresar una nueva historia clínica.	Implementar métodos y formulario para ingresar una nueva historia clínica.	Alto	10

3.7.8.1.2 Scrum Diario

Se muestran las tareas realizadas y si existieran, también las tareas que faltan por hacer. Se debe responder a cada una de estas preguntas:

- **¿Que hice ayer que ayudó al equipo de desarrollo a lograr el objetivo del *Sprint*?**

  Por tratarse del primer *Sprint* no existió una reunión previa.
- **¿Qué haré hoy para ayudar al equipo de desarrollo a lograr el objetivo del *Sprint*?**

  Comenzar a realizar las tareas propuestas en la pila de tareas utilizando conocimientos y consultas.
- **¿Veo algún impedimento que impida lograr el objetivo del *Sprint*?**

El problema que puede generarse es entender la lógica del negocio. En la **Tabla 3.19** se aprecia el esfuerzo diario durante el primer *Sprint* 1.

Por tratarse del primer *Sprint*, sus tareas fundamentales van enfocadas al diseño de la aplicación, por lo que no se puede elaborar pruebas a este nivel.

Al finalizar el *Sprint* se procederá a mostrar el número de tareas que se ejecutaron y en qué días, de esta manera se puede obtener una buena percepción de lo que se ha hecho y lo que falta por hacer.

Tabla 3.19 Esfuerzo diario - *Sprint* 1

SPRINT	INICIO	FIN
1	lunes, 29 de agosto de 2016	lunes, 05 de septiembre de 2016

							lunes	martes	miércoles	jueves	viernes	lunes
							29/08/2016	30/08/2016	31/08/2016	01/09/2016	02/09/2016	05/09/2016
					Tareas Pendientes		4	3	2	2	1	0
					Horas pendientes		30	24	18	12	6	0
ID	Requerimiento	Tareas	Responsable	Estado	Esfuerzo estimado	Horas	Esfuerzo					
1	Definir la arquitectura de la aplicación	Definir la navegación de la aplicación.	Byron	Finalizado	Alto	4	4					
		Entender la lógica del negocio	Byron	Finalizado	Alto	6	6	4				
	Construir la base de datos.	Diseñar las tablas, y atributos de la base de datos.	Byron	Finalizado	Alto	10	10	10	8	2		
2	Ingresar una nueva historia clínica.	Implementar métodos y formulario para ingresar una nueva historia clínica.	Byron	Finalizado	Alto	10	10	10	10	10	6	0

3.7.8.1.3 Revisión del Sprint 1

Se muestra cada una de las tareas ejecutadas durante el *Sprint* 1

- **Navegación por aplicación.**

La navegación se mostró durante el diseño de la aplicación en el capítulo anterior, indicada en la **Figura 2.13**.

- **Entender la lógica del negocio**

La lógica del negocio se mostró en el capítulo anterior durante el desarrollo de la aplicación. Se empleó el patrón modelo, vista, controlador para representar la lógica del negocio, como se muestra en la **Figura 2.14**.

- **Diseño de las tablas y atributos de la base de datos.**

El diseño de la base de datos se muestra en el capítulo anterior, página 84.

- **Implementación de la interfaz para ingresar una nueva historia clínica.**

La **Figura 3.22** muestra la interfaz para el ingreso de una nueva historia clínica.

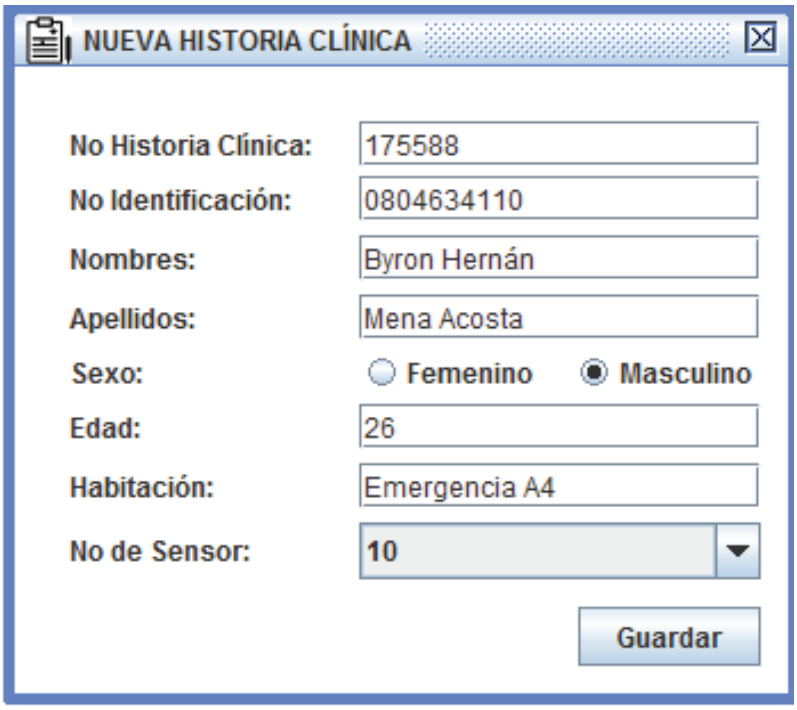


Figura 3.22 Interfaz de nueva historia clínica

Para las implementaciones de la aplicación de usuario se utilizaron jtables para las tablas, jtextfields para los campos de texto, jradiobuttons para los botones de opción y jbuttons para los botones. En la **Figura 3.22** se observa la interfaz para el ingreso de una historia clínica.

3.7.8.1.4 Retrospectiva del Sprint 1

Es necesario entender la lógica del negocio para la elaboración de las entidades y relaciones de la base de datos, sin un buen entendimiento de esta lógica la aplicación final corre el riesgo de volverse más compleja de desarrollar, requerir más tiempo a causa de eso y no cumplir con los plazos establecidos.

En la **Figura 3.23** se puede apreciar el esfuerzo realizado para cumplir con las tareas del *Sprint* 1 finalizándolas en el tiempo establecido.

Se emplearon 6 horas diarias para la realización del *Sprint* 1 los 5 primeros días, logrando que en el último día del *Sprint* existan cero tareas por realizar como los indica la **Figura 3.23**.

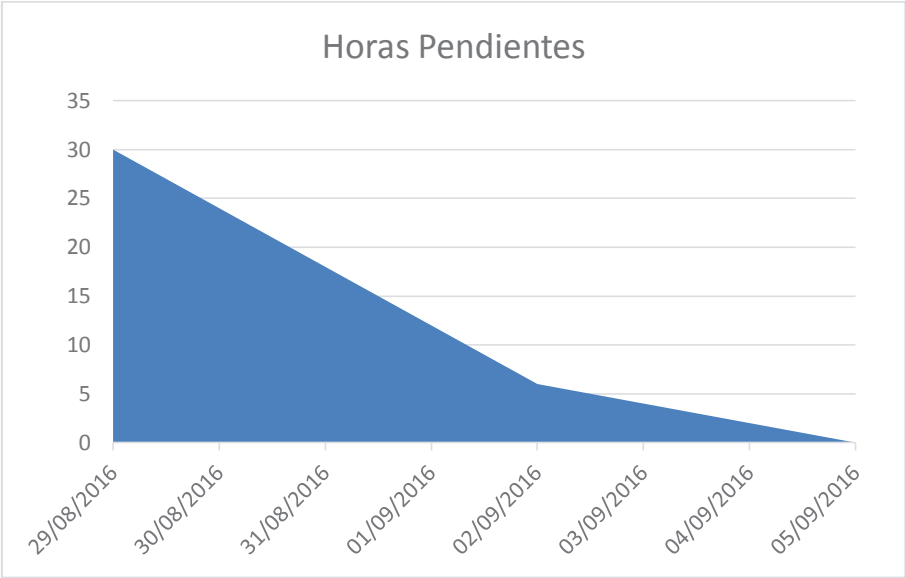


Figura 3.23 Esfuerzo *Sprint* 1

En la **Figura 3.24** se puede apreciar el número de tareas que fueron desarrolladas durante el *Sprint* 1, por la complejidad no fue posible avanzar con 2 tareas a la vez.

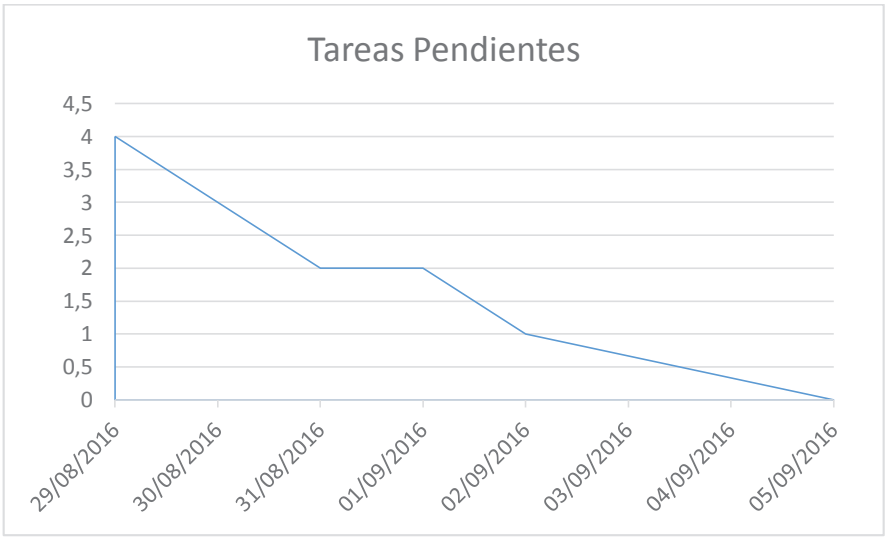


Figura 3.24 Tareas: *Sprint* 1

3.7.8.1.5 Incremento del Sprint

El resultado al finalizar el *Sprint* 1 fue el esperado, teniendo un producto potencialmente "terminado" para comenzar con el *Sprint* 2. No se realizan cambios de prioridad ni de requerimientos por lo que el *Product Backlog* se mantiene igual.

3.7.8.1.6 Pruebas de Aceptación

Las pruebas no aplican para el ID número 1 debido a que fueron orientadas a la parte de desarrollo.

El ID número 2 constaba de las siguientes, las pruebas de aceptación se muestran en las **Tabla 3.20**, **Tabla 3.21** y **Tabla 3.22** :

En la **Tabla 3.20** se puede observar que no existían requisitos previos para llenar una historia clínica.

La **Tabla 3.21** describe los eventos que se producen al ingresar una nueva historia clínica iniciando por el llenado de los datos, para posteriormente intentar guardarlos. Dentro de los limitantes que debe tener la aplicación es evitar que una historia clínica sea guardada 2 veces. Aquí también se observa cuáles fueron los resultados esperados, el principal de ellos es que se guarde la historia clínica en la base de datos y posteriormente el formulario se limpie para con ello permitir el ingreso de una nueva historia clínica.

Ante la presencia de datos sin una historia clínica, la aplicación inhabilita la opción de registrarlos en la base de datos.

Como se puede observar en los resultados la prueba fue exitosa, la prueba fue realizada por el dueño de la aplicación.

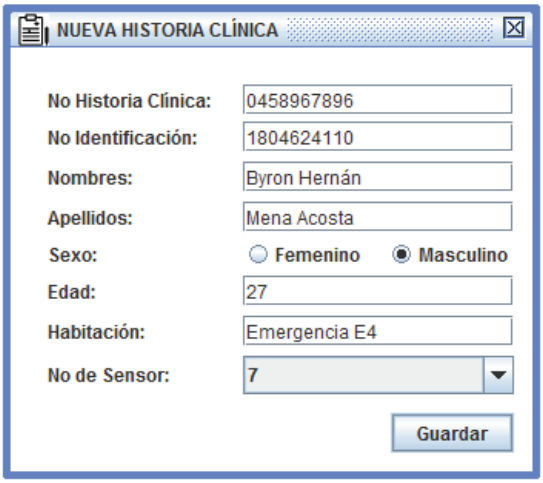
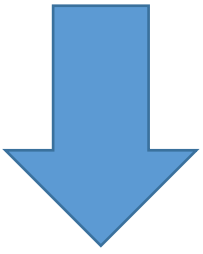
Como se puede observar en los resultados después de ingresar los datos en la aplicación, mediante una ventana de terminal se ingresa a la base de datos y se pude constatar que efectivamente la historia clínica se ha almacenado.

HU01-01 Ingresar una nueva historia clínica.

