EVALUACIÓN Y APLICACIÓN DE LA TECNOLOGÍA IEEE 802.15.4 EN EL TRANSPORTE DEL RITMO CARDÍACO COMO ENTRADA PARA UN SISTEMA DE COMUNICACIÓN AUTOMÁTICA
ANEXOS

JOSÉ DARÍO ALEGRÍA JIMÉNEZ
OSCAR GILDARDO MUÑOZ MORALES

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
GRUPO DE NUEVAS TECNOLOGÍAS EN TELECOMUNICACIONES
POPAYÁN
2006

EVALUACIÓN Y APLICACIÓN DE LA TECNOLOGÍA IEEE 802.15.4 EN EL TRANSPORTE DEL RITMO CARDÍACO COMO ENTRADA PARA UN SISTEMA DE COMUNICACIÓN AUTOMÁTICA
ANEXO

JOSÉ DARÍO ALEGRÍA JIMÉNEZ
OSCAR GILDARDO MUÑOZ MORALES

Monografía para optar al título de
Ingeniero en Electrónica y Telecomunicaciones

Director
Ing. Esp. GUEFRY LEIDER AGREDO MENDEZ

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
GRUPO DE NUEVAS TECNOLOGÍAS EN TELECOMUNICACIONES
POPAYÁN
2006

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

ANEXO A - ARRITMIAS

El paciente se encuentra a menudo en un ambiente clínico y fisiológico cambiante. La selección e interpretación de los parámetros a monitorizar, son de utilidad solamente cuando van asociados a un razonamiento clínico de la condición del paciente, basado en los elementos de la historia clínica, examen físico y otros diagnósticos auxiliares.

La monitorización del paciente tiene tres propósitos básicos:

- Alertar: Según la condición del paciente y el nivel de monitorización, le avisa a la persona indicada cualquier deterioro en la función medida.
- Diagnóstico Continúo: Permite observar el comportamiento y cambios del paciente en una condición determinada.
- Pronóstico: La observación de las tendencias en los parámetros observados en la evolución, ayuda a establecer pronóstico.

Dentro de la monitorización se incluyen los signos vitales los cuales son indicadores que reflejan el estado fisiológico de los órganos vitales (cerebro, corazón, pulmones) y expresan de manera inmediata los cambios funcionales que suceden en el organismo, cambios que de otra manera no podrían ser cualificados ni cuantificados. Los principales signos vitales son:

1. Frecuencia cardiaca, que se mide por el pulso, en latidos/minuto.
2. Frecuencia respiratoria.
3. Tensión (presión) arterial.
4. Temperatura.

El monitoreo de los signos vitales no debe convertirse en una actividad automática o rutinaria; los resultados deben ser el reflejo de la evaluación clínica confiable del paciente por parte de enfermería, y su interpretación adecuada y oportuna ayuda a la enfermera y al médico a decidir conductas de manejo

1.1 Presión Arterial

Es una medida de la presión que ejerce la sangre sobre las paredes arteriales en su impulso a través de las arterias. Debido a que la sangre se mueve en forma de ondas, existen dos tipos de medidas de presión: la presión sistólica, que es la presión de la sangre debida a la contracción de los ventrículos, es decir, la presión máxima; y la presión diastólica, que es la presión que queda cuando los ventrículos se relajan; ésta es la presión mínima. La Presión Arterial Media (PAM) se calcula con la siguiente fórmula*: Presión sistólica – Presión diastólica / 3 + Presión diastólica*.

1.2 Temperatura

Es el equilibrio entre la producción de calor por el cuerpo y su pérdida. El centro termorregulador está situado en el hipotálamo. Cuando la temperatura sobrepasa el nivel normal se activan mecanismos como vasodilatación, hiperventilación y sudoración que promueven la pérdida de calor. Si por el contrario, la temperatura cae por debajo del nivel normal se activan mecanismos como aumento del metabolismo y contracciones espasmódicas que producen los escalofríos.

1.3 Frecuencia Respiratoria

La respiración es el proceso mediante el cual se toma oxígeno del aire ambiente y se expulsa el anhídrido carbónico del organismo. El ciclo respiratorio comprende una fase de inspiración y otra de espiración.

- Inspiración: fase activa; se inicia con la contracción del diafragma y los músculos intercostales.
- Espiración: fase pasiva; depende de la elasticidad pulmonar.

En condiciones patológicas intervienen los músculos accesorios de la inspiración (escálenos y esternocleidomastoideo) y de la espiración (abdominales).

1.4 Pulso

Es la onda pulsátil de la sangre, que representa el rendimiento del latido cardiaco, es decir cantidad de sangre que entra en las arterias con cada contracción ventricular y la adaptación de las arterias, o sea, su capacidad de contraerse y dilatarse. Asimismo, proporciona información sobre el funcionamiento de la válvula aórtica.

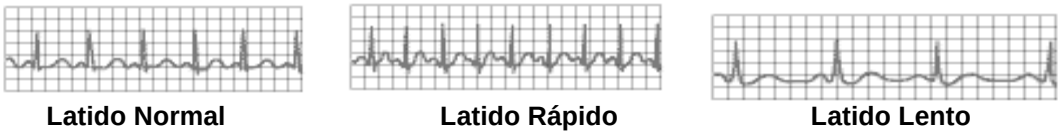
El pulso periférico se palpa fácilmente en pies, manos, cara y cuello. Realmente puede palparse en cualquier zona donde una arteria superficial pueda ser fácilmente comprimida contra una superficie ósea. La velocidad del pulso (latidos por minuto) corresponde a la frecuencia cardiaca, la cual varía con la edad, actividad física, estado emocional, fiebre, medicamentos y hemorragias, la fuerza con que se siente se denomina intensidad y la forma como se presenta sea de forma regular (normal) o irregular (anormal) se considera ritmo cardíaco.

Los valores de la frecuencia en condiciones normales puede estar entre:

- Recién nacidos: de 100 a 160 latidos por minuto
- Niños de 1 a 10 años: de 70 a 120 latidos por minuto
- Niños de más de 10 años y adultos: de 60 a 100 latidos por minuto

- Atletas bien entrenados: de 40 a 60 latidos por minuto

Cuando los latidos por minuto, están por encima o por debajo de los parámetros en condiciones normales se considera que se ha producido un ritmo irregular conocido como arritmia, este ritmo irregular pueden ser más rápido (taquicardia) o más lento (bradicardia) como lo muestra la figura A.1.



Latido Normal **Latido Rápido** **Latido Lento**

Figura A.1 Diferentes latidos del corazón

1.5 Arritmia

Una arritmia (también llamada disritmia) es un ritmo anormal del corazón, que puede hacer que éste bombee de forma menos eficaz.

Las arritmias pueden causar problemas en las contracciones de las cavidades del corazón al:

- No permitir que las cavidades se llenen con la cantidad adecuada de sangre porque la señal eléctrica hace que el corazón bombee demasiado rápido.

- No permitir que se bombee una cantidad suficiente de sangre hacia el cuerpo porque la señal eléctrica hace que el corazón bombee demasiado despacio o de forma demasiado irregular.

En cualquiera de esas situaciones, el cuerpo podría no recibir una cantidad suficiente de sangre. Esto se debe a que el corazón no puede bombear una cantidad adecuada con cada latido debido a los efectos de la arritmia sobre la frecuencia cardiaca.

Las arritmias también se dividen en dos categorías: ventriculares y supraventriculares. Las arritmias ventriculares se producen en las cavidades inferiores del corazón, denominados “ventrículos”. Las arritmias supraventriculares se producen en la zona que se encuentra encima de los ventrículos, generalmente en las aurículas, que son las cavidades superiores del corazón.

1.5.1 Bradicardia

La bradicardia es una frecuencia cardíaca muy baja. Se produce cuando el impulso eléctrico que estimula la contracción del corazón no se genera en el

marcapasos natural del corazón, el nódulo sinusal (nódulo SA), o no es enviado a las cavidades inferiores del corazón (los ventrículos) por las vías correctas.

La bradicardia afecta principalmente a las personas mayores, pero puede afectar a personas de cualquier edad, incluso a niños muy pequeños. Puede tener una de dos causas: el sistema nervioso central no comunica al corazón que debe bombear más o el nódulo SA podría estar dañado. Este daño puede deberse a una enfermedad cardiovascular, el proceso de envejecimiento o defectos heredados o congénitos, o podría ser causado por ciertos medicamentos, incluso aquellos que se administran para controlar las arritmias y la presión arterial alta.

1.5.2 Taquicardia

La taquicardia es una frecuencia cardíaca muy elevada. Hay muchos tipos diferentes de taquicardia, según dónde se origine el ritmo acelerado. Si se origina en los ventrículos, se denomina «taquicardia ventricular». Si se origina por encima de los ventrículos, se denomina «taquicardia supraventricular».

1.5.2.1 Taquicardia ventricular

La taquicardia ventricular es cuando el nódulo SA ya no controla el latido de los ventrículos, sino que otras zonas a lo largo de la vía de conducción eléctrica inferior asumen la función de marcapasos. Como la nueva señal no se desplaza por el músculo cardíaco por la vía normal, el músculo cardíaco no late en forma normal. Se aceleran los latidos del corazón y el paciente siente palpitaciones. Este ritmo irregular puede producir una extrema falta de aliento, mareo o desmayo (síncope).

1.5.2.2 Taquicardia supraventricular

La taquicardia supraventricular es una frecuencia cardíaca regular pero elevada, superior a los 150 latidos por minuto, que se origina en las aurículas. A diferencia de otros tipos de arritmia, la taquicardia supraventricular no se origina en el nódulo SA.

1.6 Dispositivos para tomar el Ritmo Cardíaco

1.6.1 Pulsioxímetro

El pulsioxímetro, u oxímetro de pulso, es un equipo utilizado en el área médica para medir dos parámetros: saturación de oxígeno en porcentaje (SpO2) y frecuencia cardiaca en pulsaciones por minuto (ppm). La saturación de oxígeno se define como la cantidad máxima de oxígeno que puede ser enlazado por la

hemoglobina en un mismo volumen de sangre. Su utilidad radica en detectar a tiempo un caso de déficit de oxígeno en la sangre debido a que una persona no puede sobrevivir más de 5 minutos sin el suministro de oxígeno al cerebro; así como también para detectar alteraciones en el ritmo cardiaco. Por esta razón es necesario en unidades de cuidados intensivos (UCI) y en las salas de cirugía, sobretodo cuando se realizan procesos quirúrgicos donde es necesario verificar el equilibrio de la concentración de los gases usados cuando se aplica anestesia al paciente.

1.6.2 Electrocardiograma

El electrocardiograma es una señal eléctrica resultante de la despolarización y repolarizacion auricular y ventricular del corazón. En la actualidad constituye uno de los métodos no invasivos más utilizados para realizar diagnósticos del estado de salud de una persona.

Para adquirir la señal de ECG necesitamos básicamente tres elementos: los electrodos, que son los sensores que se ponen en contacto con la piel del sujeto y se encargan de captar sus impulsos eléctricos; un bioamplicador, cuya función es ampliar la señal de los electrodos de micro-voltios a mili-voltios para que la señal pueda ser registrada; y por ultimo, un sistema que se encargue de mostrar y/o almacenar la señal adquirida.

Para obtener el ECG se han de posicionar los electrodos en zonas determinadas del cuerpo. En función de la posición de los electrodos el vector de campo eléctrico generado por la actividad cardíaca tendrá un modulo y dirección diferentes, lo que conlleva que, según la localización de los electro- dos, obtendremos amplitudes, polaridades y duraciones distintas. Por este motivo se han estandarizado distintas localizaciones para colocar los electrodos. Estas localizaciones reciben el nombre de derivaciones.

Desde el punto de vista eléctrico, el ciclo cardiaco tiene tres fases: despolarización, repolarizacion y descanso como lo muestra la figura A.2. En el ECG, estas fases corresponden a las siguientes ondas:

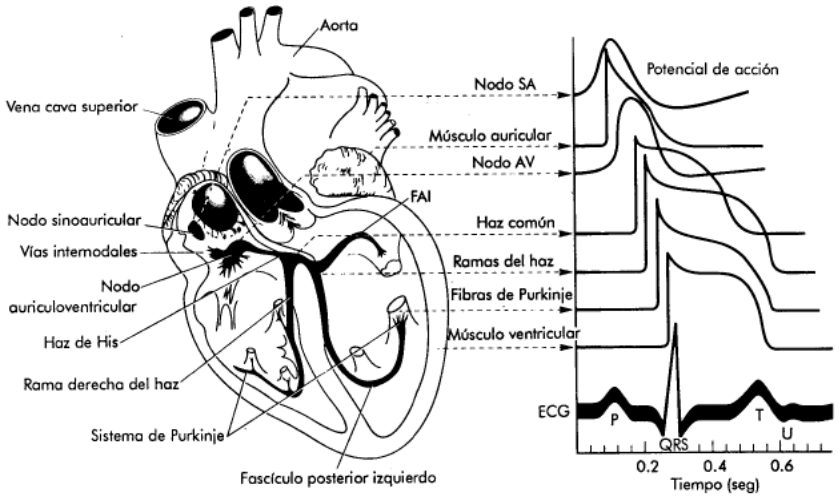


Figura A.2. Sistema conductor del corazón y correlación con el ECG

- Onda P: la onda P corresponde a la actividad eléctrica producida por a despolarización auricular, cuyo inicio marca el Nodo SA. Puede tener una apariencia monofásica, bi o difásica, o multifásica.

- Segmento PR: realmente se refiere al segmento PQ, pero no se utiliza esta terminología sino segmento PR. Durante este periodo de tiempo, que aparece como un segmento isoeléctrico en el ECG, se produce la retención de la excitación por el Nodo AV y el paso al Haz de His.

- Complejo QRS: el complejo QRS corresponde a la actividad eléctrica producida por la despolarización ventricular, momento en el que los ventrículos se contraen. La primera onda positiva, normalmente la de mayor amplitud, se llama R. La onda Q (si existe) se define como la primera onda negativa que precede a la onda R. La onda S se define como la primera onda negativa que sigue a la onda R. Se define como punto de unión (J) el final del complejo QRS, también llamado unión S-T.

- Segmento ST-T: este complejo representa la repolarizacion ventricular, aunque el comienzo de la repolarizacion ventricular, desde un punto de vista electrofisiológico, comienza de hecho unos milisegundos después del comienzo del complejo QRS.

- Onda T: la onda T corresponde al final de la actividad eléctrica producida por la repolarizacion ventricular. Análogamente a la onda P, puede tener apariencia monofásica, bi o difásica, y multifásica.

ANEXO B- TINYOS y TelosB

PARTE 1. TINYOS

TinyOS es un sistema operativo exclusivo para trabajar en redes de sensores, de libre distribución desarrollado en la Universidad de Berkeley de los Estados Unidos para ejecutar sobre sus propios nodos sensores llamados *motes*: mica, Mica2, Micadot, MicaZ, Telos y manejar las capacidades limitadas del hardware eficientemente. Trabaja en dos plataformas de desarrollo, en ambas impera un entorno basado en interprete de comandos sh, primeramente se desarrolló la versión bajo linux en primer termino fue red hat 9.0, fueron desarrolladas las aplicaciones básicas como el compilador de nesC, así como las librerías necesarias para la utilización de está. Poco más tarde se implemento el kit de desarrollo bajo cygwin, que a partir de la base de windows XP y Windows 2000 crea un entorno "linux like" que permite a usuarios menos experimentados en instalaciones bajo entornos UNIX desarrollar aplicaciones para tinyOS de forma más sencilla, aparte también instala en caso de no estar en el sistema una maquina virtual de Java. Tinyos está escrita en un sistema empotrado de red escrito en C (nesC- *Network Embedded Systems C*), un nuevo lenguaje de programación, de sintaxis similar a C con aplicaciones estructurales basado en componentes.

1.1 Estructura de un Componente

Un componente desde el punto de vista de programación esta compuesto por varias secciones y el conjunto de todas ellas dan lugar a la creación de dicho componente. En general, un componente posee tres grandes secciones que son: *Configuration, Implementation, Module*. Estas tres secciones han de estar obligatoriamente presentes en cualquier componente aunque puedan estar vacías.

TinyOs determina, que las secciones de *Configuration e Implementatión* han de ir en un fichero que recibirá el nombre del componente con la extensión *.nc* y la tercera sección de *Module* deberá de ir en otro fichero aparte que recibirá el nombre del componente concatenado con un M mayúscula (la M da el significado al fichero, es el significado de *Module*), este último fichero también poseerá la extensión *.nc*.

En tinyOS también existe un fichero *header* o cabecera con extensión *.h* que contenga todas las enumeraciones, registros o tipos de datos creados por el usuario de los que hace uso la aplicación, y cuando se realiza esto la forma de unir dicho fichero con los otros dos es utilizando al principio de los otros fichero la directiva *includes header.*

Todos los componentes que ofrece de forma intrínseca el sistema operativo se van a denominar componentes primitivos y los componentes proporcionados por terceros, contribuciones, librerías, aplicaciones se van a denominar componentes complejos.

Ahora se hará una breve explicación de cada una de las secciones que va a contener cada fichero.

1.1.1 Implementación

Esta sección se va a encargar de definir las conexiones que hay entre los diferentes componentes que utiliza la aplicación, esto es debido a que la programación de un componente (que se llevará a cabo en la sección de *module*) se hace utilizando interfaces y dichas interfaces para poder utilizarlas las ha de proporcionar un componente, entonces básicamente es esta sección se definen cuales son los componentes que proporcionan las interfaces de la aplicación (por lo general serán *componentes primitivos*). Ahora hay que definir un nuevo concepto que es la diferencia que existe entre una aplicación que está ya disponible para ser ejecutada en un sensor y un componente cualquiera. La diferencia es muy poca, y consiste en que una aplicación es un componente como cualquier cosa en este lenguaje que en su sección de implementación hace uso de un componente especial denominado *Main*., dicho componente proporciona una interfaz especial denominada *StdControl* y por tanto la aplicación ha de obligatoriamente proporcionar dicha interfaz.

Si se necesita crear un componente para un sensor, por lo general hay que utilizar las interfaces que nos proporcionan otros *componentes primitivos o no* y en definitiva para cada una de estas interfaces que se utilizan en la creación de un componente se han de definir obligatoriamente relaciones con las componentes que proporciones dichas interfaces, al proceso de definir estas relaciones se le conoce como wiring.

Estas relaciones se pueden definir de la siguiente manera; la sección de implementación constan de dos partes, la primera es una declaración de intenciones en la que se especifican todos los componentes que va a utilizar la aplicación, está declaración se realiza mediante la palabra reservada *components* después de la cual van separados por una coma simple, todos los componentes. La segunda parte de esta sección corresponde a la definición de las relaciones que hay entre las interfaces que utiliza la aplicación y las interfaces que proporcionan los componentes; y la forma de definir que la interfaz que proporciona un componente es la que se corresponde a la que se está utilizando en la aplicación es la siguiente:

componete.interfaz -> miaplicación.interfaz

A continuación se muestra un ejemplo de cómo definir las relaciones (wiring):

Se tiene un componente A que utiliza la interfaz IC que proporciona un componente B y el componente B utiliza la interfaz IC que proporciona el componente A, esto se puede resolver de la siguiente manera

```
implementation {
        components A, B;
        A.IC -> B.IC;
        B-IC -> A.IC;
}
```

Otro concepto muy utilizado en el *wiring* y que es vital para la correcta comprensión del mismo, además proporciona bastante potencia al lenguaje, es el concepto de *interfaces paramétricas*, permite tener varias instancias de una interfaz proporcionadas por un mismo componente.

La forma de indicarlo sería la siguiente:

```
implementation {
        components GenericComm, //Permite envio/recepcion de paquetes
        MiAplicacion;
        MiAplicacion.RecibeMsg -> GenericComm.RecibeMsg[MI_MENSAJE];
}
```

1.1.2 Configuración

Esta sección del componente ha de estar obligatoriamente presente, pero solo contendrá algo en el caso en el que se pretenda crear un componente no mediante su implementación de código directa (en la sección *Module)* sino mediante la composición de otros componentes ya creados, es un mecanismo que ofrece el lenguaje de programación para poder crear un nuevo componente mediante la combinación directa de otros.

La estructura de esta sección es la misma que la de la sección de *Implementación* de la sesión anterior, además posee la misma semántica, un ejemplo podría ser:

```
Configuration MiAplicacion {
        implementation {
                components GenericComm, //Permite envio/recepcion de paquetes
                TimerC; // Para realizar temporizaciones
                TimerC.StdControl-> GenericComm.StdControl;
```

```
        }
}
```

1.1.3 Módulo

Esta sección es la que por lo general, es más extensa y es en la que realmente se programa el comportamiento que se desea realizar en la aplicación. Esta a su vez, esta dividida en tres subsecciones : *Uses, Provide, Implementation*. La estructura de estas subsecciones dentro del fichero es la siguiente:

```
 module MiAplicacionM {
        provides {… }
        uses {… }
        }
implementation {…}
```

La primera subsección, *provides*, proporciona al lenguaje de programación cuales son las interfaces que va a proporcionar nuestro componente, en el caso de que nuestro componente sea una aplicación, como ya se vio anteriormente se ha de proporcionar como mínimo la interfaz *StdControl.*

Cuando se va a proporcionar una interfaz, desde un punto de vista orientado a objetos, se quiere decir que se va a implementar dicha interfaz, por lo tanto, se deberán de implementar los métodos que obligue a implementar dicha interfaz.

Dentro de la sección *provide* la forma de anunciar que se va a proporcionar una interfaz es la siguiente;

```
provides {
        interface interfazqueproporciono;
}
```

La subsección *uses* informa al lenguaje de programación que se va hacer uso de una interfaz, por lo tanto, si se hace uso de una interfaz se podrá realizar llamadas a los métodos de dicha interfaz.