Tabla 3.20 PA1-01 (Parte I)

PRUEBAS DE ACEPTACIÓN	
Número:	PA1-01
Escenario:	HU01-01 Ingresar una nueva historia clínica.
Requisitos:	Ninguno

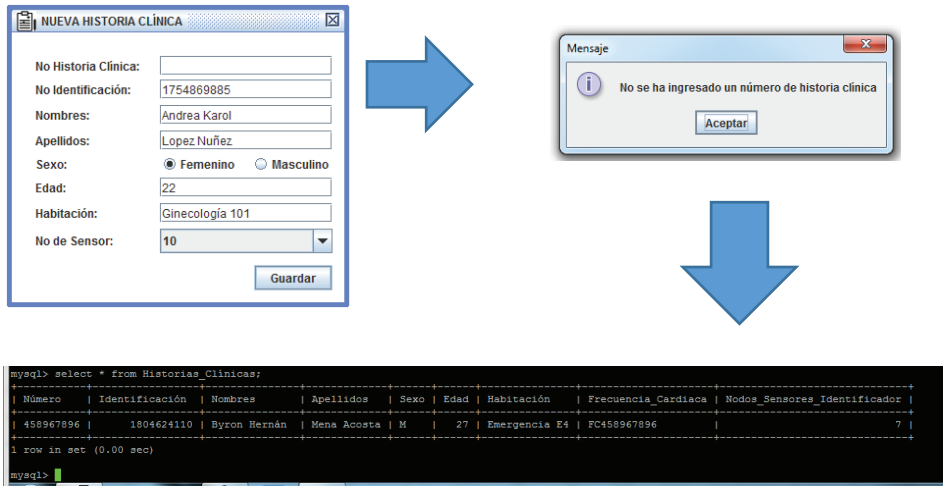
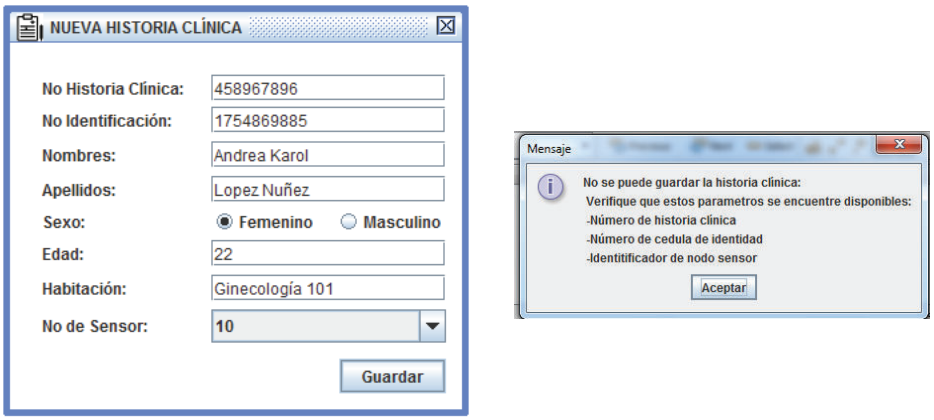
Tabla 3.21 PA1-01 (Parte II)

PRUEBAS DE ACEPTACIÓN	
Evento:	1. Llenar los datos de filiación de un paciente 2. Intentar guardar una historia clínica no numerada 3. Intentar guardar una historia clínica con número de repetido 4. Presionar el botón guardar después de ingresar una historia clínica correcta
Resultado esperado:	1. Permite el ingreso de los datos de filiación del paciente 2. No se guarda una historia clínica sin numeración 3. Se verifica que no se repita el número de historia clínica 4. Se limpiara el formulario después de guardar.
Evaluación:	Prueba exitosa.
Resultados	1. Llenar los datos de filiación de una paciente y presionar el botón guardar, posteriormente verificar que se encuentran en la base datos.   


```
mysql> select * from Historias_Clinicas;
+-----------+-----------------+-------------+-------------+------+------+----------------+---------------------+-----------------------------+
| Número    | Identificación  | Nombres     | Apellidos   | Sexo | Edad | Habitación     | Frecuencia_Cardiaca | Nodos_Sensores_Identificador |
+-----------+-----------------+-------------+-------------+------+------+----------------+---------------------+-----------------------------+
| 458967896 |      1804624110 | Byron Hernán | Mena Acosta | M    |   27 | Emergencia E4 | FC458967896         |                           7 |
+-----------+-----------------+-------------+-------------+------+------+----------------+---------------------+-----------------------------+
1 row in set (0.00 sec)

mysql>
```

Tabla 3.22 PA1-01 (Parte III)

Resultados	2. Ingreso de todos los datos, excepto el número de historia y presionar el botón guardar.  3. Ingreso de un número de historia clínica ya existente  4. Limpieza del formulario después de guardar Se muestra en el punto 1.

3.7.8.2 Sprint 2

En este *Sprint* se pretende diseñar e implementar la interfaz para la eliminación y modificación de las historias clínicas, el diseño se muestra en el capítulo anterior.

3.7.8.2.1 Tareas del Sprint

Las tareas del *Sprint* se muestran en la **Tabla 3.23**.

Tabla 3.23 Tareas: *Sprint* 2

ID	Requerimiento	Tareas	Esfuerzo	Horas
2	Modificar una historia clínica.	Implementación formulario para actualizar y eliminar una historia clínica	Alto	6
		Implementar métodos para buscar una historia clínica.	Alto	8
		Implementar método para actualizar una historia clínica	Alto	8
	Eliminar una historia clínica.	Implementar métodos para eliminar una historia clínica.	Alto	8

3.7.8.2.2 Scrum Diario

Se muestran las tareas realizadas y las tareas que aún faltan por hacer en caso de ser necesario. Se debe responder a cada una de estas preguntas:

- **¿Que hice ayer que ayudó al equipo de desarrollo a lograr el objetivo del *Sprint*?**

  Se consiguieron los resultados del *Sprint* 1, se tomó la decisión de descansar un día para renovar energías. No existen tareas pendientes por realizar.

- **¿Qué haré hoy para ayudar al equipo de desarrollo a lograr el objetivo del *Sprint*?**

  Comenzar a realizar las tareas del *Sprint* 2, se realizarán las consultas e investigaciones necesarias para cumplir con éxito los objetivos planteados

- **¿Veo algún impedimento que impida lograr el objetivo del *Sprint*?**

  No se observan ningún impedimento, se puede tener problemas al actualizar una historia clínica.

En la **Tabla 3.24** se aprecia el esfuerzo diario durante el segundo *Sprint*.

Tabla 3.24 Esfuerzo diario: *Sprint* 2

SPRINT	INICIO	FIN
1	martes, 06 de septiembre de 2016	martes, 13 de septiembre de 2016

							martes	miércoles	jueves	viernes	lunes	martes
							06/09/2016	07/09/2016	08/09/2016	09/09/2016	12/09/2016	13/09/2016
					Tareas Pendientes		3	3	2	2	1	0
					Horas pendientes		30	24	18	12	6	0
ID	Requerimiento	Tareas	Responsable	Estado	Esfuerzo estimado	Horas	Esfuerzo					
2	Modificar una historia clínica.	Implementar métodos e interfaz para buscar una historia clínica.	Byron	Finalizado	Alto	10	10	4				
		Implementar método e interfaz para actualizar una historia clínica	Byron	Finalizado	Alto	10	10	10	8	2		
	Eliminar una historia clínica.	Implementar métodos e interfaz para eliminar una historia clínica.	Byron	Finalizado	Alto	10	10	10	10	10	6	0

3.7.8.2.3 Revisión del Sprint 2

Se muestra cada una de las tareas ejecutadas durante el *Sprint* 2.

- **Implementación de la interfaz para buscar, actualizar y eliminar una historia clínica.**

La **Figura 3.25** muestra el formulario implementado que permitirá al usuario buscar una historia clínica, actualizar los datos y eliminar la historia clínica seleccionada.

El formulario consta de 3 botones, un botón “Actualizar” para guardar los datos que han sido modificados de la historia clínica que es la que esta seleccionada en el jtable, otro botón llamado “Eliminar” que permite borrar la historia clínica seleccionada del jtable y por último un botón “Buscar” que busca historias clínicas en base al parámetro del jcomboBox y el valor ingresado que se encuentra en el jtextField cercano al jtable.

3.7.8.2.4 Retrospectiva del Sprint 2

Se utilizó una interfaz común para las tareas correspondientes a este *Sprint*. En la **Figura 3.26** se puede apreciar el esfuerzo realizado para cumplir con las tareas del *Sprint*, finalizando las tareas en el tiempo establecido.

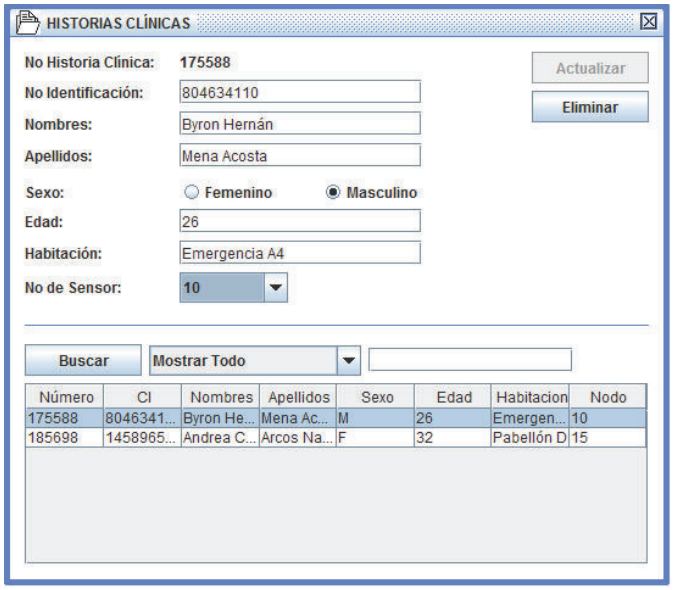


Figura 3.25 Interfaz para actualizar y eliminar una historia clínica

Se emplearon 6 horas diarias para la realización del *Sprint* de tal manera que el último día ya no existían tareas por completar.

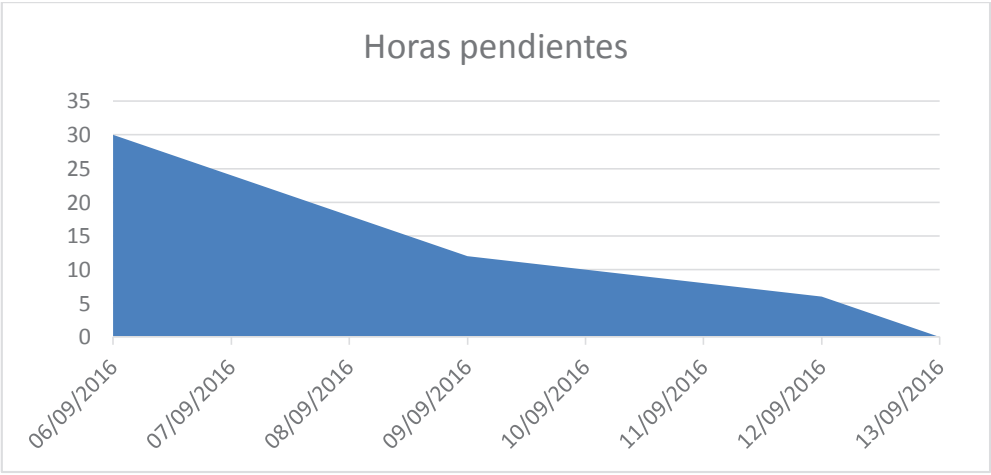


Figura 3.26 Esfuerzo: *Sprint* 2

En la **Figura 3.27** se pueden apreciar las tareas ejecutadas durante el *Sprint*, no fue posible ejecutar más de una tarea por día de trabajo.

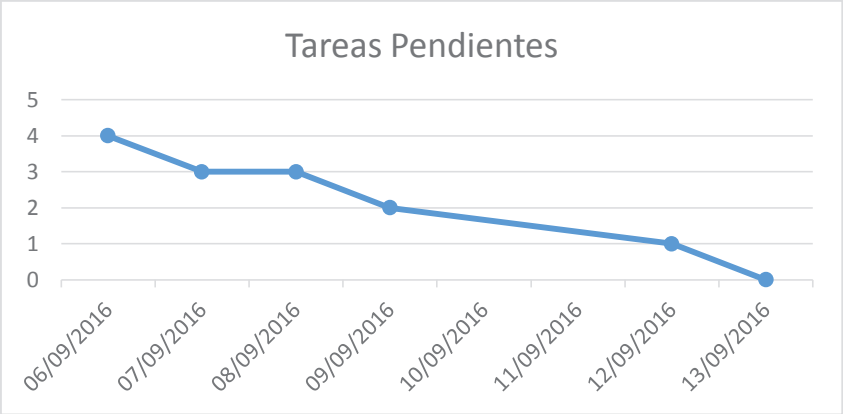


Figura 3.27 Tareas: *Sprint* 2

3.7.8.2.5 Incremento del Sprint

El resultado al finalizar el sprint 2 fue el esperado, teniendo un producto potencialmente "terminado" para comenzar con el *Sprint* 3. No se realizan cambios de prioridad ni de requerimientos por lo que el *Product Backlog* se mantiene igual.