Sin embargo, cuando se va a utilizar una interfaz, se derivan ciertas acciones que hay que realizar para el correcto funcionamiento de la aplicación:

Primero, si se utiliza una interfaz, obligatoriamente en la sección de *implementation* debe de haber un *wiring* que conecte dicha interfaz con un componente que la proporcione. Segundo y último, el utilizar una interfaz con lleva

implícitamente el tener que implementar los eventos que se puedan producir por el hecho de haber utilizado la interfaz.

La forma de indicar que vamos a utilizar una interfaz es la siguiente:

```
uses {
        interface interfacequeutilizo;
}
```

La última subsección, es quizás la más extensa, es la subsección *implementation,* esta contendrá todos los métodos necesarios para proporcionar el comportamiento deseado del componente o la aplicación. La estructura de la misma, es similar a la de un programa en lenguaje C pero con sutiles diferencias, aunque más bien que diferencia son añadidos para poder adaptar la programación al estilo de la programación orientada a eventos.

Esta subsección ha de contener como mínimo:

- Las variables globales que va a utilizar nuestra aplicación
- Las funciones que tenga que implementar debido a las interfaces que estoy proporcionando
- Los eventos que tenga que implementar debido a las interfaces de las que estoy realizando uso.

1.2 Tipos de datos

Los tipos de datos que se pueden utilizar en *NesC* son todos los que proporciona *C estándar* más unos cuantos más, que en realidad no aportan potencia de calculo pero son muy útiles para la construcción de paquetes ya que proporcionan al usuario información a cerca del numero de bits que ocupan y esto es importante a la hora de transmitir información vía radio.
Los tipos adicionales son:

- *uint16_t* que viene a ser un entero sin signo que ocupa 16 bits
- *uint8_t* que viene a ser un entero sin signo que ocupa 8 bits.
- *result_t,* se utiliza para devolver si usa función se ha ejecutado con éxito o no, viene a ser como un booleano pero con los valores SUCCESS y FAIL
- *bool*, es un valor booleano que puede valer *TRUE* o *FALSE*

1.3 Tipos de Funciones

En *NesC* las funciones pueden ser de tipos muy variados, primero existen las funciones clásicas con su misma semántica que en *C* y la forma de invocarlas es

la misma que en *C*. Ahora bien, existen otros tipos de funciones añadidos, estos son: *task, event,command* así que vamos a explicar sus diferencias y la forma de invocarlos. Las funciones *command* o comandos son básicamente funciones, que al igual que las clásicas se ejecutan de forma sincronía, es decir que cuando son llamadas se produce su ejecución inmediatamente. La forma de llamar a una de estas funciones es *call interfaz.nombreFuncion*. Las funciones *task* o tareas son funciones que se ejecutan concurrentemente en la aplicación, utilizan la misma filosofía que los *hilos o threads*, básicamente es una función normal que se invoca de la siguiente forma: *post interfaz.nombreTarea* y que inmediatamente después de su invocación, continua la ejecución del programa invocador. Las funciones *event* o eventos son funciones que son llamadas cuando se levanta una señal en el sistema, básicamente poseen la misma filosofía que la programación orientada a eventos, de manera que cuando el componente recibe un evento se realizará la invocación de dicha función aunque existe un método para poder invocar manualmente este tipo de funciones, es : *signal interfaz.nombreEvento* , además en el caso de tratarse de interfaces parametrizadas poder utilizar la variante *signal interfaz.nombreEvento[parámetro]*.

1.4 Componentes Primitivas de TinyOs

TinyOs ofrece al programador innumerables componentes primitivas, se explicaran las más importantes. La forma de verlas será una reunión entre los componentes que proporciona TinyOs y las interfaces que proporcionan, así se podrá tener una idea más general de la finalidad del componente.

1.4.1 Componente Main

Este es un componente especial que representa el cuerpo main al estilo de C de un programa y que necesita una interfaz StdControl para su funcionamiento, dicha interfaz será proporcionada por el componente que quiera convertirse en una aplicación.

1.4.1.1 Interfaz StdControl

La interfaz StdControl es una interfaz especial que han de proporcionar todos los componentes que se quieran llamar aplicación y por tanto sean ejecutables en un sensor.

Dicha interfaz obliga a implementar los siguientes métodos

command result_t init();
Este método se va a invocar cuando el sensor se arranque, es decir cuando se conecte

command result_t start();

Este método se va a invocar después de la invocación del método *init()* y cuando el sensor pase de off/on

command result_t stop();

Este método se va a invocar cuando el sensor pasa de on/off

1.4.2 Componentes GenericComm

Este componente proporciona mecanismos para poder enviar y recibir paquetes vía radio y vía puerto serie. Actúa como switch entre la radio y el puerto serie, de manera que si un paquete se envía vía radio a la dirección del puerto serie, cuando lo reciba este componente será enviado dicho paquete por el puerto serie. Utiliza el modelo de mensajes activos (AM- *Active Message*)[1] estándar de TinyOs por lo que se basa en el envío de paquetes *TOSMsg,* estos paquetes poseen la siguiente estructura;

```
uint16_t addr;
uint8_t type;
uint8_t group;
uint8_t length;
int8_t data[TOSH_DATA_LENGTH];
uint16_t crc;
```

El significado de cada uno de los campos es el siguiente:

- . *addr* , es la dirección a la que va destinado el paquete, existen ciertas direcciones especiales:
    - *TOS_BCAST_ADDR* – Es la dirección de broadcast
    - *TOS_LOCAL_ADDRESS* – Es la dirección local (localhost)
    - *TOS_UART_ADDR* – Es la dirección del puerto serie
- *type*, es el tipo de paquete que se está enviando, este tipo se utiliza en las interfaces parametricas para determinar que instancia de la interfaz se va a hacer cargo de procesar dicho paquete.
- . *group*, es el grupo al que pertenece el sensor que está enviando el paquete, de manera que solo lo reciban aquellos sensores que pertenezcan al

[1] Cada paquete de la red especifica un ID gestor que invoca al nodo receptor

mismo grupo, de está manera pueden coexistir dos redes diferentes de sensores sobre el mismo medio físico aéreo y sobre el mismo canal de radio.

- *lenght*, es la longitud total del paquete

- . *CRC*, pues eso mismo, es la corrección de errores

- *data*, este es el campo a mi parecer más importante y es el que nos posibilita jugar con los paquetes, en este campo se especifican los datos que se van a enviar, estos datos pueden ser una estructura que conforme un nuevo tipo de paquete de nivel superior.

Este componente proporciona las siguientes interfaces:

- *SendMsg[TIPO_PAQUETE]*

  Esta interfaz posibilita el poder enviar paquetes a un determinado destino, estos paquetes han de ser del tipo que se especifique en el parámetro de la interfaz.

- *ReceiveMsg[TIPO_PAQUETE]*

  Esta interfaz posibilita el poder recibir paquetes de un tipo determinado, especificado en el parámetro de la interfaz. Un sensor recibirá un paquete siempre y cuando el paquete sea enviado por un sensor perteneciente al mismo grupo y que además el tipo de paquete coincida con el parámetro de la interfaz y que el destino del paquete o bien sea de broadcast o bien coincida por mi dirección local.

- *StdControl*

  Si un componente proporciona la interfaz StdControl, es por que es una aplicación y por tanto es necesario llamar a los métodos de dicha interfaz a la vez que a los de nuestro componente.

1.4.2.1 Interfaz SendMsg

Esta interfaz proporciona mecanismos para poder enviar mensajes a otros sensores o al puerto serie del pc. Obliga a implementar el siguiente método:

command result_t send(uint16_t address, uint8_t length, TOS_MsgPtr msg)

Este método realiza el envío de un mensaje a la dirección *address*, dicho mensaje es *msg* (*TOS_MsgPtr* es un puntero a una estructura *TOS_Msg* que debe ser global ya que si se declara local, la duración de ese mensaje solo será hasta que la función termine y cuando esto ocurre todavía no se ha enviado el mensaje).

El componente que utilice esta interfaz debe de implementar el siguiente evento:

event result_t sendDone(TOS_MsgPtr msg, result_t success);

Este evento se va a producir en el caso en el que el envío de un mensaje haya sido satisfactorio.

1.4.2.2 Interfaz ReceiveMsg

Esta interfaz proporciona los mecanismos necesarios para poder recibir un paquete en el componente que la utilice. No obliga a implementar ningún método, sin embargo cualquier componente que haga uso de dicha interfaz se verá obligado a implementar el siguiente evento:

event TOS_MsgPtr receive(TOS_MsgPtr m);

Este evento se va a producir ante la llegada de un paquete y ha de devolver al final de su invocación el mismo paquete que se ha recibido para poder pasarlo a capas de nivel superior.

1.4.3 Componente TimerC

Este componente proporciona los mecanismos necesarios para poder llevar a cabo funciones de temporización de hasta 10 timer diferentes y para ello proporciona las siguientes interfaces:

- *Timer[id]*

  Esta interfaz posibilita el poder tener múltiples timers diferentes, y para ello se utiliza su parametrización de manera que como máximo puedo tener 10 timers. Para hacer el *wiring* de esta interfaz se suele utilizar la función que proporciona el compilador *built-in* que es *unique*, la cual proporcionaba un identificador único.

- *StdControl*

  Lo que significa que en la interfaz *StdControl* que proporcione el componente que estamos desarrollando tendremos que incluir

llamadas a las correspondientes funciones de esta interfaz para que la aplicación *TimerC* funcione correctamente.

1.4.3.1 Interfaz Timer

Esta interfaz proporciona mecanismos de temporización, para ello obliga a implementar los siguientes métodos:

command result_t start(char type, uint32_t interval);

Este método produce la inicialización o el arranque de la temporización, los parámetros que recibe son *type* para determinar si es una temporización continua, es decir que se repite de forma indefinida cada *interval* segundos, que en caso de ser así tomará el valor *TIMER_REPEAT,* o por el contrario si es un timer eventual que solo se invocará una vez, cuando se halla transcurrido el tiempo, que en este caso el caso será *TIMER_ONE_SHOT.*

command result_t stop();

Este método produce la parada de la temporización del reloj.

Ahora bien, un componente que haga uso de esta interfaz, esta obligado a implementar el siguiente evento:

event result_t fired();

Este evento se produce cuando se halla pasado el tiempo y por tanto haya terminado el timer, es decir es la acción que se va a ejecutar cada *interval* tiempo.

1.5 Estructura de TinyOS

La estructura de los directorios que componen TinyOS es la siguiente:

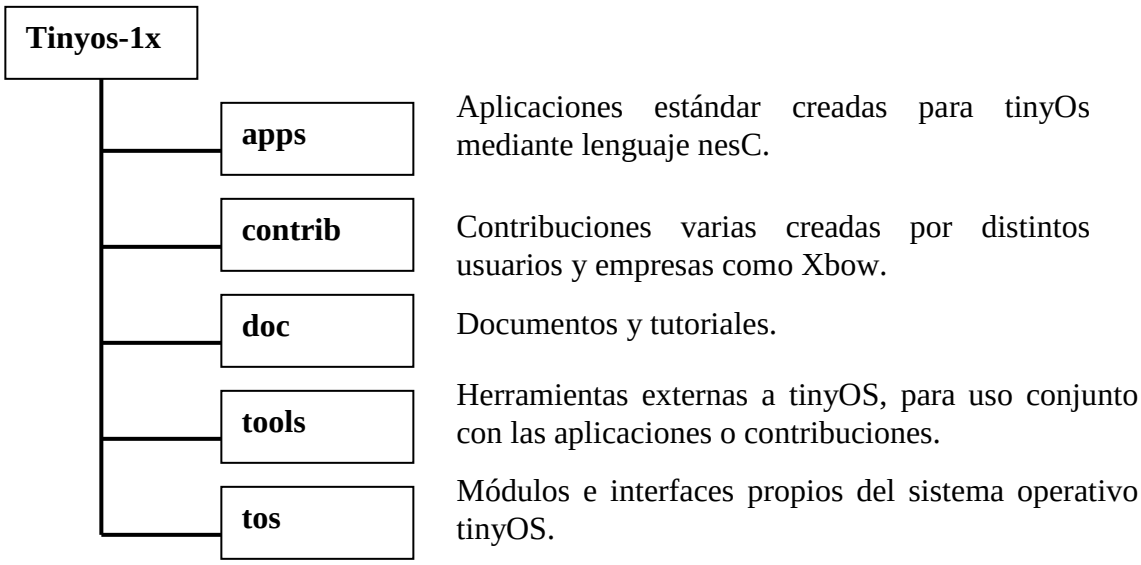


Figura B.1 Estructura principal de TinyOS

Dentro del directorio de /tinyos-1.x/tos se encuentran las siguientes directorios:

Tabla B.1 Directorio tinyos-1x/tos

Directorio	Descripción
/tos/interfaces	Contiene todas las interfaces que son proporcionadas por los componentes primitivos y por las aplicaciones de ejemplo
/tos/lib	Contiene librerías para resolver determinados problemas
/tos/system	Contiene todos los componentes primitivas que proporciona TinyOs
/tos/types	Contiene los tipos que se utilizan en las primitivas de TinyOS
/tos/platform	Contiene los ficheros necesarios para la ejecución en las diversas plataformas
/tos/senseorboard	Contiene los ficheros que son específicos de cada placa.

PARTE 2. TelosB

El dispositivo TelosB es un módulo de baja potencia usado en el desarrollo de las redes de sensores inalámbricas. TelosB cumple fielmente con el estándar Zigbee por lo tanto es compatible con otros nodos y sistemas desarrollados por otras empresas.

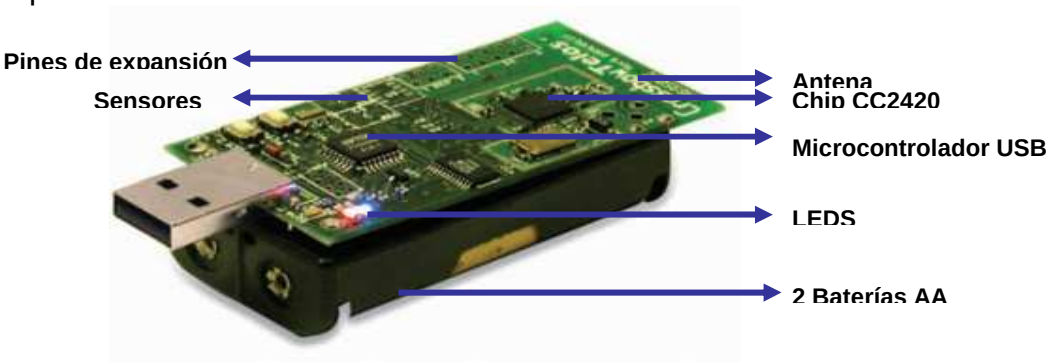


Figura B.2 Dispositivo TelosB

Este dispositivo está formado por componentes de alta calidad como el microcontrolador de Texas Instrument MSP430, el transmisor de Chipcon CC2420, una memoria flash M25P80 de 1MB para la ejecución de código, y un módulo USB con memoria de 2k y su respectivo controlador (ver tabla B.2 y figura B.2).

La programación de TelosB se hace a través de comunicación USB, consta con pines para el conector SMA en caso de querer integrar una antena externa y usa baterías AA conectadas en serie para la alimentación de energía haciendo un rango total de voltaje entre 2,1 a 3,6 voltajes DC.

Tabla B.2 Características TelosB

Telos b		
MCU	Chip	Texas Instrument MSP430
	Tipo	8 MHz, 16-bit RISC
	SRAM	10kB
	Conversor ADC/DAC	12 bits
Transceptor RF	Chip	CC2420 de Chipcon
	Frecuencia radio	Mín: 2400 MHz, Máx: 2483.5 MHz
	Máx. tasa datos	250 kbps
	Sensibilidad	-90 dBm (min), -94 dBm
	Potencia RF	-24 dBm to 0 dBm
	Rango Outdoor	75 -100 mts
	Rango Indoor	20 – 30 mts
Potencia	Tipo	2 baterías AA

PARTE 3 Instalación de TinyOS

TinyOS requiere para su correcto funcionamiento del uso de diversos comandos y funcionalidades de Linux, por lo tanto, si se desea trabajar en un entorno Windows es necesaria la instalación de la plataforma Cygwin, encargada de emular un entorno Linux en Windows

3.1 Instalación de TinyOS en Linux (debian 3.1 kernel 2.6)

Este ítem describe cómo descargar e instalar el software y las herramientas necesarias para el desarrollo TelosB usando TinyOS en Linux.

3.1.1 cvs de TinyOS

El repositorio cvs de TinyOS está ubicado en el proyecto SourceForge de tinyOS. Para obtener la última versión se debe utilizar una dirección IP real y digitar el siguiente comando al prompt del shell (o se puede encontrar en el CD de apoyo):

```
#cd ~/root
#cvs -d:pserver:anonymous@tinyos.cvs.sourceforge.net:/cvsroot/tinyos
login
```

Se presione *enter* cuando pida el password, y después se digita el siguiente comando:

```
#cvs -z3 -d:pserver:anonymous@tinyos.cvs.sourceforge.net:/cvsroot/tinyos
co tinyos-1.x
```

3.1.2 Variables de Entorno

Se debe adicionar las siguientes variables para telosB y tinyOS, en el script ~/.bashrc.

```
#tinyos
export TOSROOT="/root/tinyos-1.x"
export TOSDIR="$TOSROOT/tos"
export MAKERULES="$TOSROOT/tools/make/Makerules"

#java
export JDKROOT="/usr/lib/j2se/1.4"
export JAVAXROOT="/usr/lib/j2se/1.4/"
export
CLASSPATH=".:$TOSROOT/tools/java:$JAVAXROOT/jre/lib/ext/comm.jar:$TOSROOT
/tools/java/jars/jdom.jar:$JAVAXROOT/jre/lib/ext”
export PATH="$PATH:$TOSROOT/tools/java/jni"
export PATH="$JDKROOT/bin:$JDKROOT/jre/bin:$PATH"
export PATH="/usr/local/bin:$PATH"
```

```
export TOSCLASSPATH="$TOSROOT/tools/java/javapath"
export CLASSPATH=.:$TOSCLASSPATH:$CLASSPATH

#msp
export MSPGCCROOT="/opt/msp430"
export PATH="$MSPGCCROOT/bin:$PATH"
#export PATH="$PATH:$MSPGCCROOT/bin"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/lib/"
export
LD_LIBRARY_PATH="$LD_LIBRARY_PATH::"usr/lib/j2se/1.4/jre/lib/i386/libTOSC
omm.so"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:"$JAVAXROOT/jre/bin/"
export TELOS_REV=B

export PATH="$PATH:$TOSROOT/tools/java/net/tinyos/sim"
export PATH="$PATH:$TOSROOT/tools/java/net/tinyos/message"
export MOTECOM=serial@COM1:telos
```

3.1.3 MSPGCC

MSPGCC es el compilador usado por TinyOS para la plataforma del microcontrolador TI MSP430. La forma de configurarlo es a través del siguiente comando:

```
#sudo ./build-mspgcc install
```

3.1.4 MSP430-bsl

El msp430-bsl es el programador para el MSP430 desarrollado por el proyecto de MSPGCC. TelosB requiere una versión de distribución especial de TinyOS para soportar el hardware. Para instalar MSP430-bsl hay que hacer un enlace de bsl.py a msp430-bsl, como se muestra en las siguientes líneas:

```
#cd /root/tinyos-1.x/tools/src/mspgcc-pybsl
#ln -s bsl.py /opt/msp430/bin/msp430-bsl
```

3.1.5 nesC

NesC es el lenguaje de programación de tinyOS, y requiere de su propio compilador, se puede bajar por medio del repositorio cvs de la siguiente forma (o se puede encontrar en el CD de apoyo):

```
#cd usr/local/src
#cvs -d:pserver:anonymous@nescc.cvs.sourceforge.net:/cvsroot/nescc login
```

Presione *enter* cuando pida el password, y después se digita el siguiente comando:

```
#cvs -z3 -d:pserver:anonymous@nescc.cvs.sourceforge.net:/cvsroot/nescc co
-P nesc
```

Antes de compilar se instalan los siguientes paquetes (se puede hacer a través de synaptic):

- automake
- emacs
- gperf

Para compilar nesC se debe ubicar en la carpeta de nesC y digitar los siguientes comandos

```
#./Bootstrap
#./configure
#make
#make install

#cd ~/tinyos-1.x/tools/src/ncc
#./Bootstrap
#./configure
#sudo make install
```

Para comprobar de que quedó bien instalado se digita el siguiente comando:

```
# ncc –version
```

Debe retornar algo como ncc 1.2alpha5

3.1.6 Java

Java es utilizado por las herramientas de la interfaz entre el dispositivo telosB y el PC. j2sdk1.4 se puede obtener a través del siguiente repositorio (en el script /etc/apt/sources.list):

deb ftp://ftp.tux.org/java/debian/ sarge non-free

y con synaptic se puede obtenerlo.