3.7.8.2.6 Pruebas de Aceptación

Para el ID número 2 constan las siguientes tareas a las que deben ser aplicadas las pruebas de aceptación:

HU01-02	Modificar una historia clínica.
HU01-03	Eliminar una historia clínica

La **¡Error! La autoreferencia al marcador no es válida.** y **Tabla 3.26** son las pruebas de aceptación HU01-02.

La **Tabla 3.25** corresponde a los datos y eventos generados que serán generados durante la prueba de aceptación por parte del dueño del producto.

Para iniciar el usuario deberá seleccionar una historia clínica que será cargada en los campos que conforman el formulario.

Tabla 3.25 PA1-02 (Parte I)

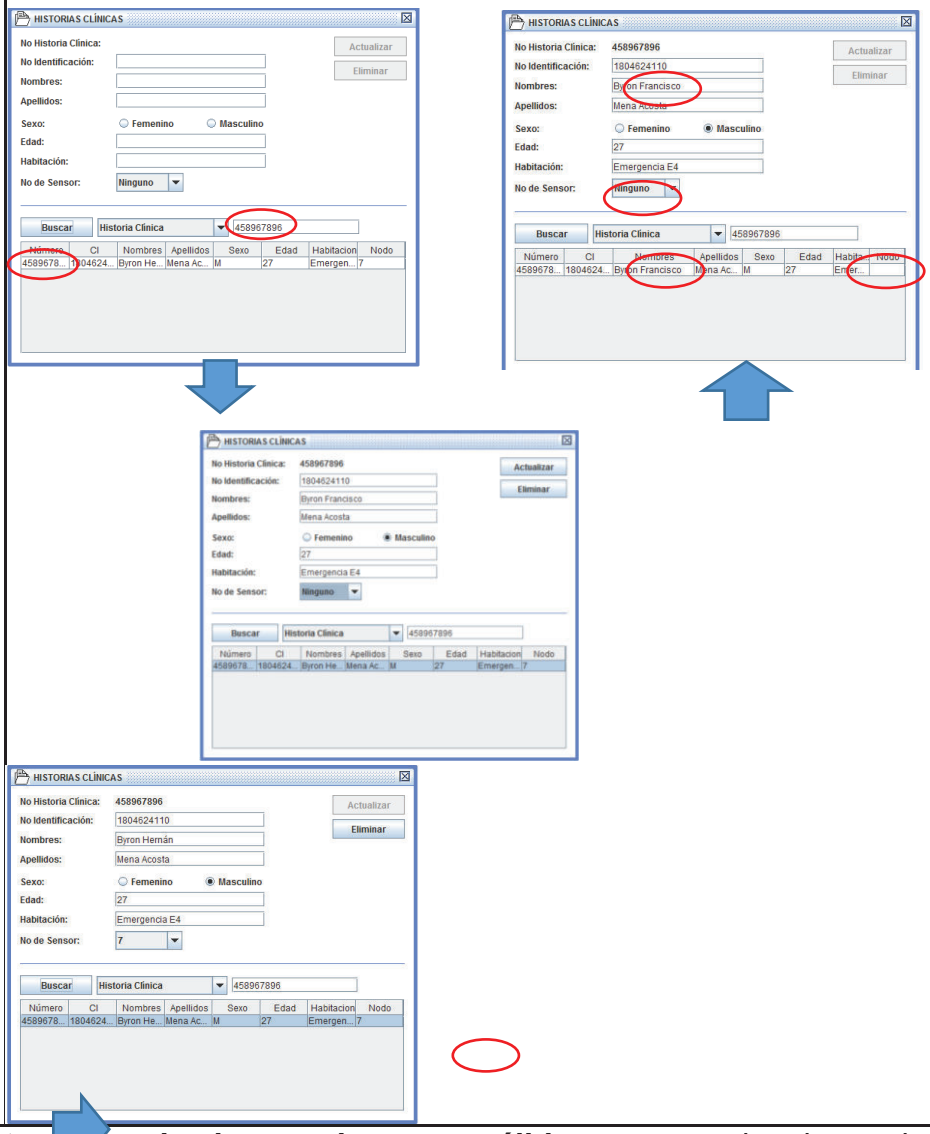
PRUEBAS DE ACEPTACIÓN	
Número:	PA1-02
Escenario:	HU01-02 Modificar una historia clínica.
Requisitos:	Seleccionar una historia clínica del jtable
Evento:	1. Seleccionar el parámetro de búsqueda "Historia Clínica" en el jcomboBox de búsqueda, llenar el jtextField con el valor a buscar y presionar el botón Buscar. 2. Seleccionar una historia clínica del jtable mediante un clic. 3. Cambiar los datos en los jtextfields, jradiobuttons y jcomboBox. 4. Presionar el botón actualizar.

Después de la aceptación en la **Tabla 3.26** se observa los resultados de la prueba, la búsqueda de una historia clínica, edición de los datos y actualización de los mismos.

Tabla 3.26 PA1-02 (Parte II)

PRUEBAS DE ACEPTACIÓN	
Evaluación	Prueba exitosa

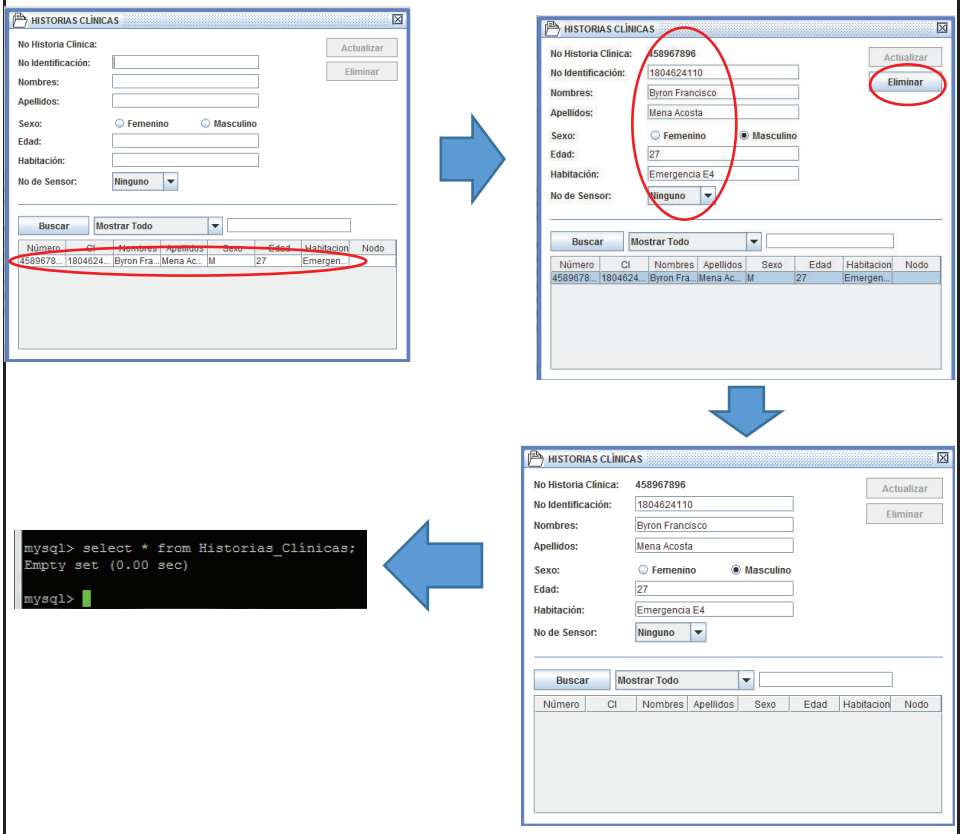
Resultados:



La **¡Error! La autoreferencia al marcador no es válida.** corresponde a la prueba de aceptación HU01-03

Tabla 3.27 PA1-03

PRUEBAS DE ACEPTACIÓN	
Número:	PA1-03
Escenario:	HU01-02 Modificar una historia clínica.
Requisitos:	Seleccionar una historia clínica del jtable
Evento:	1. Seleccionar una historia clínica del jtable mediante un clic. 2. Los datos de la historia clínica se muestran en los jtextFields, jradioButtons y jcomboBox. 3. Presionar el botón Eliminar

Resultado esperado:	1. Se elimina la historia clínica del jtable y de la base de datos.
Evaluación:	Prueba exitosa
Resultados:	

3.7.8.3 Sprint 3

En este *Sprint* se pretende diseñar e implementar la interfaz para ingresar, visualizar y eliminar los nodos, el diseño se muestra en el capítulo anterior.

3.7.8.3.1 Tareas del Sprint 3

En la Tabla 3.28 se muestran las tareas del sprint 3.

ID	Requerimiento	Tareas	Esfuerzo	Horas
2	Visualizar el número de	Implementación del formulario	Alto	3

	nodos sensores disponibles	para ingresar, eliminar y visualizar nodos.		
		Implementar métodos para ingresar un nuevo nodo sensor	Medio	6
		Implementar métodos para eliminar un nuevo nodo sensor	Medio	6
		Implementar métodos para visualizar el número de nodos sensores disponibles.	Medio	6

Tabla 3.28 Tareas: Sprint 3

3.7.8.3.2 Scrum Diario

Se muestran las tareas realizadas y las tareas que aún faltan por hacer en caso de ser necesario. Se debe responder a cada una de estas preguntas:

- **¿Que hice ayer que ayudó al equipo de desarrollo a lograr el objetivo del *Sprint*?**

  Se consiguieron los resultados del Sprint 2, se tomó la decisión de descansar un día para renovar energías. No existen tareas pendientes por realizar.

- **¿Qué haré hoy para ayudar al equipo de desarrollo a lograr el objetivo del Sprint?**

  Comenzar a realizar las tareas del Sprint 3, se deben realizar las consultas e investigaciones necesarias para cumplir con éxito cada uno de los objetivos planteados

- **¿Veo algún impedimento que impida lograr el objetivo del Sprint?**

  No se observa ningún impedimento.

En la **Tabla 3.29** se aprecia el esfuerzo diario durante el tercer Sprint.

SPRINT	INICIO	FIN
3	miércoles, 14 de septiembre de 2016	miércoles, 21 de septiembre de 2016

							miércoles	jueves	viernes	lunes	martes	miércoles
							14/09/2016	15/09/2016	16/09/2016	19/09/2016	20/09/2016	21/09/2016
					Tareas Pendientes		5	4	3	2	1	0
					Horas pendientes		28	22	16	10	4	0
ID	Requerimiento	Tareas	Responsable	Estado	Esfuerzo estimado	Horas	Esfuerzo					
2	Visualizar el número de nodos sensores disponibles	Implementación del formulario para ingresar, eliminar y visualizar nodos.	Byron	Finalizado	Alto	3	3					
		Implementar métodos para ingresar un nuevo nodo sensor	Byron	Finalizado	Medio	6	6	3				
		Implementar métodos para eliminar un nuevo nodo sensor	Byron	Finalizado	Medio	6	6	6	3			
		Implementar métodos para visualizar el número de nodos sensores disponibles.	Byron	Finalizado	Medio	6	6	6	6	3		
		Implementar métodos e interfaz de tabla informativa de nodos sensores desconectados	Byron	Finalizado	Medio	7	7	7	7	7	4	0

Tabla 3.29 Esfuerzo diario: Sprint 3

3.7.8.3.3 Revisión del Sprint 3

Se muestra cada una de las tareas ejecutadas durante el Sprint 3.

- **Implementación del formulario para ingresar, eliminar y visualizar nodos.**

La **Figura 3.28** muestra el formulario que permitirá al usuario visualizar los nodos sensores disponibles, ingresar un nuevo nodo sensor o eliminar un nodo sensor existente.