3.1.7 Herramientas de java para tinyOS

TinyOS dentro de sus carpetas tiene una llamada java, en la cual tiene unos programas específicamente diseñados para este sistema operativo, la forma de compilarlos es digitando las siguientes líneas:

```
#cd $TOSROOT/tools/java
```

```
#make
#make
```

3.1.8 javax.comm

Javax.com se obtiene de la página (no tomar la versión javax.comm 3): http://javashoplm.sun.com/ECom/docs/Welcome.jsp?StoreId=22&PartDetailId=7175-comm_api-2.0.3-oth-JPR&SiteId=JSC&TransactionId=noreg

Se digita las siguientes líneas:

```
#tar xvzf javax_comm-2_0_3-solsparc.tar.Z
# cp commapi/comm.jar $JDKROOT/jre/lib/ext/
# /bin/echo Driver=gnu.io.RXTXCommDriver >
$JDKROOT/jre/lib/javax.comm.properties
```

3.1.9 TOSComm

TOSComm es un paquete beta proporcionado por TinyOS que proporciona el acceso del puerto serial por medio de Java. Para utilizarlo se requiere herramientas tales como SerialForwarder. TOSComm tiene la intención de reemplazar a JavaComm. Para instalarlo hay que tener instalado el paquete swig(con synaptic) y después se digita los siguientes comandos:

```
#cd $TOSROOT/beta/TOSComm
# sudo chmod a+w /usr/lib/j2se/1.4/jre/lib/i386/
#make install
```

Debe retornar sin errores.

3.1.10 TinyOS Scripts

Algunos scripts de TinyOS, específicamente el set-mote-id, son necesarios para el proceso de compilación. Para instalar estos scripts se digita los siguientes comandos:

```
#cd $TOSROOT/tools/scripts
#make install prefix=/usr/local
```

3.1.11 Make TinyOS

A través de los siguientes comandos:

```
#cd $TOSROOT/
#TINYOS_NP=BNP make
```

3.1.12 Make telosb

Para compilar algunas aplicaciones que trae tinyOS para telosB se digita los siguientes comandos:

```
#cd apps
#make telosb
```

Algunas aplicaciones presentan fallas.

3.1.13 Motelist

Motelist es una utilidad de código de línea que despliega el puerto serie que tiene adherido el telosB, disponible en $TOSROOT/tools/src/motelist/. Para utilizarlo se digita los siguientes comandos:

```
#cd /usr/local/bin
#ln -s $TOSROOT/tools/src/motelist/motelist-linux motelist
```

3.1.14 FTDI

FTDI es el controlador del usb del telosB, lo cual permite manejar el controlador del puerto virtual COM (VCP- *Virtual COM Port*). El controlador VCP emula un puerto serial estándar del PC de tal manera que el dispositivo USB se puede comunicar como un dispositivo estándar RS232. El controlador se puede obtener de la página http://www.ftdichip.com/Drivers/VCP.htm. Para instalarlo hay que seguir los siguientes pasos.

- Se extrae el archivo ftdi_sio.tar.gz a través del los siguientes comandos:

  ```
  #gunzip ftdi_sio.tar.gz
  #tar -xvf ftdi_sio.tar
  ```
- Se crea el controlador
  ```
  #make
  ```
- Se conecta el dispositivo telosB al pc.
- Se chequea si por defecto el controlador ha sido cargado, a través del siguiente comando
  ```
  #lsmod
  ```

Se pude ver que el controlador ftdio ha sido cargado.

3.1.15 SerialForwarder

SerialForwarder es un programa proporcionado por tinyOS para realizar la comunicación de la aplicación del pc (localizada por defecto en el puerto TCP

9001) con el puerto serial. Para SerialForwarder funcione bien se debe digitar los siguientes comandos:

```
#cd :~/tinyos-1.x/tools/java/jni
# make install
#cd –r /root/tinyos-1.x/tools/java/jni/libgetenv.so
usr/lib/j2se/1.4/jre/lib/i386
```

3.2 Instalación de TinyOS en Windows

Primero hay que tener en cuenta que si se tiene instalado cygwin instalado, por recomendación hay que desistalarlo para asegurar de que tinyOS quede completamente instalado.

3.2.1 Instalación de TinyOS

Los pasos para instalar tinyOS en windows son los siguientes:

- Dar doble click sobre el ejecutable ***tinyos-1.1.11-3is.exe*** de la carpeta de tinyOS del CD de apoyo.
- Cuando aparece una ventana de InstallShield Wizard , escoger *next.*
- Se recomienda escoger ***Complete Install***. Esto instala Cygwin, Java, el compilador MSPGCC para MSP430 , MSP430-bsl, motelist y los demás programas de TinyOS.

3.2.2 Instalación del Controlador VCP de FTDI USB

El dispositivo TelosB utiliza el FTDI FT232BM para crear el puerto USB como puerto virtual COM, por lo tanto se necesita instalar el controlador FT232BM VCP, de la siguiente forma: cuando se conecta el dispositivo telosB por primera vez, aparece una ventana informando que se ha encontrado nuevo hardware, hay que seleccionar “instalar de una ubicación específica (avanzado)” y buscar la carpeta FT232BM VCP del CD de apoyo.

PARTE 4 COMPILACIÓN DE LAS APLICACIONES DE NESC

Esta parte se muestra como programar sus dispositivos telosB a través del puerto USB.

Las aplicaciones que tiene tinyOS para programarlos en los dispositivos se encuentra se encuentra en la carpeta ~/tinyos-1.x/app/. Como ejemplo se puede programar Blink que es una aplicación sencilla en la cual permite prender el led

rojo del dispositivo telosB con un periodo de 1 hz. La programación se realiza de la siguiente manera:

```
# cd ~/tinyos-1x/apps/Blink
#make install.x telosb
```

Donde:

- x hace referencia a la dirección del nodo, que esun valor que va entre 0 a 255
- telosb es la plataforma usada, puede ser micaz, telosb, mica2, según la plataforma que se esté usando, pero en esté caso se utiliza el dispositivos telosB.

ANEXO C- CÓDIGO

PARTE 1. Código Java

1.1 Aplicación del Módulo Transmisor

La aplicación del módulo Transmisor está conformada por la simulación del ritmo cardíaco. Los archivos más importantes son los siguientes:

1.1.1 ECGPANEL.java

Ubicado en el archivo /ritmo/interface/ECGPANEL.java). Permite dibujar la interfaz grafica del ECG para cada paciente

```
package net.tinyos.practica.interfaces;

import java.awt.Color;
import java.awt.Graphics;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.Random;

import javax.swing.JPanel;

public class ECGPanel extends JPanel implements Runnable {
	public static short pac = 1;
	    public static String paciente = "";
	public static int setPoint = 80;
	    public static double actual = 80;
	    public static double alto = 120;
	    public static double bajo = 60;
	    public boolean sendingAlarma=false;
	public static net.tinyos.practica.action.EnviarAction action=null;

	    public ECGPanel(int nMuestras){
	        muestras=new Cola(nMuestras);
	        new Thread(this).start();
	    }

	    private static boolean ACTIVE = true;
	    private Cola muestras=null;

	    public void paint(Graphics g) {
	        g.clearRect(0,0,getWidth(),getHeight());

	        g.setColor(Color.BLACK);
	        g.fillRect(0,0,getWidth(),getHeight());
	        g.setColor(Color.DARK_GRAY);
	        g.drawRoundRect(10,10,getWidth()-20,getHeight()-20,20,20);

	        for (int i = 20; i < Math.max(getWidth(),getHeight()); i+=20) {
	            g.drawLine(i,10,i,getHeight()-15);
	            g.drawLine(10,i,getWidth()-15,i);
```

```
		}
		g.setColor(Color.WHITE);
		g.drawLine(10,getHeight()/2,getWidth()-15,getHeight()/2);
		drawFunction(g);
		drawCredit(g);
	}

	private void drawCredit(Graphics g) {


		//g.fillRoundRect(40,getHeight()-110,150,getHeight()-100,20,20);
		g.setColor(Color.WHITE);
		if(this.paciente!=null);
		g.drawString("Paciente: "+this.paciente,getWidth()/2-50,20);
		if(this.actual<this.alto&&this.actual>this.bajo){
			g.setColor(Color.GREEN);
		}else{
			if(!sendingAlarma){
				if(this.actual>this.alto){
					System.out.println("TAQUI");
			action.enviar(net.tinyos.practica.util.PaqueteConstans.TAQUI,pac);
				}else{
					System.out.println("BRADI");
			action.enviar(net.tinyos.practica.util.PaqueteConstans.BRADI,pac);
				}
				//espera un 20 seg para enviar nueva alarma
				new Thread(){
					public void run(){
						try {
							sendingAlarma=true;
							Thread.sleep(20000);
							sendingAlarma=false;
						} catch (Exception e) {}
					}
				}.start();
			}
			g.setColor(Color.RED);
		}
		g.drawString("Medida Actual: "+(int)this.actual,50,getHeight()-80);

		g.setColor(Color.LIGHT_GRAY);
		g.drawString("Nivel Alto: "+this.alto,50,getHeight()-50);
		g.drawString("Nivel Bajo: "+this.bajo,50,getHeight()-30);


	}

	private void drawFunction(Graphics g) {
		g.setColor(Color.YELLOW);
		Iterator it = muestras.iterar();
		int x0=10,y0=0;
		if(it!=null){
			int x=10;
			while(it.hasNext()){
				x+=2;
				int y = ((Integer)it.next()).intValue();
				g.drawLine(x0,((getHeight()/2)-y0),x,((getHeight()/2)-y));
				x0=x;y0=y;
```

```
                }
            }
        }

        public void run() {
            while(ACTIVE){
                //muestras.put(Integer.valueOf(new Random().nextInt(100)));
                nextMuestra(setPoint);
                repaint();
                try {
                    Thread.sleep(1000);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }

        private void nextMuestra(int muestra){
            actual=muestra+Math.random()*10;
            double pulso=actual;
            //primer rizo
            double pr=(pulso/267);
            for (double x = 0; x < Math.PI; x+=pr) {

                muestras.put(new Integer((int)(10*Math.sin(x))));
            }
            //espacio
            for (int x = 0; x < 3; x++) {
                muestras.put(new Integer(0));
            }
            //segundo rizo
            for (double x = Math.PI; x < 2*Math.PI; x+=(2*pr)) {
                muestras.put(new Integer((int)(10*Math.sin(x))));
            }
            //pico
            for (double x = 0; x < Math.PI; x+=(3*pr-0.1)) {
                muestras.put(new Integer((int)(Math.random()*10+pulso*Math.sin(x))));
            }
            //segundo rizo
            for (double x = Math.PI; x < 2*Math.PI; x+=(3*pr)) {
                muestras.put(new Integer((int)((1600/pulso)*Math.sin(x))));
            }
            //espacio
            for (int x = 0; x < (320/pulso); x++) {
                muestras.put(new Integer(0));
            }
            //          primer rizo
            for (double x = 0; x < Math.PI; x+=pr) {
                muestras.put(new Integer((int)(20*Math.sin(x))));
            }
            //          primer rizo
            for (double x = 0; x < Math.PI; x+=pr) {
                muestras.put(new Integer((int)(7*Math.sin(x))));
            }
            //espacio
            for (int x = 0; x < (320/pulso); x++) {
                muestras.put(new Integer(0));
            }
```

```
        }
}
```