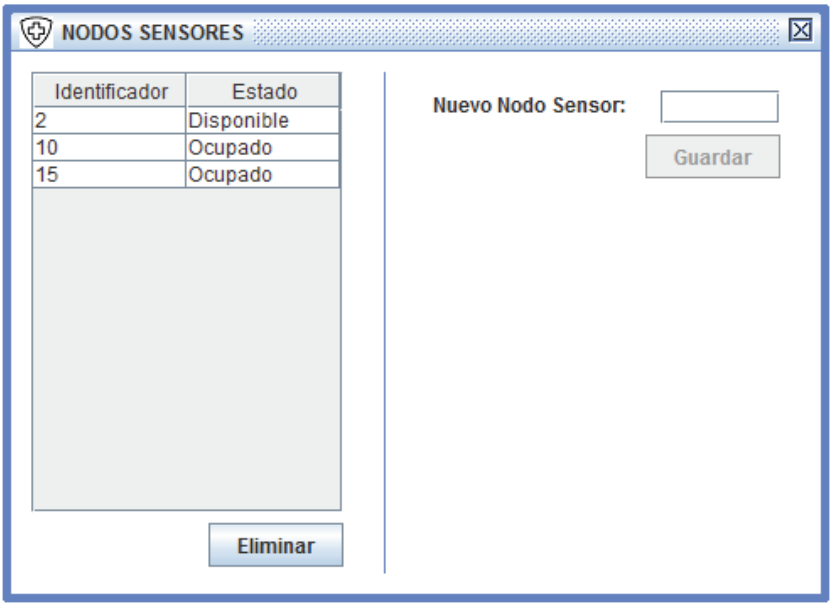


Figura 3.28 Formulario nodos sensores

El formulario consta de un jtable que posee los nodos registrados y un indicador de si están ocupados o disponibles, un jtextfield donde se escribirá el identificador de un nuevo nodo sensor para guardarlo, un botón "Guardar" que permite agregar un nuevo nodo sensor y un botón "Eliminar" que permite quitar un nodo sensor que no se encuentre en uso seleccionado previamente del jtable.

3.7.8.3.4 Implementación de métodos para ingresar un nuevo nodo sensor

Se implementó el código correspondiente al componente modelo y controlador para almacenar un nuevo nodo sensor en la base de datos.

3.7.8.3.5 Implementación de métodos para eliminar un nodo sensor

Se implementó el código correspondiente al componente modelo y controlador para eliminar un nodo sensor.

3.7.8.3.6 Implementación de métodos para visualizar el número de nodos sensores disponibles

Se implementó el código correspondiente al componente modelo y controlador para buscar los nodos disponibles.

3.7.8.3.7 Implementar métodos e interfaz de tabla informativa de nodos sensores desconectados

Fue agregado un "*List*" que permite mostrar un mensaje cada vez que se deje de transmitir acorde a los criterios del diseño, se implementó el código correspondiente al componente modelo y controlador. Se consideró con el dueño del producto que 20 minutos sin transmisión es indicativo de un nodo perdido.

En la figura Tabla 3.30 se puede apreciar el diagrama de clases encargado de la gestión de los nodos sensores

Tabla 3.30 Diagrama de clase nodos sensores

3.7.8.3.8 Retrospectiva del Sprint 3

En la **Figura 3.29** se puede apreciar el esfuerzo realizado para cumplir con las tareas del sprint, finalizando las tareas en el tiempo establecido.

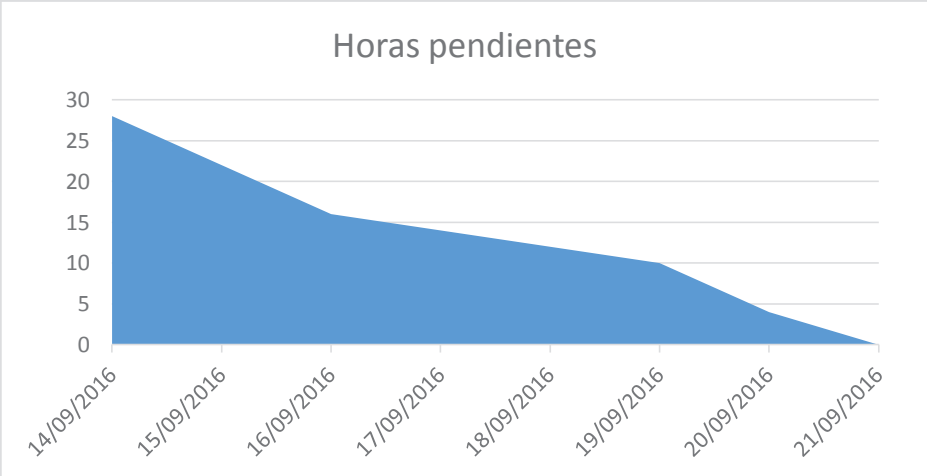


Figura 3.29 Esfuerzo: Sprint 3

Se emplearon 6 horas diarias para la realización del Sprint de tal manera que el último día ya no existían tareas por completar.

En la **Figura 3.30** se pueden apreciar las tareas ejecutadas durante el sprint, no fue posible ejecutar más de una tarea por día de trabajo.

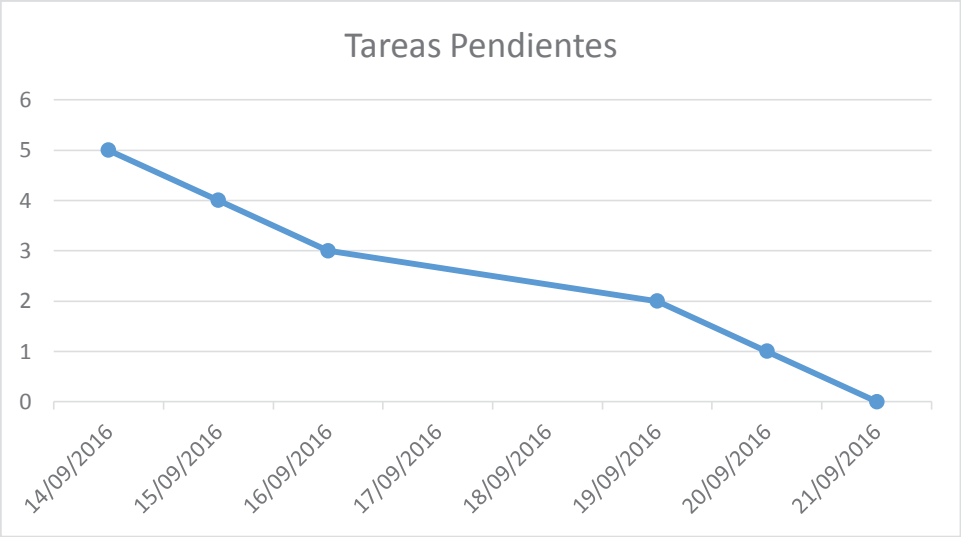


Figura 3.30 Tareas: Sprint 3

3.7.8.3.9 Incremento del Sprint

El resultado al finalizar el sprint 3 fue el esperado, teniendo un producto potencialmente "terminado" para comenzar con el sprint 3. No se realizan cambios de prioridad ni de requerimientos por lo que el *Product Backlog* se mantiene igual.

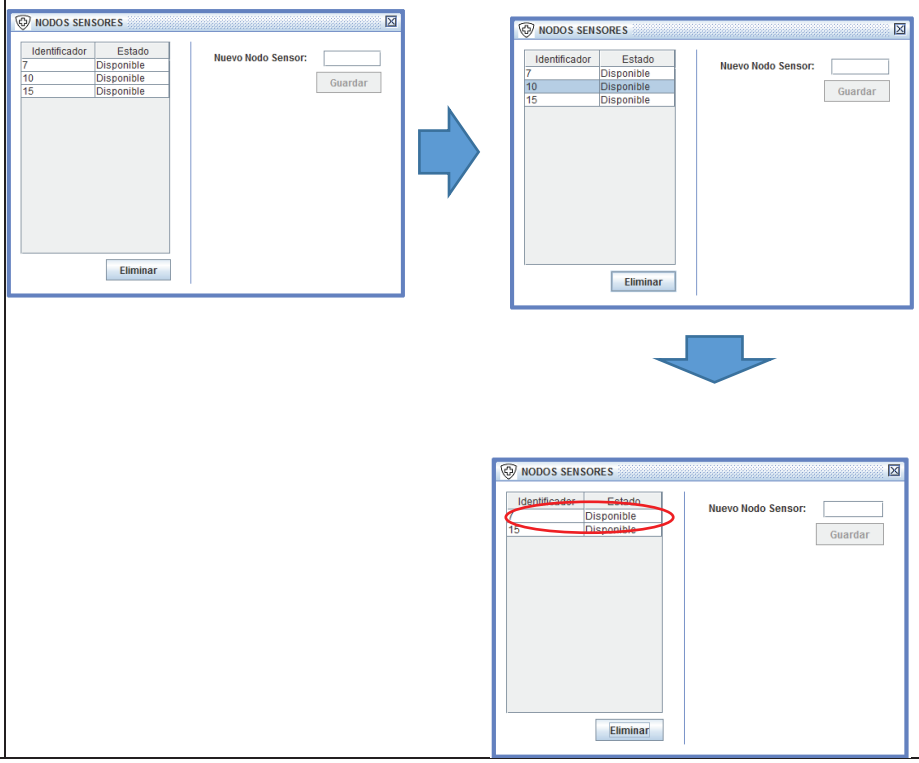
3.7.8.3.10 Pruebas de Aceptación

Para el ID número 2 constan las siguientes historias de usuario a las que deben ser aplicadas las pruebas de aceptación:

HU01-04 Visualizar el número de nodos sensores disponibles

En la **¡Error! La autoreferencia al marcador no es válida.** se observa la prueba de aceptación de HU01-04, como se observa los nodos sensores se almacenaban de uno a la vez, se observa que nodos están ocupados y cuáles no, y finalmente se puede eliminar cualquier nodo seleccionado que no se encuentre ocupado.

Tabla 3.31 PA1-04

PRUEBAS DE ACEPTACIÓN	
Número:	PA1-04
Escenario:	HU01-04 Visualizar el número de nodos sensores disponibles
Requisitos:	Seleccionar una historia clínica del jtable
Evento:	1. Guardar varios nodos 2. Seleccionar un nodo del jtable 3. Presionar el botón "Eliminar"
Resultado esperado:	1. Los nodos almacenados se muestran en el jtable 2. Se elimina el nodo del jtable
Evaluación:	Prueba exitosa.
Resultados:	

3.7.8.4 Sprint 4

Se implementará el formulario para visualizar los registros de la frecuencia cardíaca, métodos para buscar una historia clínica y descargar los registros como una imagen

.jpeg o .png. También se pretende crear el formulario principal de la aplicación y el formulario de ingreso.

3.7.8.4.1 Tareas del Sprint 4

En la **Tabla 3.32** se muestran las tareas del sprint 4.

Tabla 3.32 Tareas: Sprint 4

ID	Requerimiento	Tareas	Esfuerzo	Horas
3	Visualizar los registros de la frecuencia cardíaca en un gráfico de líneas.	Implementar formulario para visualizar el registro de la frecuencia cardíaca.	Alto	4
		Implementar métodos para buscar una historia clínica.	Alto	4
		Implementar métodos para mostrar los registros de la frecuencia cardíaca en un gráfico de líneas	Alto	10
4	Descargar registros de la frecuencia cardíaca en un gráfico de líneas.	Implementar métodos e interfaz para descargar registros de la frecuencia cardíaca en un gráfico de líneas en formato .png	Bajo	4
5	Crear la interfaz para el ingreso a la aplicación	Implementar métodos y formulario de menú principal	Alto	4
		Implementar métodos e interfaz para ingresar a la aplicación.	Alto	4

3.7.8.4.2 Scrum Diario

Se muestran las tareas realizadas y las tareas que aún faltan por hacer en caso de ser necesario.

Se debe procede a responder las siguientes preguntas:

- **¿Que hice ayer que ayudó al equipo de desarrollo a lograr el objetivo del Sprint?**

Se consiguieron los resultados del Sprint 3. No existen tareas pendientes por realizar.

- **¿Qué haré hoy para ayudar al equipo de desarrollo a lograr el objetivo del Sprint?**
  Comenzar a realizar las tareas del Sprint 4, se realizarán las consultas e investigaciones necesarias para cumplir con éxito los objetivos planteados

- **¿Veo algún impedimento que impida lograr el objetivo del Sprint?**
  No se observan ningún impedimento.

En la **Tabla 3.33** se aprecia el esfuerzo diario durante el cuarto y último Sprint.

Tabla 3.33 Esfuerzo diario: Sprint 4

SPRINT	INICIO	FIN
4	jueves, 22 de septiembre de 2016	jueves, 29 de septiembre de 2016

							jueves 22/09/2016	viernes 23/09/2016	lunes 26/09/2016	martes 27/09/2016	miércoles 28/09/2016	jueves 29/09/2016
					Tareas Pendientes		6	5	4	3	2	0
					Horas pendientes		30	24	18	12	6	0
ID	Requerimiento	Tareas	Responsable	Estado	Esfuerzo estimado	Horas	Esfuerzo					
3	Visualizar los registros de la frecuencia cardiaca en un gráfico de líneas.	Implementar formulario para visualizar el registro de la frecuencia cardiaca.	Byron	Finalizado	Alto	4	4					
		Implementar métodos para buscar una historia clínica.	Byron	Finalizado	Medio	4	4	2				
		Implementar métodos para mostrar los registros de la frecuencia cardiaca en un gráfico de líneas	Byron	Finalizado	Alto	10	10	10	6			
4	Descargar registros de la frecuencia cardiaca en un gráfico de líneas.	Implementar métodos e interfaz para descargar registros de la frecuencia cardiaca como imagen .png o .jpeg	Byron	Finalizado	Bajo	4	4	4	4	4		
5	Crear la interfaz para el ingreso a la aplicación	Implementar métodos y formulario de menú principal	Byron	Finalizado	Alto	4	4	4	4	4	2	
		Implementar métodos e interfaz para ingresar a la aplicación.	Byron	Finalizado	Alto	4	4	4	4	4	4	0

3.7.8.4.3 Revisión del Sprint 4

Se muestra cada una de las tareas ejecutadas durante el Sprint 4.