1.1.2 MainClass.java

Ubicado /ritmo/MainClass.java

```
package net.tinyos.practica;

import net.tinyos.message.MoteIF;

import java.net.Socket;
import java.net.SocketAddress;
import java.net.UnknownHostException;
import java.io.*;
import java.util.Properties;

import net.tinyos.practica.data.PaqueteMIG;
import net.tinyos.practica.gui.MainForm;
import net.tinyos.practica.util.Constans;



public class MainClass {
   public static void main(String[] args) {

      File prop = new File(Constans.Properties);

      try {
         FileInputStream fin = new FileInputStream(prop);
         Properties p = new Properties();
         p.load(fin);
         if (p.getProperty("SF_ACTIVATE") != null) {
            String activate = p.getProperty("SF_ACTIVATE");
            if (activate.equals("SI")) {
               Constans.SF_ACTIVATE = true;
            } else {
               Constans.SF_ACTIVATE = false;
            }
         }
         if (p.getProperty("SF_IP") != null) {
            Constans.SF_IP = p.getProperty("SF_IP");
         }
         if (p.getProperty("SF_COMM") != null) {
            Constans.SF_COMM = p.getProperty("SF_COMM");
         }
         if (p.getProperty("SF_PORT") != null) {
            Constans.SF_PORT = Integer.parseInt(p.getProperty("SF_PORT"));
         }
      } catch (FileNotFoundException e) {

      } catch (IOException ioe) {
         ioe.printStackTrace();
      }
      new MainForm();
```

```
    }
}
```

1.1.3 EnviarAction.java

Ubicado en el archivo /ritmo/action/EnviarAction.java, permite enviar al dispositivo teloB el evento requerido.

```
package net.tinyos.practica.action;

import net.tinyos.practica.util.Constans;
import net.tinyos.practica.util.PaqueteConstans;

import javax.swing.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.io.IOException;

import net.tinyos.practica.gui.MainForm;
import net.tinyos.practica.data.PaqueteMIG;
import net.tinyos.practica.data.Conexion;
import net.tinyos.message.MoteIF;



public class EnviarAction implements ActionListener {
    MainForm mainForm;

    public EnviarAction(MainForm mainForm) {
        this.mainForm = mainForm;
    }

    public MainForm getMainForm() {
        return mainForm;
    }

    public void setMainForm(MainForm mainForm) {
        this.mainForm = mainForm;
    }

    public void enviar(short i,short paciente){
        PaqueteMIG nuevo = new PaqueteMIG();
        nuevo.set_count((short) 0);
        nuevo.set_band((short) 0);
        nuevo.set_type(PaqueteConstans.PETICION);
        nuevo.set_sala(PaqueteConstans.SALA_1);
        nuevo.set_pac(paciente);
        nuevo.set_data(i);
        System.out.println("DATA: "+i);
        {
            try {
                MoteIF conexion = Conexion.getConexion(this.getMainForm());
                        conexion.send( 45, nuevo);
                this.getMainForm().getLogger().append("> El paquete ha sido enviado correctamente\n");

            } catch (IOException e1) {
```

```
            JOptionPane.showMessageDialog(this.getMainForm().getJPrincipal(), "Error no se ha establecido la
conexión");
            this.getMainForm().getLogger().append("> Error se ha perdido la conexión.(El paquete no ha sido
enviado)\n");
            this.getMainForm().setDisConectedStatus();
            Conexion.closeConexion();
         }
      }
   }


   public void actionPerformed(ActionEvent e) {
      boolean error = false;
      JButton boton = (JButton) e.getSource();
      PaqueteMIG nuevo = new PaqueteMIG();
      nuevo.set_count((short) 0);
      nuevo.set_band((short) 0);
      nuevo.set_type(PaqueteConstans.PETICION);

     if (boton.getName().equals(Constans.NPrimerPac)) {
                  this.getMainForm().getLogger().append("\n> Enviando un paquete del Primer Paciente\n");
                  this.getMainForm().getJPac1().setText("");
         nuevo.set_sala(PaqueteConstans.SALA_1);
         nuevo.set_pac(PaqueteConstans.PACIENTE_1);
      }
      if (boton.getName().equals(Constans.NSegundoPac)) {
         this.getMainForm().getLogger().append("\n> Enviando un paquete del Segundo Paciente\n");
           nuevo.set_sala(PaqueteConstans.SALA_1);
         nuevo.set_pac(PaqueteConstans.PACIENTE_2);
         this.getMainForm().getJPac2().setText("");
      }
           if (boton.getName().equals(Constans.NTercerPac)) {
         this.getMainForm().getLogger().append("\n> Enviando un paquete del Tercer Paciente\n");
         this.getMainForm().getJPac3().setText("");
           nuevo.set_sala(PaqueteConstans.SALA_1);
         nuevo.set_pac(PaqueteConstans.PACIENTE_3);
      }
      if (boton.getName().equals(Constans.NCuartoPac)) {
         this.getMainForm().getLogger().append("\n> Enviando un paquete del Cuarto Paciente\n");
           nuevo.set_sala(PaqueteConstans.SALA_1);
         nuevo.set_pac(PaqueteConstans.PACIENTE_4);
         this.getMainForm().getJPac4().setText("");
      }

           if (!error) {
         // Lo envio ala otro mote
         try {
            MoteIF conexion = Conexion.getConexion(this.getMainForm());
                  conexion.send( 45, nuevo);
            this.getMainForm().getLogger().append("> El paquete ha sido enviado correctamente\n");

         } catch (IOException e1) {
            JOptionPane.showMessageDialog(this.getMainForm().getJPrincipal(), "Error no se ha establecido la
conexión");
            this.getMainForm().getLogger().append("> Error se ha perdido la conexión.(El paquete no ha sido
enviado)\n");
            this.getMainForm().setDisConectedStatus();
            Conexion.closeConexion();
```

```
        }
      }
    }
}
```

1.2 Aplicación del Módulo Receptor

La aplicación del módulo Receptor está conformada por la asignación de turnos de los médicos. Los archivos más importantes son los siguientes:

1.2.1 ContenedorPrincipal.java

Ubicado en el archivo /ritmo/intefaces/ContenedorPrincipal.java. Permite crear los dos usuarios (el administrador y el médico) y asignar turnos a los médicos.

```
/ * ContenedorPrincipal.java
 *

*/package net.tinyos.practica.interfaces;

import net.tinyos.practica.interfaces.datos.Scripts;
import java.util.Calendar;
import java.util.Collection;
import java.util.Date;
import java.util.Iterator;
import java.util.Map;
import javax.swing.JDesktopPane;
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

import net.tinyos.practica.data.Conexion;
import net.tinyos.message.MessageListener;
import net.tinyos.message.MoteIF;
import net.tinyos.practica.data.PaqueteMIG;

import javax.swing.JInternalFrame;
import javax.swing.JPanel;

import java.io.*;
/**
 * El metodo <i>printContent</i> es un ejemplo de como acceder a los datos que el usuario
 * ha ido insertando...
 */

public class ContenedorPrincipal extends javax.swing.JFrame implements MessageListener {
    /**
     * Ruta del archivo de configuración de scripts
     */
    private static final String scriptPath="scripts.config";
    /**
     * Numero de Medico y Pacientes de la aplicacion
     */
    public static final int nMedicos=10;
    public static final int nPacientes=20;
```

```
/**
 * Definición de login y password para modificación
 **/
private static String login="medico";
private static String passwd="medico";

/**
 * Definición de login y password para solo lectura
 **/
private String loginReadOnly="admin";
private String passwdReadOnly="ritmo";


private static ContenedorPrincipal instances=null;
private static javax.swing.JDesktopPane desk=null;
private static final int ancho=700;
private static final int alto=500;

public static Interna init=null;
public static Interna validar=null;
public static Interna validarReadOnly=null;
public static Interna agregarDatos=null;
public static Interna asignacion=null;
private static MoteIF cx=null;
private static Scripts script=null;

public void connectAndWait(){
    cx=Conexion.getConexion((MessageListener)this);
    script = new Scripts(scriptPath);
}

/** Creates new form ContenedorPrincipal */
public ContenedorPrincipal() {
    instances=this;
    desk=new JDesktopPane();
    desk.setBackground(Color.GRAY);
    this.setContentPane(desk);
    Dimension screen=Toolkit.getDefaultToolkit().getScreenSize();;
    setLocation(  (screen.width-ancho)/2,(screen.height-alto)/2);
    this.setSize(ancho,alto);
    this.setTitle("Control y Asignación de Consultas Médicas");

    init=new Interna(new PInicio(),"Ventana Principal",false,false,true);
    init.centre(this,500,300);

    validar=new Interna(new PValidacin(false),"Validar Acceso de Modificación",false,false,false);
    validar.centre(this,350,130);

    validarReadOnly=new Interna(new PValidacin(true),"Validar Acceso de Lectura",false,false,false);
    validarReadOnly.centre(this,350,130);

    agregarDatos=new Interna(new AgregarDatos(),"Validar Acceso",false,true,true);
    agregarDatos.centre(this,500,300);

    asignacion=new Interna(new Asignacion(),"Tabla de Asignaciones Médicas",true,true,true);
    inicio();
}
```

```
public static void inicio(){
    desk.removeAll();desk.repaint();
    desk.add(init);
}

public static void validar(){
    desk.removeAll();desk.repaint();
    desk.add(validar);
}

public static void asignaciones(boolean readOnly){
    try{
        desk.add(asignacion);
    }catch(Exception ex){desk.add(asignacion);}
    asignacion.centre(instances,580,150);
    ((Asignacion)asignacion.getContent()).readOnly(readOnly);
    asignacion.setVisible(true);
    asignacion.toFront();

}

public static void validarAceptar(String login,String passw,boolean readOnly){
    ((AgregarDatos)agregarDatos.getContent()).readOnly(readOnly);
    if(readOnly){
        if(login.equals(instances.loginReadOnly)&&passw.equals(instances.passwdReadOnly)){
            desk.removeAll();desk.repaint();
            agregarDatos.setVisible(true);
            desk.add(agregarDatos);
        }else{
            ((PValidacin)validarReadOnly.getContent()).setMessage("Acceso Invalido");
        }
    }else{
        if(login.equals(instances.login)&&passw.equals(instances.passwd)){
            desk.removeAll();desk.repaint();
            agregarDatos.setVisible(true);
            desk.add(agregarDatos);
        }else{
            ((PValidacin)validar.getContent()).setMessage("Acceso Invalido");
        }
    }
}

/**
 * en este metodo se muestra como recuperar cada uno de los valores de los datos
 */
public static void printContent(){

    //información de los médicos
    Map medicos=((AgregarDatos)agregarDatos.getContent()).getMedicos();
    Iterator ms=medicos.keySet().iterator();
    while(ms.hasNext()){
        Object codigo = ms.next();
        String nombre=((Medico)medicos.get(codigo)).getMedico();
        System.out.println(codigo+" "+nombre);
    }

    Map pacientes=((AgregarDatos)agregarDatos.getContent()).getPacientes();
```

```
        Iterator ps=pacientes.keySet().iterator();
        while(ps.hasNext()){
            Object codigo = ps.next();
            String value=((Pacientes)pacientes.get(codigo)).toString();
            System.out.println(codigo+" "+value);
        }

        //Asignaciones
        Asignacion asig= ((Asignacion)asignacion.getContent());
        Map mapa = asig.getAsignaciones();

        System.out.println("Lunes"+mapa.get("lunes"));
        System.out.println("Martes"+mapa.get("martes"));
        System.out.println("Miercoles"+mapa.get("miercoles"));
        System.out.println("Jueves"+mapa.get("jueves"));
        System.out.println("Viernes"+mapa.get("viernes"));
        System.out.println("Sabado"+mapa.get("sabado"));
        System.out.println("Domingo"+mapa.get("domingo"));

        //ejemplo para el lunes
        Collection lunes = (Collection)mapa.get("lunes");
        System.out.println("LUNES");
        Iterator it = lunes.iterator();
        String h12_6am = it.next().toString();
        String h6_12am = it.next().toString();
        String h12_6pm = it.next().toString();
        String h6_12pm = it.next().toString();

        System.out.println("12 - 6 AM "+h12_6am);
        System.out.println("6 - 12 AM "+h6_12am);
        System.out.println("12 - 6 PM "+h12_6pm);
        System.out.println("6 - 12 PM "+h6_12pm);

    }

    public static String getMedicoDeTurno(){
        Calendar now =  Calendar.getInstance();
        int dia = now.get(Calendar.DAY_OF_WEEK);
        String diaName="";
        switch(dia){
            case 1:
                diaName="domingo";
            break;
            case 2:
                diaName="lunes";
            break;
            case 3:
                diaName="martes";
            break;
            case 4:
                diaName="miercoles";
            break;
            case 5:
                diaName="jueves";
            break;
            case 6:
                diaName="viernes";
            break;
```

```
            case 7:
                diaName="sabado";
            break;
        }

        Collection h = (Collection)((Asignacion)asignacion.getContent()).getAsignaciones().get(diaName);

        int hora = now.get(Calendar.HOUR);
        int ampm=now.get(Calendar.AM_PM);
        String turno="";
        if(hora<6){
            if(ampm==0){
                //am
                turno="12-6AM";
            }else{
                //pm
                turno="12-6PM";
            }
        }else{
            if(ampm==0){
                //am
                turno="6-12AM";
            }else{
                //pm
                turno="6-12PM";
            }
        }
        String codigoMedico = getCodigo(h,turno);
        return codigoMedico.toLowerCase();
        /**
         Map medicos=((AgregarDatos)agregarDatos.getContent()).getMedicos();
         if(codigoMedico.equals("")||medicos.get(codigoMedico.toUpperCase())==null)return "ninguno";
        return ((Medico)medicos.get(codigoMedico.toUpperCase())).getMedico();
        */
    }

    private static String getCodigo(Collection h,String turno){
        if(turno.equalsIgnoreCase("12-6AM")){
            return h.toArray()[0].toString();
        }else if(turno.equalsIgnoreCase("6-12AM")){
            return h.toArray()[1].toString();
        }else if(turno.equalsIgnoreCase("12-6PM")){
            return h.toArray()[2].toString();
        }else if(turno.equalsIgnoreCase("6-12PM")){
            return h.toArray()[3].toString();
        }else return "NULL";
    }

    public static void callAgredarDatos(){
        desk.removeAll();desk.repaint();
        desk.add(validar);
    }

    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">//GEN-BEGIN:initComponents
    private void initComponents() {
        addWindowListener(new WindowListener(){

                                public void windowActivated(WindowEvent arg0) {
```

```
                        // TODO Auto-generated method stub

                    }

                    public void windowClosed(WindowEvent arg0) {
                        System.exit(0);
                    }

                    public void windowClosing(WindowEvent arg0) {
                        System.exit(0);
                    }

                    public void windowDeactivated(WindowEvent arg0) {
                        // TODO Auto-generated method stub

                    }

                    public void windowDeiconified(WindowEvent arg0) {
                        // TODO Auto-generated method stub

                    }

                    public void windowIconified(WindowEvent arg0) {
                        // TODO Auto-generated method stub

                    }

                    public void windowOpened(WindowEvent arg0) {
                        // TODO Auto-generated method stub

                    }});
    }// </editor-fold>//GEN-END:initComponents

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                ContenedorPrincipal c=new ContenedorPrincipal();
                c.setVisible(true);
                c.connectAndWait();

            }
        });
    }

    static void verDatos() {
        desk.removeAll();desk.repaint();
        desk.add(validarReadOnly);
    }

    static void close(JInternalFrame jInternalFrame) {
        if(((Interna)jInternalFrame).getContent().getClass().equals(AgregarDatos.class)){
            inicio();
        }
    }


    public void messageReceived(int to, net.tinyos.message.Message m) {
        PaqueteMIG recibido = (PaqueteMIG) m;
```

```
        String medicoDeTurno=getMedicoDeTurno();
        System.out.println("Medico de Turno "+medicoDeTurno);
        String numero=script.getProperty(medicoDeTurno);
        System.out.println("Extension "+numero);

        /**
         * Informacion del paciente
         */
        System.out.println("paquete "+m.toString());
        if(numero!=null)
            script(numero,"p"+recibido.get_pac()+""+recibido.get_data(),"s1");
        System.out.println("p"+recibido.get_pac()+""+recibido.get_data());
        try{
            Runtime.getRuntime().exec(new String[]{"/root/tinyos-1.x/tools/java/prueba1.sh"});
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    private void script(String numero,String contexto,String extension){
        try{
            FileOutputStream fou=new FileOutputStream("salida.call");
            fou.write(("Channel: SIP/"+numero+"\nMaxRetries: 1\nRetryTime: 60\nWaitTime: 30\nContext:
"+contexto+"\nExtension: "+extension+"\nPriority: 1").getBytes());
            fou.flush();
            fou.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    // End of variables declaration//GEN-END:variables

}
```

PARTE 2- CÓDIGO NESC

La aplicación de nesC del telosB transmisor y del telosB receptor se diferencia en la identificación del destino, por lo tanto se documentará para uno de los dos.

2.1 SensorTxM.nc

```
includes paquetes;

module PracticaM {
        provides {
        interface StdControl;
        }


        uses {

            interface SendMsg;
            interface ReceiveMsg;
            interface StdControl as SubControl;
```

```
        interface Leds;
    }
}

implementation {
                        TOS_Msg envio;


result_t sendPaquete (uint16_t destino , uint16_t tipo_pkt , uint16_t paciente , uint16_t data, uint8_t sala ,
uint8_t band ){

        p_paquete nuevo;
                nuevo =(p_paquete)envio.data;
                nuevo->type = tipo_pkt;
                nuevo->pac = paciente;
                nuevo->data = data;
                nuevo->sala = sala;
                nuevo->band = band;
                call Leds.redOn();

                if ((call SendMsg.send(destino,  sizeof(paquete) , &envio )== SUCCESS)){
                  return SUCCESS;
                }

        return FAIL;
}

        command result_t StdControl.init() {

                call SubControl.init();
                call Leds.init();

    return SUCCESS;
        }

        command result_t StdControl.start() {

                call SubControl.start();
                return SUCCESS;
        }

        command result_t StdControl.stop() {

                call SubControl.stop();
                return SUCCESS;
        }


event result_t SendMsg.sendDone(TOS_MsgPtr msg, result_t success){

        p_paquete pkt;
        pkt = (p_paquete)&msg->data;
         call Leds.redOff();

        return SUCCESS;
}
```

```
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr men) {

        p_paquete pkt;
        int d;
         int s;
         pkt = (p_paquete)&men->data;
        call Leds.redOn();
         d = pkt->data;
        s = pkt->sala;

                if (pkt->type == PETICION){

           switch (pkt->pac){
              case PACIENTE_1:
                                sendPaquete( 45 ,1  , 1 , d, 1, 0);
                                        call Leds.yellowOn();
                                break;

               case PACIENTE_2:

                                sendPaquete(45, 1 , 2 ,d , 1,0);
                                        call Leds.greenOn();
                                break;


              case PACIENTE_3:

                                sendPaquete(45, 1 , 3 , d, 1,0);
                                        call Leds.yellowOff();
                                break;


              case PACIENTE_4:

                                sendPaquete( 45, 1 , 4 , d, 1,0);
                                        call Leds.greenOff();
                                break;

             default:
                    break;
           }
        }else{

        switch (pkt->pac){


              case PACIENTE_1:
                                sendPaquete( TOS_UART_ADDR,1  , 1 , d, 1,0);
                                                call Leds.yellowOn();
                                break;

               case PACIENTE_2:

                                sendPaquete(TOS_UART_ADDR, 1 , 2 , d, 1,0);
                                                call Leds.yellowOn();
                                break;

              case PACIENTE_3:
```

```
                                sendPaquete(TOS_UART_ADDR, 1 , 3 , d, 1,0);
                                        call Leds.yellowOff();
                                break;


                    case PACIENTE_4:

                                sendPaquete( TOS_UART_ADDR, 1 , 4 , d, 1,0);
                                        call Leds.yellowOff();
                                break;

                    default:
                            break;
                    }
                }
                return men;
    }
}
```

2.2 SensorTx.nc

```
includes paquetes;


// Configuración de la interconexion
configuration Practica {
}
// Veamos como vamos a interconexionar los componentes o módulos

implementation {

// Componentes que se van a utilizar en nuestra aplicación
  components Main, // Es el componente de arranque
                        PracticaM, // Es nuestro modulo de enrutamiento
                        TimerC, //Proporciona temporizadores
                        GenericComm,
                        LedsC,
                        QueuedSend; // Una cola de envio


  Main.StdControl ->   PracticaM.StdControl;
  Main.StdControl ->  QueuedSend.StdControl ;

  PracticaM.SubControl -> GenericComm;
  PracticaM.SendMsg      -> QueuedSend.SendMsg[PAQUETE];
  PracticaM.ReceiveMsg   -> GenericComm.ReceiveMsg[PAQUETE];
  PracticaM.Leds -> LedsC.Leds;

}
```

2.3 paquetes.h

```
enum{
        PAQUETE = 17
};
```

```
enum{
    AM_PAQUETE = 17
   };

// Diferentes tipos de paquetes que va a tener el algoritmo
enum {
        PETICION = 1,
        RESPUESTA = 2,
        ERROR = 3
};

enum {
        SALA_1 = 1,
        SALA_2 = 2,
        SALA_3 = 3
};

enum{
        PACIENTE_1 = 1,
        PACIENTE_2 = 2,
        PACIENTE_3 = 3,
        PACIENTE_4 = 4

};
enum{
        TAQUI = 1,
      BRADI = 2,

};

// Estrucuta de un paquete
typedef struct paquete {
        uint8_t type;
        uint8_t pac;
        uint16_t data;
        uint8_t band;
        uint8_t count;
        uint8_t sala;
} __attribute__ ((packed)) paquete;

typedef paquete *p_paquete; // Una variable que representa un puntero a un paquete
```