- **Implementación del formulario para visualizar los registros de la frecuencia cardíaca.**

La **Figura 3.31** muestra el formulario que permitirá al usuario visualizar los registros de la frecuencia cardíaca y descargarlos en caso de necesitarlo.

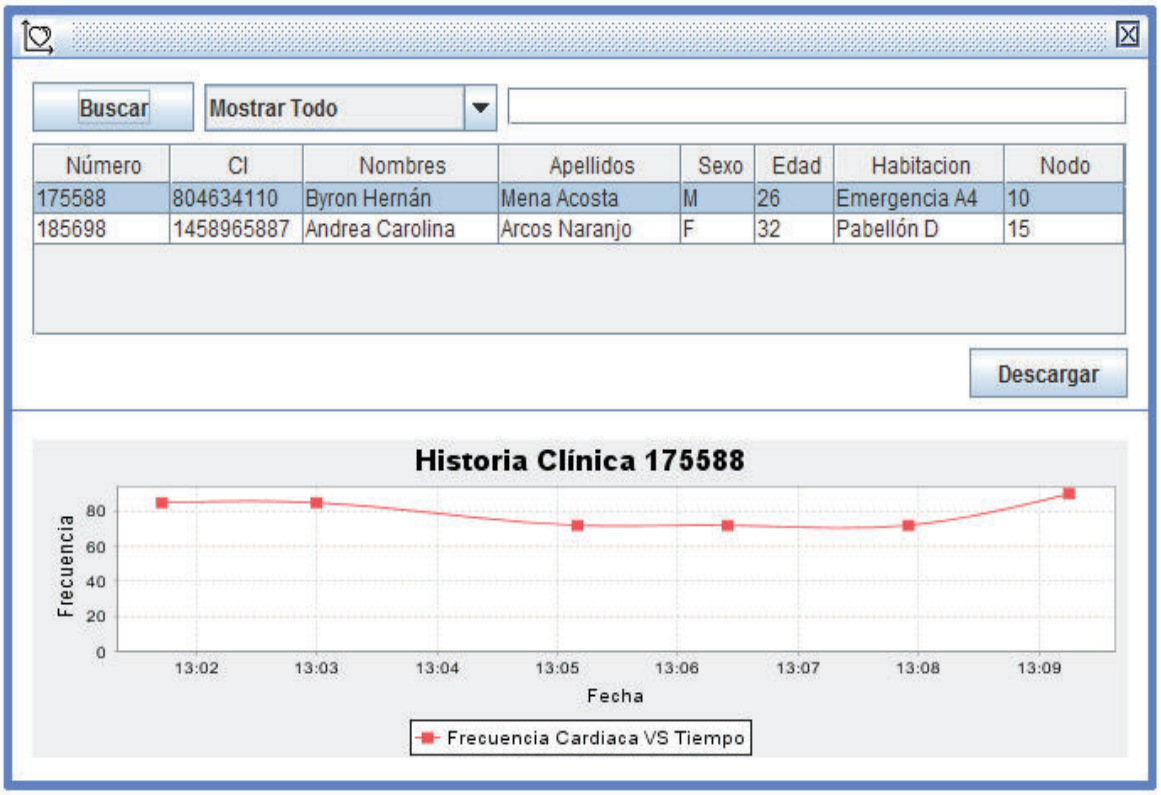


Figura 3.31 Formulario de registro de la frecuencia cardíaca

El formulario consta de un jcomboBox que indica el parámetro a buscar, un jtextField donde se indica el valor y un botón “Buscar” para iniciar la búsqueda y mostrar los resultados en un jtable.

Posee además un botón “Descargar” que permite guardar los registro en forma de imagen sea en formato .png o .jpeg y un “panel” donde se muestran los registros de la frecuencia cardíaca del paciente seleccionado del jtable.

3.7.8.4.4 Implementación de métodos para buscar una historia clínica

Se utilizó como base el código realizado en el Sprint 3 de búsqueda de historias clínicas.

3.7.8.4.5 Implementación de métodos para mostrar registros de la frecuencia cardíaca

Fue de gran ayuda la utilización de la librería jFreeChart. Una vez seleccionada la historia clínica del “jtable” se procede a su graficar en el “panel”

3.7.8.4.6 Implementación de métodos para descargar una historia clínica.

Con la ayuda de la librería jFreeChart se procedió a la captura de los datos que forman el gráfico de líneas para después guardarlos en formato .png o jpeg al presionar el botón guardar.

3.7.8.4.7 Implementar métodos y formulario del menú principal

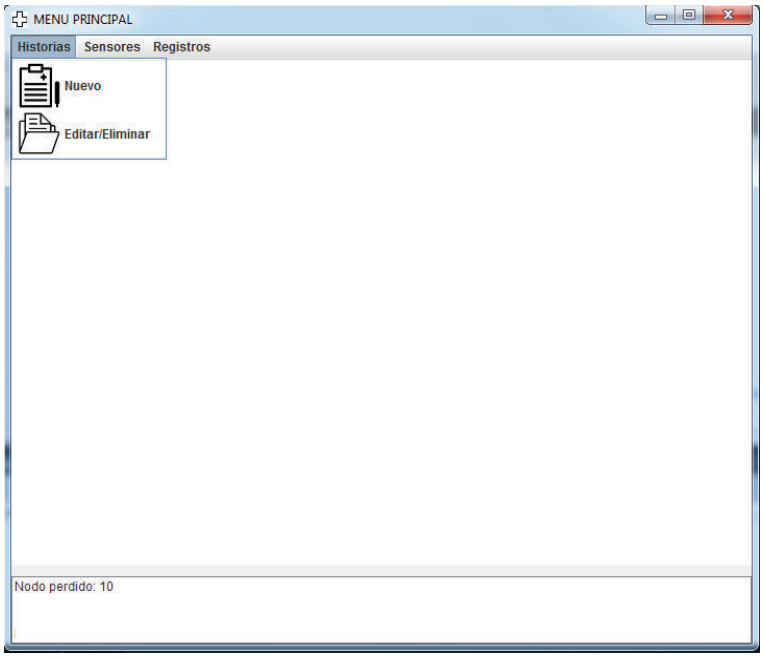


Figura 3.32 Menú Principal: Historias Clínicas

El menú consta de un formulario que contendrá subformularios, una barra de menú y una lista que indica si un nodo se ha perdido. El menú fue elaborado siguiendo el diseño de la navegación de la aplicación.

En la **Figura 3.32** se observa dos submenús; “Nuevo” y “Editar/Eliminar” que abrirán los formularios para crear una nueva historia clínica y el formulario para editar y eliminar historias clínicas respectivamente.

La **Figura 3.33** muestra el menú “Sensores” que contiene el submenú “Nuevo/Eliminar” para abrir el formulario de los nodos sensores.

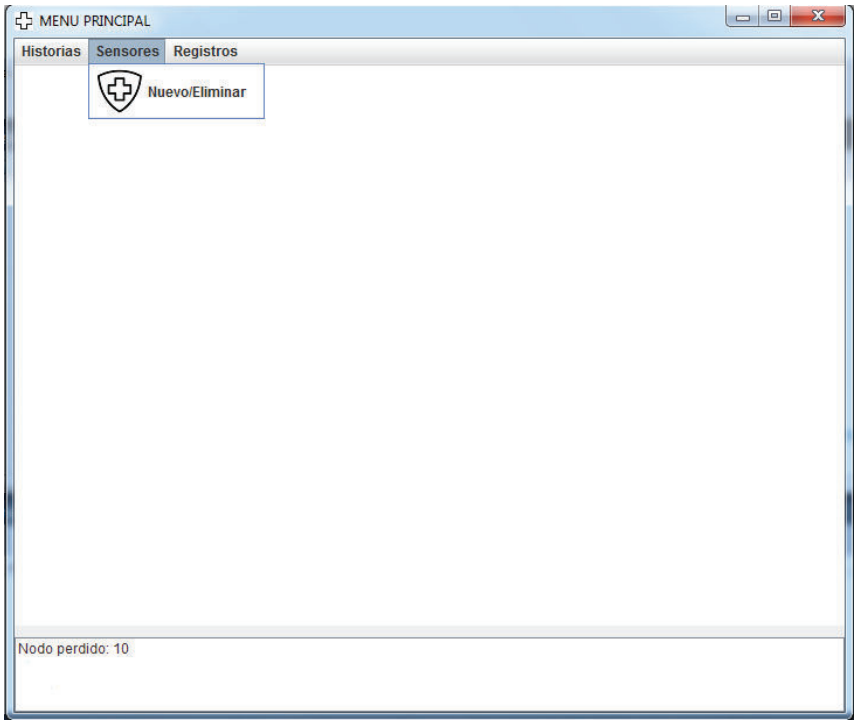


Figura 3.33 Menú Principal: Sensores

La **Figura 3.34** muestra el menú “Registros” que posee el submenú “Mostrar” encargado de abrir el formulario de “Registros”.

3.7.8.4.8 Implementar métodos e interfaz para ingresar a la aplicación

Para el ingreso de la aplicación se creó un formulario como el mostrado en **Figura 3.35**. Se debe ingresar el usuario y contraseña correspondientes a la base de datos y para obtener acceso. Se utilizó un archivo de texto que contiene la configuración de la base de datos incluido usuario y contraseña.

Al ingresar el usuario y contraseña los nuevos valores actualizan a los existentes en el archivo de configuración RedBDD.txt, el programa lee la información del archivo

actualizado e intenta comunicarse con la base de datos utilizando estos valores; si la comunicación se realiza el usuario ingresará a la aplicación caso contrario se muestra un mensaje de “usuario y/o contraseña incorrectos” y se niega el acceso al usuario, por lo tanto es necesario establecer primero la conexión con la base de datos antes de poder utilizar la aplicación.

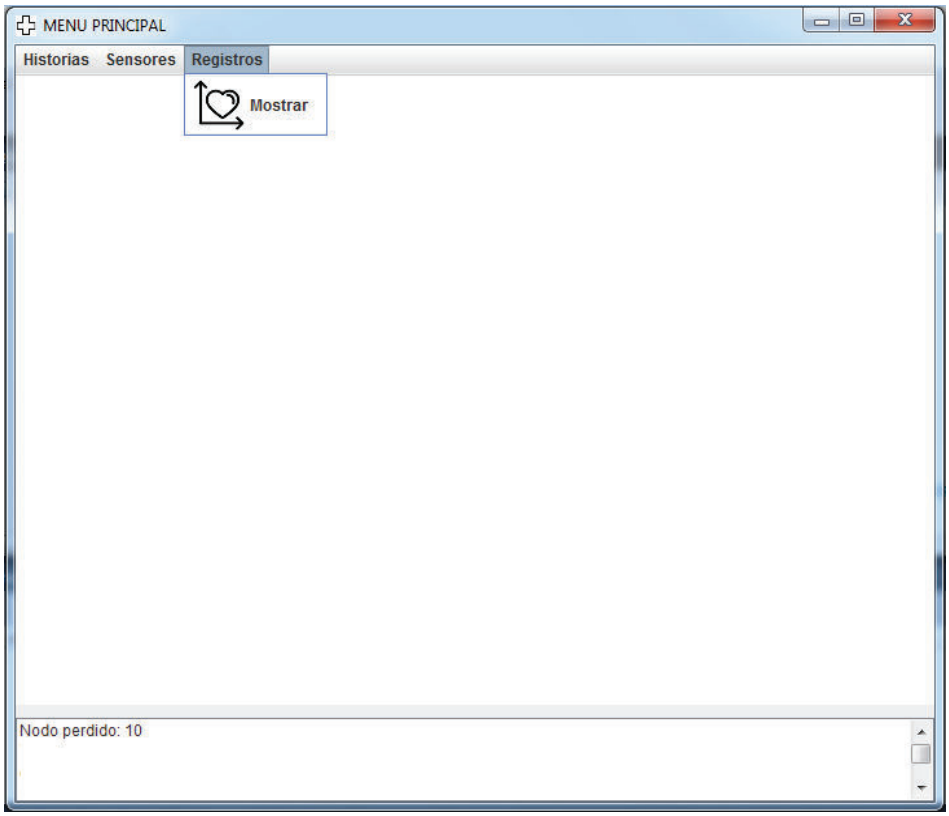


Figura 3.34 Menú Principal: Registros

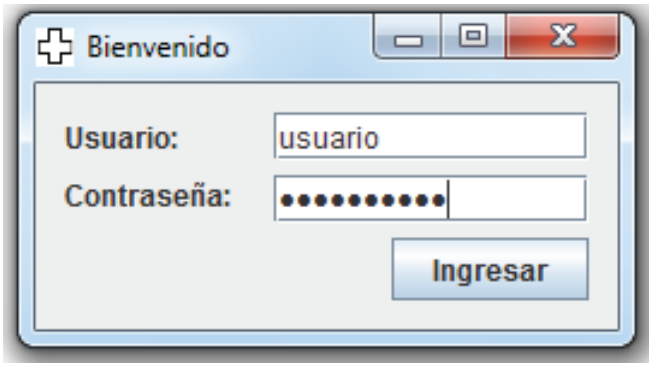


Figura 3.35 Ingreso a la aplicación

3.7.8.4.9 Retrospectiva del Sprint 4

En la **Figura 3.36** se puede apreciar el esfuerzo realizado para cumplir con las tareas del Sprint, finalizando las tareas en el tiempo establecido.

Se emplearon 6 horas diarias para la realización del Sprint de tal manera que el último día ya no existían tareas por completar.

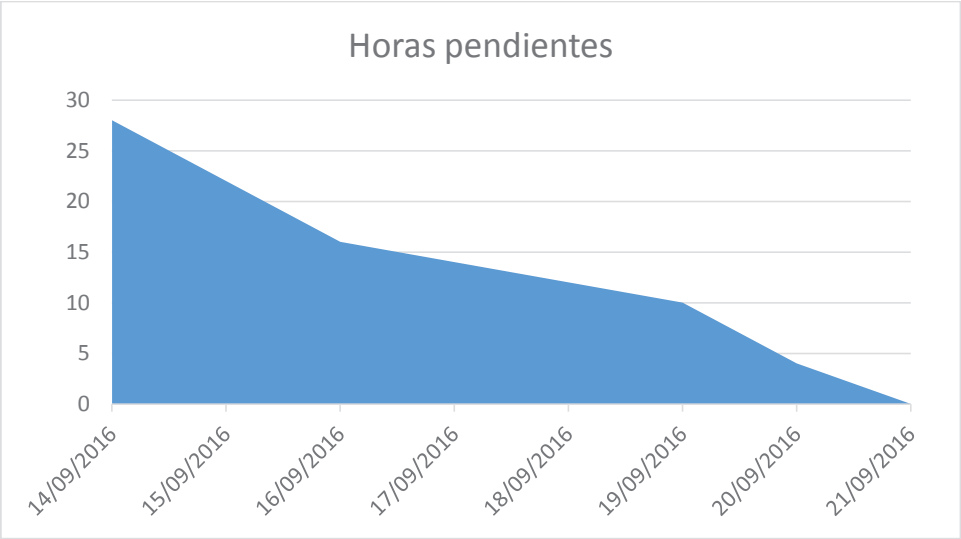


Figura 3.36 Esfuerzo: Sprint 3

En la **Figura 3.37** se pueden apreciar las tareas ejecutadas durante el sprint, no fue posible ejecutar más de una tarea por día de trabajo.

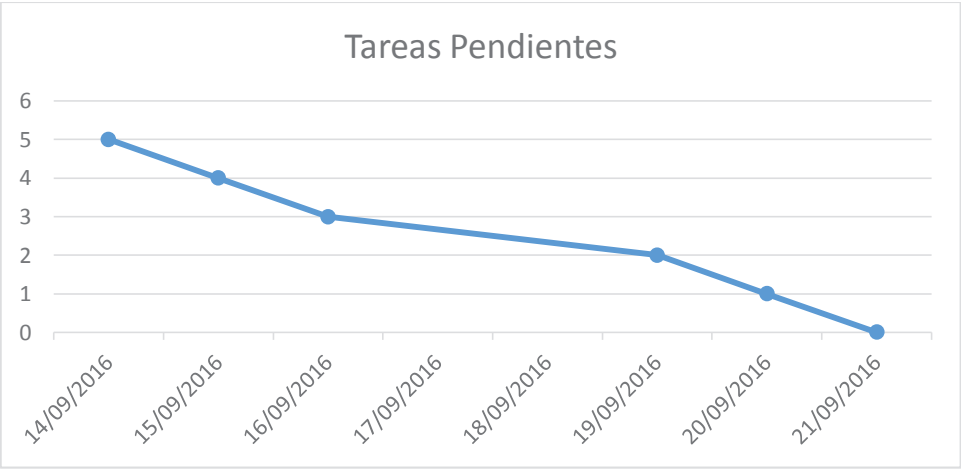


Figura 3.37 Tareas: Sprint 3

3.7.8.4.10 Incremento del Sprint

El resultado al finalizar el sprint 4 fue el esperado, teniendo un producto “terminado”.

3.7.8.4.11 Pruebas de Aceptación

Las siguientes historias de usuario a las que deben ser aplicadas las pruebas de aceptación son:

HC02-01	Visualizar los registros de la frecuencia cardíaca en un gráfico de líneas.
HC03-01	Descargar registros de la frecuencia cardíaca en un gráfico de líneas.

Con esto concluye el desarrollo de la aplicación utilizando la metodología Scrum, se pudo comprobar que la ventaja que brinda al permitir realizar cambios inmediatos en casos de observaciones generadas por el cliente o el usuario final del producto, el código correspondiente a esta aplicación se encontrara en el Anexo D.

Como se mencionó en la parte de diseño se implementó el patrón modelo, vista, controlador con el fin de separar la lógica de la aplicación del diseño y del acceso a la base de datos, brindado un orden al desarrollo de la aplicación.

En la **Tabla 3.36** se observa las clases creadas como parte del Modelo del patrón, utilizadas para las conexiones con la base de datos.

En la **Tabla 3.34** se observa la prueba de aceptación realizada para HU02-01 y que constituye el objetivo del presente trabajo de titulación de presentar la frecuencia cardíaca de una manera gráfica, fácil y entendible.

La información para la generación de las gráficas proviene directamente de la base de datos y se actualiza cada vez que se ingresa a ella.

La opción de poder descargar las gráficas generadas constituye un aspecto llamativo y de utilidad para poder hacer pronósticos o la generación de reportes, las extensiones elegidas fueron .jpeg y .png.

Tabla 3.34 PA2-01

PRUEBAS DE ACEPTACIÓN	
Número:	PA2-01
Escenario:	HU02-01 Visualizar los registros de la frecuencia cardíaca en un gráfico de líneas.
Requisitos:	Registrar valores de frecuencia cardíaca en la base de datos
Evento:	Seleccionar una historia clínica
Resultado esperado:	En el panel se despliega un gráfico de líneas correspondiente a la "frecuencia cardíaca vs tiempo"
Evaluación:	Prueba exitosa.
Resultado:	

En la **Tabla 3.37** están las clases que conforman la parte Vista del patrón, está conformada por los formularios presentados anteriormente, la nomenclatura "JFnombre" indica que se trata de un formulario, mientras que "JInombre" indica que se trata de un formulario interno.

Tabla 3.35 PA3-01

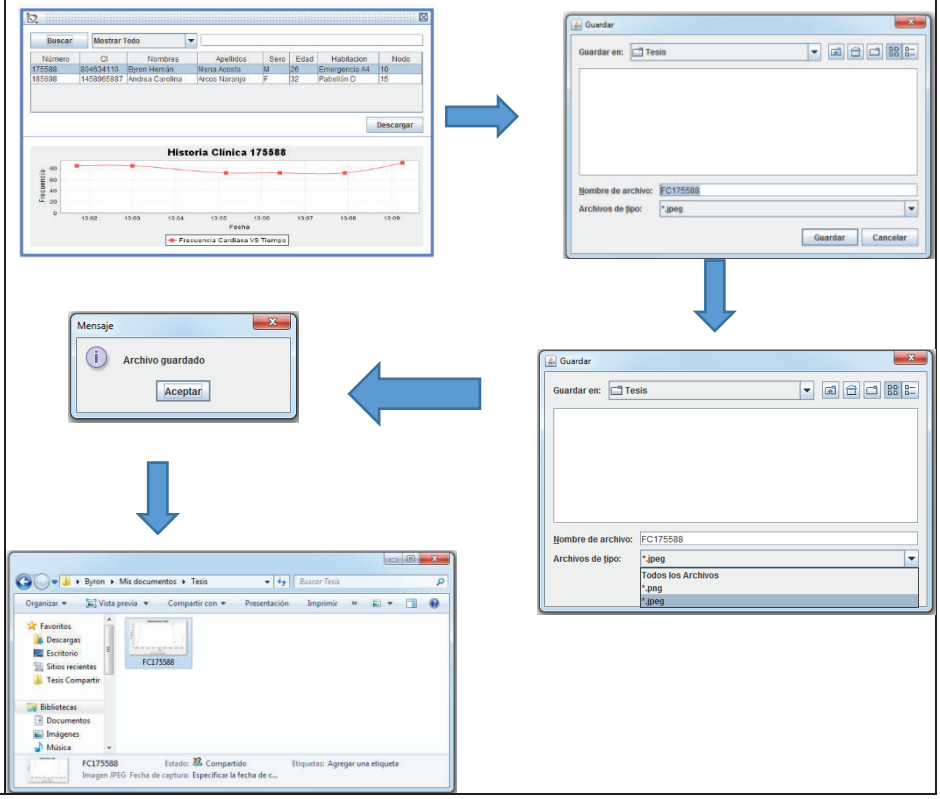
PRUEBAS DE ACEPTACIÓN	
Número:	PA3-01
Escenario:	HU03-01 Descargar registros de la frecuencia cardíaca en un gráfico de líneas.
Requisitos:	Registrar valores de frecuencia cardíaca en la base de datos
Evento:	1. Seleccionar una historia clínica y presionar el botón “Descargar” 2. Seleccionar un nombre y ubicación para guardar el archivo 3. Seleccionar el formato (.png o .jpeg) del archivo 4. Presionar el botón “Guardar”
Resultado esperado:	1. El archivo se almaceno en la computadora en formato
Evaluación:	Prueba exitosa.
Resultados	

Tabla 3.36 Patrón: Modelo

Tabla de Base de Datos	Clase	Descripción
Historias_Clínicas	modeloHistoriasClínicas	Es un representación de la Historia Clínica
	modeloHistoriasClínicasDAO	Posee todos los métodos encargados de crear, modificar y eliminar historias clínicas en la base de datos.
Nodos_Sensores	modeloNodosSensores	Es una representación de Nodos Sensores
	modeloNodosSensoresDAO	Posee todos los métodos encargados de crear y eliminar nodos sensores de la base de datos.
FC{n}	modeloTablasFC	Es una representación de las tablas FC que almacenan la frecuencia cardíaca.
	modeloTablasFCDAO	Posee los métodos encargados del ingreso y lectura de los valores de frecuencia cardíaca
	modeloConexión	Clase que establece conexión con la base de datos y permite que las demás clases realicen su funciones.

Tabla 3.37 Patrón: Vista

Clase	Descripción
JFAplicación	Lo conforma el menú principal
JFIngreso	Corresponde al formulario de ingreso o *Login*
JIEditarEliminar	Corresponde al formulario encargado de Editar y Eliminar Historias Clínicas
JINuevaHistoriaClínica	Corresponde al formulario encargado de crear una nueva historia clínica
JIRegistros	Corresponde al formulario encargado de mostrar los registros

En la **Tabla 3.38** se muestran las clases que conforman la parte Controlador del patrón, aquí es donde se concentra la lógica de la aplicación y se puede considerar la parte más importante de la aplicación.

Cada una de las clases que conforman Vista o Modelo permiten la comunicación con el usuario final de la aplicación o la base de datos, respectivamente, para el caso de la aplicación Tiempo Real utilizada para el desarrollo del prototipo, Modelo se encarga de comunicarse con la clase encargada de recibir los datos por el puerto serial desde el nodo Gateway.

Controlador es el que mayor tiempo requerirá para desarrollarlo, debido a que debe controlar lo que sucede tanto en Vista como en Modelo, y un pequeño cambio puede alterar gravemente el funcionamiento de toda la aplicación.

Como parte de la mini aplicación encargada de mostrar en tiempo real los datos generados por el nodo sensor se elaboraron las clases mostradas en la **Tabla 3.39**.