2.4 SensorRXPruebaM.nc

Esto módulo es el utilizado por el dispositivo telosB receptor para realizar las pruebas permitiendo enviar un paquete cada 10ms al telosB transmisor hasta llegar a 200 paquetes.

```
includes paquetes;
module PracticaM {

      provides {
         interface StdControl;
      }
```

```
    uses {
        interface Timer;
        interface SendMsg;
        interface ReceiveMsg;
        interface StdControl as SubControl;
        interface Leds;
    }
}

implementation {

        TOS_Msg envio;
        uint16_t paciente;
        uint16_t dato;
        uint8_t c;

result_t sendPaquete (uint16_t destino , uint16_t tipo_pkt , uint16_t paciente , uint16_t data, uint8_t sala ,
uint8_t band ){

        p_paquete nuevo;
                nuevo =(p_paquete)envio.data;
                nuevo->type = tipo_pkt;
                nuevo->pac = paciente;
                nuevo->data = data;
                    nuevo->sala = sala;
                nuevo->band = band;
                if ((call SendMsg.send(destino,  sizeof(paquete) , &envio )== SUCCESS)){
                  return SUCCESS;
                }

    return FAIL;
}

        command result_t StdControl.init() {

                call SubControl.init();
                call Leds.init();

    return SUCCESS;
        }

        command result_t StdControl.start() {

            call SubControl.start();

                return SUCCESS;

        }

        command result_t StdControl.stop() {

                call SubControl.stop();
                return call Timer.stop();

        }

        event result_t Timer.fired(){
```

```
                call Leds.redOn();
                 switch (paciente){
            case PACIENTE_1:
                        if (c < 200){
                                call Leds.yellowOn();
                                c=c+1;
                                sendPaquete( 9 ,2  , 1 , dato, 1,c);
                                                        }
                         break;

            case PACIENTE_2:
                         if (c < 200){
                                c=c+1;
                                sendPaquete( 9 ,2  , 2 , dato, 1,c);
                                                        }
                         break;

            case PACIENTE_3:
                         if (c < 200){
                                c=c+1;
                                sendPaquete( 9 ,2  , 3 , dato, 1,c);
                                                        }
                        break;



            case PACIENTE_4:
                        if (c < 200){
                                c=c+1;
                                sendPaquete( 9 ,2  , 4 , dato, 1,c);
                                                        }
                         break;


           default:
                         break;

          }

         if(c == 200){
                dato=0;
                return FAIL;
                }
          return SUCCESS;

}

event result_t SendMsg.sendDone(TOS_MsgPtr msg, result_t success){

         p_paquete pkt;
         pkt = (p_paquete)&msg->data;
         call Leds.redOff();

         return SUCCESS;
}

event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr men) {
```

```
        p_paquete pkt;
        pkt = (p_paquete)&men->data;
        call Leds.redOn();
        dato = pkt->data;
        c=0;

        if (pkt->type == PETICION){

           switch (pkt->pac){
            case PACIENTE_1:
                            paciente = pkt->pac;
                            call Timer.start(TIMER_REPEAT , 10);

                            break;

           case PACIENTE_2:

                            paciente = pkt->pac;
                            call Timer.start(TIMER_REPEAT , 10);

                            break;

           case PACIENTE_3:

                           paciente = pkt->pac;
                            call Timer.start(TIMER_REPEAT, 10);

                           break;

              case PACIENTE_4:

                           paciente = pkt->pac;
                            call Timer.start(TIMER_REPEAT, 10);

                            break;

             default:
                           break;
           }


        }

        return men;
    }
}
```

2.5 SensorRxPrueba.nc

```
includes paquetes;
// Configuración de la interconexion
configuration Practica {
}

// Veamos como vamos a interconexionar los componentes o modulos

implementation {
```

```
// Componentes que se van a utilizar en nuestra aplicación
  components Main, // Es el componente de arranque
                     PracticaM, // Es nuestro modulo de enrutamiento
                     TimerC, //Proporciona temporizadores
                     GenericComm,
                     LedsC,
                     QueuedSend; // Una cola de envio


  Main.StdControl ->   PracticaM.StdControl;
  Main.StdControl ->  QueuedSend.StdControl ;
  Main.StdControl -> TimerC;

  PracticaM.SubControl -> GenericComm;
  PracticaM.SendMsg      -> QueuedSend.SendMsg[PAQUETE];
  PracticaM.ReceiveMsg   -> GenericComm.ReceiveMsg[PAQUETE];
  PracticaM.Leds -> LedsC.Leds;
  PracticaM.Timer -> TimerC.Timer[ unique("Timer") ];

}
```

PARTE 3- ASTERISK

3.1 sip.conf

Se crearon 4 usuarios, con número de extensión 2151,2116, 2114 y 2115, pero se puede crear más usuarios de la misma manera.

```
[general]
port = 5060

[2151]                  ;login del usuario
secret=123              ;clave del usuario
type=friend             ;Puede enviar y recibir llamadas
context=default         ;Contexto al que pertenece en extensions.conf
host=dynamic            ;Puede acceder desde cualquier direccion ip
allow=all               ;Permite a todos los numeros que lo llaman

[2116]                   ;login del usuario
secret=123              ;clave del usuario
type=friend             ;Puede enviar y recibir llamadas
context=default         ;Contexto al que pertenece en extensions.conf
host=dynamic            ;Puede acceder desde cualquier direccion ip
allow=all               ;Permite a todos los numeros que lo llaman

[2114]                  ;login del usuario
secret=123              ;clave del usuario
type=friend             ;Puede enviar y recibir llamadas
context=default         ;Contexto al que pertenece en extensions.conf
host=dynamic            ;Puede acceder desde cualquier direccion ip
allow=all               ;Permite a todos los numeros que lo llaman

[2115]                  ;login del usuario
secret=123              ;clave del usuario
type=friend             ;Puede enviar y recibir llamadas
```

```
context=default         ;Contexto al que pertenece en extensions.conf
host=dynamic            ;Puede acceder desde cualquier direccion ip
allow=all               ;Permite a todos los numeros que lo llaman
```

3.2 extensions.conf

La extension pxy hace referencia a: “x “es el paciente y “y” al problema que tiene el paciente en ese momento; puede ser taquicardia y bradicardia. Los números 301a, 302a, 303a, 304a hacen referencia al mensaje pregrado de los pacientes y taqui y bradi al problema.

```
[general]
static=yes
writeprotect=yes

[p11]
exten => s1,1,Background(301a)
exten => s1,2,goto(taqui,s,1)

[p12]
exten => s1,1,Background(301a)
exten => s1,2,goto(bradi,s,1)


[p21]
exten => s1,1,Background(301b)
exten => s1,2,goto(taqui,s,1)

[p22]
exten => s1,1,Background(301b)
exten => s1,2,goto(bradi,s,1)

[p31]
exten => s1,1,Background(301c)
exten => s1,2,goto(taqui,s,1)

[p32]
exten => s1,1,Background(301c)
exten => s1,2,goto(bradi,s,1)

[p41]
exten => s1,1,Background(301d)
exten => s1,2,goto(taqui,s,1)

[p42]
exten => s1,1,Background(301d)
exten => s1,2,goto(bradi,s,1)


[bradi]
exten => s,1,Background(bradi)
exten => s,2,Hangup

[taqui]
exten => s,1,Background(taqui)
exten => s,2,Hangup

[default]
exten=2116,1,Dial(SIP/2116,10)
```

exten=2151,1,Dial(SIP/2151,10)
exten=2114,1,Dial(SIP/2114,10)
exten=2115,1,Dial(SIP/2114,10)

ANEXO D. METODOLOGÍA

Para la realización de este proyecto se dividió su ejecución en dos partes:

- Investigativa
- Implementación

Para la primera parte se tomo como referencia el Modelo de Investigación Documental (M.I.D)[28], el cual proporcionó una serie de lineamientos que permitieron cambios para adaptarse a las características específicas del proyecto, las fases a seguir lo indican la tabla D.1

Tabla D.1 Fases M.I.D

FASES DE REFERENCIA	CARACTERÍSTICAS
Fase Preparatoria	Se identificaran de manera clara la temática seleccionada y los núcleos temáticos que componen el contenido central.
Fase Descriptiva	Revisión detallada de cada una de las unidades de análisis
Fase de Construcción Teórica Global	Se presentaran los resultados obtenidos en el estudio del área temática, logros, identificación de vacíos, limitaciones y estado actual de la investigación.
Fase de Extensión y Publicación	Divulgación de los resultados obtenidos.

Para la implementación se opto por el Modelo de Prototipo Evolutivo[29], el cual permitió diseñar y construir las partes más importantes, que posteriormente se ajustaron y ampliaron hasta la terminación.

Se eligió este modelo porque presenta ventajas en la generación de signos visibles de progreso y la modificación sobre la marcha y sumadas las ventajas del modelo anterior, se consideraron varias opciones a nivel teórico antes de comenzar a construir el prototipo.

Es necesario aclarar que la ejecución de de los dos modelos se hizo en forma simultanea. A continuación se enmarcan las etapas generadas dentro de la

realización de proyecto agrupando las fases de los modelos mencionados anteriormente.

Etapa de planeación y capacitación.

1. Recolección de información.
   - Estado del arte de la Redes de Sensores Inalámbricas
   - Investigación del estándar 802.15.4 y especificación ZigBee para el desarrollo del trabajo de grado para la comunicación hacia la unidad central.
   - Búsqueda de los equipos y herramientas hardware y software que incorporen ZigBee para realizar la comunicación hacia la unidad central.
   - Recolección de la información sobre la frecuencia cardiaca para su caracterización.
2. Alcances de la PBX VoIP (Asterisk) para la comunicación de los mensajes audibles.

Etapa de análisis y síntesis de la información.

3. Análisis de las capacidades de la PBX VoIP para su funcionamiento como unidad central.
4. Análisis de las capacidades y especificaciones para la elección de las herramientas hardware y software ZigBee encontradas que satisfacen la construcción de comunicación.
5. Análisis de la frecuencia cardiaca para realizar su caracterización mediante el ritmo cardíaco.

Etapa de diseño.

6. Construcción de modelos teóricos.
   - Modelo de una aplicación para el piloto transporte del ritmo cardíaco con el estándar ZigBee hacia una unidad central y el enrutamiento de los mensajes.
7. Recolección y análisis de requerimientos.
   - Requerimiento de fiabilidad de datos transmitidos.
   - Requerimientos de bajo consumo de potencia.
   - Requerimientos de enrutamiento.
8. Primera aproximación de la Aplicación.
   - Diseño de la aplicación.
   - Adaptación de los equipos para los servicios específicos.
   - Desarrollo de la solución.

Etapa de Elaboración de la Aplicación.

9. Dimensionamiento adaptado a las herramientas existentes.

10. Plan de pruebas y validación en el montaje de la aplicación.
11. Puesta en marcha del piloto transporte de Ritmo Cardíaco hacia la unidad central y el enrutamiento de la información.
12. Culminación de la monografía (en cada una de las etapas se desarrollo el documento) y paper.