Esta aplicación se encuentra en el menú “Registros>Tiempo Real” de la aplicación. Esta aplicación no permite trabajar solo con un nodo a la vez, fue probada con nodos sensores que enviaban datos cada 10 milisegundos sin que esto interfiera en el normal funcionamiento de la aplicación.

Tabla 3.38 Patrón: Controlador

Clase	Vista	Descripción
controladorInicio	JAplicación	Se encarga de controlar la apertura del formulario principal y de los formularios internos.
controladorLogin	JFIngreso	Se encarga de controlar el ingreso y permitir que se ejecute el controladorInicio
controladorEditarEliminar	JIEditarEliminar	Se encarga de controlar el formulario que permite editar y eliminar historias clínicas
controladorNuevaHistoriaClínica	JINuevaHistoriaClínica	Se encarga de controlar el formulario encargado de crear una historia clínica
controladorNodosSensores	JISensores	Se encarga de controlar el formulario que gestiona los nodos sensores
controladorRegistros	JIRegistros	Se encarga de controlar el formulario que muestra los registros de frecuencia cardíaca

Tabla 3.39 Aplicación de Tiempo Real

Componente	Clase	Descripción
Modelo	modeloTiempoReal	Se encarga de recibir los datos del computador Gateway
	modeloRedBDD	Se encarga de obtener la configuración de Red del archivo RedBDD.txt
Vista	JITiempoReal	Formulario donde se muestra la gráfica en tiempo real
Controlador	controladorTiempoReal	Se encarga de controlar el formulario JITiempoReal
	GráficaEnTiempoReal	Elabora una gráfica de manera periódica a base de la clase Gráfica
	Gráfica	Es la encargada de generar la gráfica con los valores suministrados por HiloValoresY
	HiloValoresY	Esta clase es un hilo[56] encargado de la escucha permanente de mensajes provenientes del computador Gateway

3.8 PRUEBAS

Para el correcto funcionamiento del prototipo a más de las pruebas ejecutadas para el desarrollo de la aplicación y base de datos, se realizaron pruebas para verificar el correcto funcionamiento del prototipo terminado.

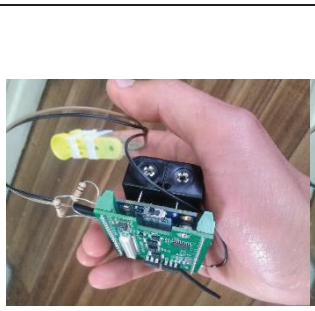
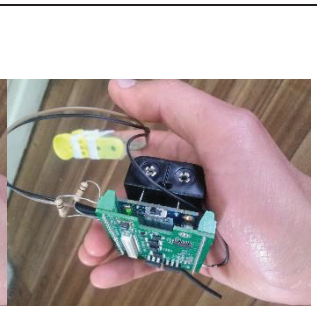
[56] **Hilo:** Segmento de código que permite ejecutar múltiples tareas

El formato de pruebas es similar al elaborado durante la aplicación de SCRUM pero con la diferencia que en este caso ya no existe referencia a historias de usuario.

3.8.1 CONEXIÓN SENSOR CARDIACO CON NODO SENSOR

En la **Tabla 3.40** se muestra las pruebas de la conexión entre el sensor y el nodo.

Tabla 3.40 Pruebas nodo sensor

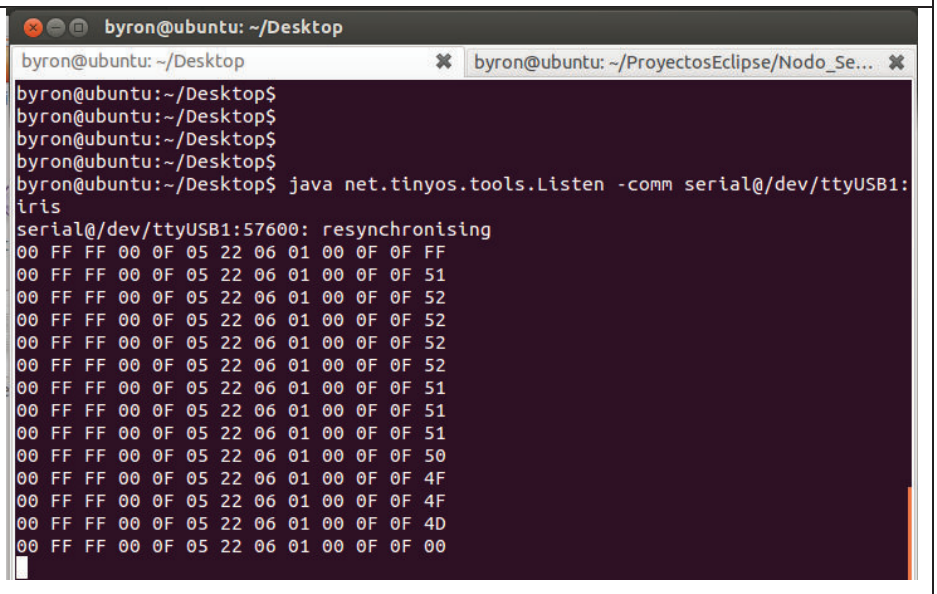
PRUEBAS NODO SENSOR	
Número:	1
Escenario:	Visualizar los pulsos en el nodo sensor
Requisitos:	Conexión del nodo sensor y el sensor cardiaco a la placa MDA300CA. Programa un LED para prenderse y apagarse ante cambios de voltaje en su entrada A0.
Evento:	1. Encendido el nodo sensor colocar el dedo de un voluntario por su parte anterior sobre el sensor de pulso. 2. Colocar los dedos índice, medio y anular en el voluntario sobre la arterial radial[57] del brazo en el que se encuentra el nodo sensor.
Resultado esperado:	El LED se prende y apaga sincrónicamente con los latidos en la arteria radial.
Evaluación:	Prueba exitosa.
Resultados:	  

[57] **Arteria radial:** Arteria del antebrazo que se origina como rama de bifurcación externa de la arteria humeral.

3.8.2 CONEXIÓN NODO SENSOR CON NODO GATEWAY

En la **Tabla 3.41** se muestra la prueba y resultados de la conexión entre el nodo sensor y el nodo Gateway.

Tabla 3.41 Pruebas de conectividad IEEE 802.15.4 y nodo Gateway

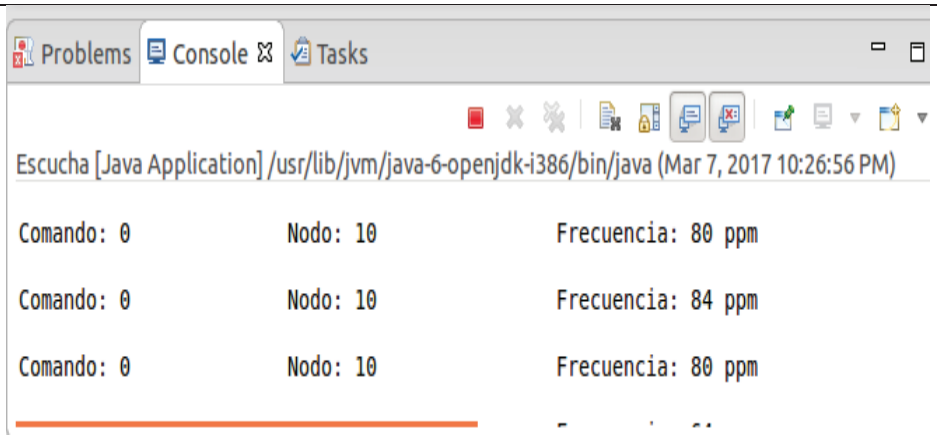
PRUEBAS DE CONECTIVIDAD IEEE802.15.4 Y NODO GATEWAY	
Número:	2
Escenario:	Visualizar tramas IEEE 802.15.4
Requisitos:	Nodo Sensor enviando identificador y frecuencia cardíaca Nodo Gateway conectado a una computadora. Ejecución de java net.tinyos.tools.*Listen*
Evento:	1. El nodo Gateway conectado a la computadora y un terminal abierto que ha ejecutado java net.tinyos.tools.*Listen* esperando el encendido del nodo sensor.
Resultado esperado:	1. Al encender el nodo sensor se visualiza en el terminal de la computadora la trama IEEE 802.15.4.
Evaluación:	Prueba exitosa.
Resultados	

3.8.3 CONEXIÓN DEL NODO SENSOR Y EL COMPUTADOR GATEWAY

En la **Tabla 3.42** se muestra las pruebas de la comunicación entre el nodo sensor y el nodo Gateway.

Para esto fue necesario cargar en un nodo sensor un código que sea capaz de leer la información recibida por una de las entradas analógicas de la placa MDA300CA las señales provenientes del nodo sensor, calcular la frecuencia cardíaca y enviarlas al nodo Gateway de manera periódica.

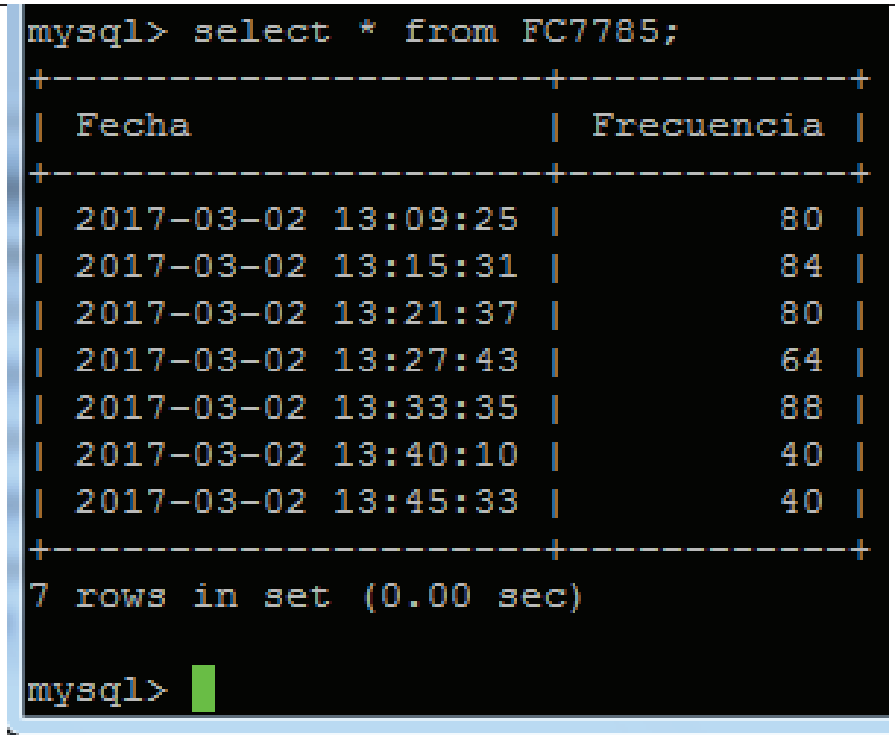
Tabla 3.42 Pruebas de conexión nodo sensor - Gateway

PRUEBAS DE CONEXIÓN NODO SENSOR - GATEWAY	
Número:	3
Escenario:	Mostrar identificador y frecuencia cardíaca
Requisitos:	Nodo Sensor enviando paquetes. Nodo Gateway conectado a una computadora. Ejecución de aplicación de Gateway.
Evento:	1. El nodo sensor envía datos.
Resultado esperado:	1. Se visualiza en la consola de la aplicación el identificador del nodo y su valor de frecuencia cardíaca en formato decimal.
Evaluación:	Prueba exitosa.
Resultados:	

3.8.4 CONEXIÓN COMPUTADOR GATEWAY A BASE DE DATOS

En la **Tabla 3.43** se observa la prueba realizada que demuestra la correcta conexión entre el nodo Gateway y la base de datos.

Tabla 3.43 Prueba computador Gateway y Base de Datos

PRUEBAS COMPUTADOR GATEWAY Y BASE DE DATOS	
Número:	4
Escenario:	Almacenamiento de información en la base de datos
Requisitos:	Aplicación de la computadora Gateway recibiendo información Historia clínica creada con un nodo sensor asignado
Evento:	1. Se recibe en el Gateway un mensaje del nodo sensor
Resultado esperado:	1. Al recibir un mensaje el computador Gateway almacena esta información en la tabla de Frecuencia Cardíaca asignada a la historia clínica en la base de datos.
Evaluación:	Prueba exitosa.
Resultados:	

```
mysql> select * from FC7785;
+---------------------+------------+
| Fecha               | Frecuencia |
+---------------------+------------+
| 2017-03-02 13:09:25 |         80 |
| 2017-03-02 13:15:31 |         84 |
| 2017-03-02 13:21:37 |         80 |
| 2017-03-02 13:27:43 |         64 |
| 2017-03-02 13:33:35 |         88 |
| 2017-03-02 13:40:10 |         40 |
| 2017-03-02 13:45:33 |         40 |
+---------------------+------------+
7 rows in set (0.00 sec)

mysql>
```

3.8.5 PRUEBA DEL PROTOTIPO

En la **Tabla 3.44** se observa la prueba del prototipo y una medición realizada utilizando el pulso radial, mientras que en la **Tabla 3.45** y **Tabla 3.46** se compara el prototipo con el celular *S6 edge*.

Tabla 3.44 Prueba del prototipo

PRUEBA DEL PROTOTIPO	
Número:	5
Escenario:	Prueba del prototipo
Requisitos:	Nodo sensor encendido y conectado a una persona postrada en cama. Conocer donde se encuentra pulso radial o carotideo durante 60.
Evento:	Se obtienen 5 medidas de la frecuencia cardíaca en el voluntario con el prototipo para obtener un promedio, cerca de finalizar la prueba se mide la frecuencia cardíaca utilizando el pulso radial y carotideo durante 60 segundos
Resultado esperado:	El promedio de los valores de la frecuencia cardíaca obtenido con el prototipo respecto al valor utilizando el pulso radial no difiere en un porcentaje mayor al 10%
Evaluación:	Prueba exitosa.
Resultados:	Valores obtenidos con el prototipo Comando: 0 Nodo: 10 Frecuencia: 80 ppm Comando: 0 Nodo: 10 Frecuencia: 84 ppm Comando: 0 Nodo: 10 Frecuencia: 80 ppm Comando: 0 Nodo: 10 Frecuencia: 64 ppm Comando: 0 Nodo: 10 Frecuencia: 88 ppm $\text{Frecuencia}_{\text{Prototipo}} = 79{,}2 \text{ latidos por minuto}$ $\text{Frecuencia}_{\text{Radial}} = 81 \text{ latidos por minutos}$ $\text{Variación} = \frac{\text{Frecuencia}_{\text{Radial}} - \text{Frecuencia}_{\text{Prototipo}}}{\text{Frecuencia}_{\text{Radial}}}$ $\text{Variación} = 2.22\%$

Tabla 3.45 Prueba prototipo y celular *S6 edge* (Parte I)

<table>
<tr><th colspan="2">PRUEBA PROTOTIPO Y CELULAR S6 EDGE</th></tr>
<tr><td>Número:</td><td>6</td></tr>
<tr><td>Escenario:</td><td>Prueba del prototipo y celular S6 edge</td></tr>
<tr><td>Requisitos:</td><td>Nodo sensor encendido y conectado a una persona postrada en cama.
Sensor de pulso de celular S6 edge</td></tr>
<tr><td>Evento:</td><td>Se obtienen 5 medidas de la frecuencia cardíaca en el voluntario con el prototipo para obtener un promedio, cerca de finalizar la prueba se mide la frecuencia cardíaca utilizando el sensor de pulso incorporado en el equipo celular S6 edge</td></tr>
<tr><td>Resultado esperado:</td><td>El valor de la frecuencia cardíaca obtenido con el prototipo respecto al valor del sensor comercial no debe diferir en un porcentaje mayor al 10%</td></tr>
<tr><td>Evaluación:</td><td>Prueba exitosa.</td></tr>
<tr><td>Resultados:</td><td>Valores obtenidos con el prototipo
Comando: 0 Nodo: 10 Frecuencia: 80 ppm
Comando: 0 Nodo: 10 Frecuencia: 84 ppm
Comando: 0 Nodo: 10 Frecuencia: 80 ppm
Comando: 0 Nodo: 10 Frecuencia: 64 ppm
Comando: 0 Nodo: 10 Frecuencia: 88 ppm
Comando: 0 Nodo: 10 Frecuencia: 80 ppm
Comando: 0 Nodo: 10 Frecuencia: 84 ppm
Comando: 0 Nodo: 10 Frecuencia: 80 ppm
Comando: 0 Nodo: 10 Frecuencia: 64 ppm
Comando: 0 Nodo: 10 Frecuencia: 88 ppm
Valores obtenido con el equipo celular S6 edge
<table>
<tr><th>Muestras</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th></tr>
<tr><td>Frecuencia cardíaca [ppm]</td><td>77</td><td>75</td><td>80</td><td>76</td><td>80</td><td>76</td><td>75</td><td>77</td><td>80</td><td>78</td></tr>
</table>
</td></tr>
</table>

Tabla 3.46 Prueba prototipo y celular *S6 edge* (Parte II)

PRUEBA PROTOTIPO Y CELULAR S6 EDGE	
Resultados:	

$$\text{Variación} = \frac{\overline{\text{Frecuencia}_{S6}} - \overline{\text{Frecuencia}_{\text{Prototipo}}}}{\overline{\text{Frecuencia}_{S6}}}$$

$$\text{Variación} = \frac{77.4 - 79.2}{77.4}$$

$$\text{Variación} = -2.02\%$$

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

En este apartado se muestran las conclusiones del desarrollo y pruebas realizadas del trabajo de titulación así como recomendaciones del mismo.

4.1 CONCLUSIONES

- El continuo desarrollo e investigación con tecnologías IEEE802.15.4 permitirá en un futuro tener dispositivos económicos y fácilmente administrables por usuarios sin necesidad de tener un conocimiento profundo de este tipo de redes.
- Al trabajar con métodos no invasivos como es el caso del sensor de pulso utilizado en este trabajo de titulación se corre el riesgo de que un movimiento brusco altere los resultados.
- Este trabajo de titulación abre una ventana al desarrollo de aplicaciones de sistemas de monitoreo ambulante económicos y con mejores características.
- Para ampliar el número de usuarios en el sistema, solamente es necesario colocar un nuevo nodo Gateway, mas no el adquirir otra base de datos.
- El prototipo posee la capacidad para implementar un sistema completo de electrocardiografía con 12 derivaciones, sería un dispositivo fácil de transportar en la maleta de un médico.
- A pesar de que las tecnologías *bluetooth* e IEEE802.15.4 pueden ser implementadas para las mismas aplicaciones sin mucha diferencia en sus costos de instalación, la ventaja se puede observar en IEEE802.15.4 al requerir un menor número de baterías y realizar una mejor utilización del espectro.
- La información que puede solucionar un problema puede provenir de cualquier parte, desde una prestigiosa revista, un foro no oficial o una conversación con amigos.

4.2 RECOMENDACIONES

- Se recomienda en caso de utilizar un dedo del paciente, no utilizar el brazo dominante para minimizar la cantidad de movimientos que realiza el paciente.
- Se recomienda la implementación de distintos sensores utilizados en el campo médico como un saturador de oxígeno, temperatura corporal y presión arterial.
- De ser posible trabajar con métodos invasivos la implementación de un medidor de glucosa sería de importancia para el monitoreo de pacientes diabéticos, una enfermedad creciente en nuestro medio de acuerdo al Ministerio se Salud.
- El uso cada vez mayor de dispositivos móviles que utilizan los sistemas operativos Android e IOS brindarían una mayor flexibilidad al médico para controlar a los pacientes que tiene a su cargo.
- Un sistema de alarma puede ser considerado para el desarrollo de un futuro trabajo de titulación, esto sería de principal utilidad para pacientes con enfermedades crónicas (diabetes, hipertensión), permitiendo controlar con mayor precisión los factores que influyen en el desarrollo de estos padecimientos entre ellos los hábitos (ejercicio, alimentación) y/o medicamentos.
- Algunas características como un diseño cómodo para el usuario, una cubierta que proteja los elementos y un diseño atractivo comercialmente no fueron considerados y pueden ser desarrollados en futuros proyectos.

REFERENCIAS BIBLIOGRÁFICAS

[1] A. Guyton and J. Hall, “Guyton y Hall Tratado de Fisiología Médica,” in ELSEVIER, vol. 12, 2010, p. 1092.

[2] P. M. Candioti Busaniche Matias, Echagüe Jose Manuel, “Laboratorio de Habilidades Médicas: SIGNOS VITALES,” Univ. Nac. del Litoral, p. 19, 2011.

[3] H. E. Jhon, “Tratado de Fisiología Médica,” in El corazón como bomba, 13th ed., Barcelona, España: Elsevier, 2016, p. 1092.

[4] L. Tytgat, O. Yaron, S. Pollin, I. Moerman, and P. Demeester, “Avoiding collisions between IEEE 802.11 and IEEE 802.15.4 through coexistence aware clear channel assessment,” EURASIP J. Wirel. Commun. Netw., vol. 2012, no. 1, p. 137, 2012.

[5] SMC Networks Asia Pacific Pte Ltd., “Networking Basics,” Singapore, 2011.

[6] P. Committee, “IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN - Specific Requirements,” in IEEE Std 802.15.1-2002, IEEE Compu., no. June, The Institute of Electrical and Electronics Engineers, Inc., 2002, pp. 1–70.

[7] A. Elahi and A. Gschwender, "ZigBee Wireless Sensor and Control Network". Boston: Pearson Prentice Hall, 2010.

[8] Freescale semiconductor, "Long Term Evolution Overview", 1st ed. Denver, Colorado, 2008.

[9] V. K. Gagner "Wireless Communications & Networking". Elsevier, 2007.

[10] C. Buratti, M. Martalò, G. Ferrari, and R. Verdone, "Sensor Networks with IEEE 802.15.4" Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[11] Z. Shelby and C. Bormann, 6LoWPAN: the wireless embedded internet, vol. 43. 2011.

[12] S. R. Caprile, “Desarrollo de aplicaciones con comunicación remota basadas

en módulos ZigBee y 802.15.4," Boston, 2013, p. 362.

[13] S. Basagni, M. Conti, S. Giordano, I. Stojmenovic, and I. Stojmenovi, "Mobile ad hoc networking", vol. 10. 2004.

[14] M. Blvd, "XMesh User ' s Manual," no. March, p. 130, 2012.

[15] MEMSIC, "Mote Processor Radio & Mote Interface Boards User Manual," 2010, p. 39.

[16] F. Ficarola, "[Tinyos-help] MDA300CA Board with TinyOS 2.1.1," 2012. [Online]. Available: http://mail.millennium.berkeley.edu/pipermail/tinyos-help/2012-July/055198.html. [Accessed: 29-Oct-2016].

[17] Crossbow Technology, MTS/MDA Sensor and Data Acquisition Boards User's Manual. San José, California, 2004.

[18] J. Murphy and Y. Gitman, "PulseSensor," Open Hardware, 2016. [Online]. Available: http://pulsesensor.com/pages/open-hardware. [Accessed: 23-Nov-2016].

[19] P. Levis and D. Gay, "TinyOS Programming," Cambridge Univ. Press, vol. 1st. Ed., pp. 21–36, 2009.

[20] U. de Colorado, "mig - message interface generator for nesC," mig - message interface generator for nesC. [Online]. Available: http://cs.uccs.edu/~cs526/mote/nesc/doc/mig.html. [Accessed: 15-Sep-2016].

[21] Standford University, "Installing TinyOS 2.1.1," 2013. [Online]. Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/Installing_TinyOS_2.1.1. [Accessed: 02-Aug-2016].

[22] T. Instruments, "GCC - Open Source Compiler for MSP ." [Online]. Available: http://www.ti.com/tool/MSP430-GCC-OPENSOURCE. [Accessed: 22-Sep-2016].

[23] L. Arnedo, J. Gama, and G. Masramon Prat, "Como funciona SCRUM."

[Online]. Available: https://proyectosagiles.org/como-funciona-scrum/. [Accessed: 23-Aug-2016].

[24] L. Guía, D. De Scrum, and L. Reglas, “La Guía de Scrum,” *Creat. Commons*, p. 21, 2013.

[25] M. Rahimi, “Mica2 data acquisition board,” *Cent. Embed. Netw. Sensing, UCLA*, pp. 1–13, 2003.

[26] Apple, “Tu frecuencia cardíaca. Qué es y dónde puedes verla en el Apple Watch.,” 2016. [Online]. Available: https://support.apple.com/es-es/HT204666. [Accessed: 23-Aug-2016].

[27] P. G. Llanio R., “Exploración del sistema circulatorio. región precordial,” *Propedéutica Clin. y Semiología Médica Tomo 1*, pp. 120–132, 2003.

[28] R. Pichler, *Management with Scrum*, 1st ed. Stoughton, Massachusetts: Boston, 2010.

ANEXOS

ANEXO A: MANUAL DE INSTALACIÓN DE FTDI PARA LINUX

ANEXO B: CÓDIGO DE LOS NODOS SENSORES

ANEXO C: CÓDIGO DE LA APLICACIÓN

Todos los anexos se incluyen en el CD adjunto.