Redes Inalámbricas de Sensores con características de tiempo real como herramienta para la Monitorización de Constantes Vitales, en el Marco del Proyecto MERIS

ANEXO

Ing. Carolina Ríos Fuentes

Director:

Magíster. Ing. Héctor Fabio Jaramillo O.

Universidad del Cauca
Instituto de Postgrados de Ingeniería Electrónica y Telecomunicaciones IPET
Maestría en Ingeniería área Telemática
Servicios Avanzados de Telecomunicaciones
Popayán, Octubre de 2009

Tabla de contenido

Lista de figuras

Lista de Tablas

Introducción

Este documento contiene, la lista de elementos de montaje superficial que son requeridos para la fabricación de la placa de sensado y la de procesamiento diseñadas para el CEI, además la asignación de pines del conector derecho e izquierdo del nodo CEI y los esquematicos de la placa de sensado y procesamiento; finalmente se presenta el código fuente del nodo IRIS y la interfaz de datos que corre sobre la estación base.

1.1 Lista de Elementos y asignación de pines de conectores

En la tabla 1, se listan los elementos requeridos para la fabricación de la placa de sensado y procesamiento diseñadas, con su respectivo precio; en la cual se puede apreciar que los costos más elevados son los referenciados para los sensores y que los bloques de adaptación para estos tiene un valor aproximado de 60 dólares. De los elementos requeridos para la placa de procesamiento el microcontrolador es el más costoso.

Componentes por placa	Referencia	Precio 1-9 (€)	Nº unidades	Total (€)
Placa de Sensores				
Sensor Adult Finger Clip		168	1	168,000
conector Nellcor (DB9)			1	0,000
Sensor Thermal airflow		130	1	130,000
Conector safety			1	0,000
Transistor BC856A	Farnell 1459042RL	0,156	2	0,312
Transistor MMBT2222	Farnell 1467896	0,031	2	0,062
Ampli Operacional TLV2254	Farnell 8454965	2,4	1	2,400
Conectores Socket	Farnell 1098346	2,65	2	5,300
Condensadores 10uF	Farnell 1458902	0,97	1	0,970
Condensadores 3pF	Farnell 7568223	0,34	1	0,340
Condensadores 47pF	Farnell 7568380	0,2	1	0,200
Condensadores 0,1uF	Farnell 317287	0,056	1	0,056
Condensadores 1uF	Farnell 1327676	0,113	2	0,226
Resistencias 0R	Farnell 9233130	0,048	1	0,048
Resistencias 1k	Farnell 9233385	0,048	6	0,288
Resistencias 15k	Farnell 9233520	0,024	3	0,072
Resistencias 5k	Farnell 9331301	0,034	1	0,034
Resistencias 100k	Farnell 9233628	0,048	2	0,096
Resistencias 150k	Farnell 9233644	0,048	1	0,048
Resistencias 180k	Farnell	0	1	0,000
Resistencias 47k	Farnell	0	1	0,000
Resistencias 5M	Farnell	0	1	0,000
TOTAL				**308,45**
Placa procesamiento				
Micro MSP430FG439	Farnell 1555272	15	1	15,000
Pulsador KMS	Farnell 4156298	0,33	2	0,660
Cristal 32.768kHz 3.3V	Farnell 1457182	1,32	1	1,320
Conector Header	Farnell 1098336	2,66	2	5,320
Conectores Socket	Farnell 1098346	2,65	2	5,300
Pin Header 90º 36way	Farnell 1097956	1,6	1	1,600
Condensadores 0,1uF	Farnell 317287	0,056	6	0,336
Condensadores 1uF	Farnell 1327676	0,113	2	0,226
Condensadores 10uF	Farnell 1458902	0,97	2	1,940
Resistencias 0R	Farnell 9233130	0,048	1	0,048
Resistencias 100R	Farnell	0	2	0,000
Resistencias 100K	Farnell 9233628	0,048	1	0,048
Resistencias 514k	Farnell	0	1	0,000
TOTAL				**31,80**
			TOTAL	**340,25**

Tabla 1 Lista de elementos y su costo

En la tabla 2, se presentan las señales a las que van conectadas los conectores de cada uno de los sensores y en las tablas 3 y 4 se presentan la asignación de pines utilizados para el conector izquierdo y derecho de las placas del nodo CEI respectivamente.

PIN	CONECTOR	
	PULSOXIMETRO	TERMOCUPLA
1	AK *Out Rojo*	Out Termocupla
2	KA *Out Infrarojo*	Tierra
3	Cátodo del Diodo PIN	No existe
4	Ánodo del Diodo PIN	No existe
5	Tierra	No existe

Tabla 2 Descripción de pines de los conectores termocupla y pulsoxímetro

HEADER	SOKET	CONECTADO A	HEADER	SOKET	CONECTADO A
60	1	Tierra analógica	1	60	Tierra analógica
59	2	Alimentación analógica 3.3 v	2	59	Alimentación analógica 3.3 V
58	3	Cátodo Diodo PIN, Pin 77µ	3	58	
57	4	AO_0, Pin 76µ	4	57	Intensidad LED rojo Pin 10µ
56	5	Ánodo Diodo PIN, Pin 75µ	5	56	Activa LED rojo Pin 56µ
55	6	Salida AO_1, Pin 2µ	6	55	Activa LED infrarrojo Pin 57µ.
42	19	Vref, Pin 7µ	19	42	
41	20	Rx USART, Pin 54µ	20	41	
40	21	Tx USART, Pin 55µ	21	40	
38	23	Entrada AO_1, Pin 3µ	23	38	
37	24	Señal Termo Amplificada (AO 0). Pin 4µ.	24	37	
36	25	Intensidad LED Infrarrojo Pin 5µ	25	36	
32	29	Alimentación digital 3.3 V	29	32	Alimentación digital 3.3 V
31	30	Tierra digital	30	31	Tierra digital

Tabla 3 Pines del conector izquierdo placa final

HEADER	SOKET	CONECTADO A	HEADER	SOKET	CONECTADO A
60	1	Tierra digital	1	60	Tierra digital
59	2	Alimentación digital 3.3 V	2	59	Alimentación digital 3.3 V
58	3	Alimentación digital 1.2 V	3	58	Alimentación digital 1.2 V
57	4	Alimentación digital 5V	4	57	Alimentación digital 5 V
34	27	Alimentación digital -5V	27	34	Alimentación digital -5 V
33	28	Alimentación digital 2.5 V	28	33	Alimentación digital 2.5 V
32	29	Alimentación digital 3.3 V	29	32	Alimentación digital 3.3 V
31	30	Tierra digital	30	31	Tierra digital

Tabla 4 Pines del conector derecho placa final

1.2 Esquematicos placa de sensado y procesamiento

En la figura 1, se presenta el esquematico de la placa de sensado; y en las figuras 2 y 3 el esquematico de la placa de procesamiento, esta se dividió en dos partes para facilitar su visaulización, la primera parte muestra las conecxiones entre los concentores de la placa y la segunda los elementos que se requieren para el funcionamiento del microcontrolador.

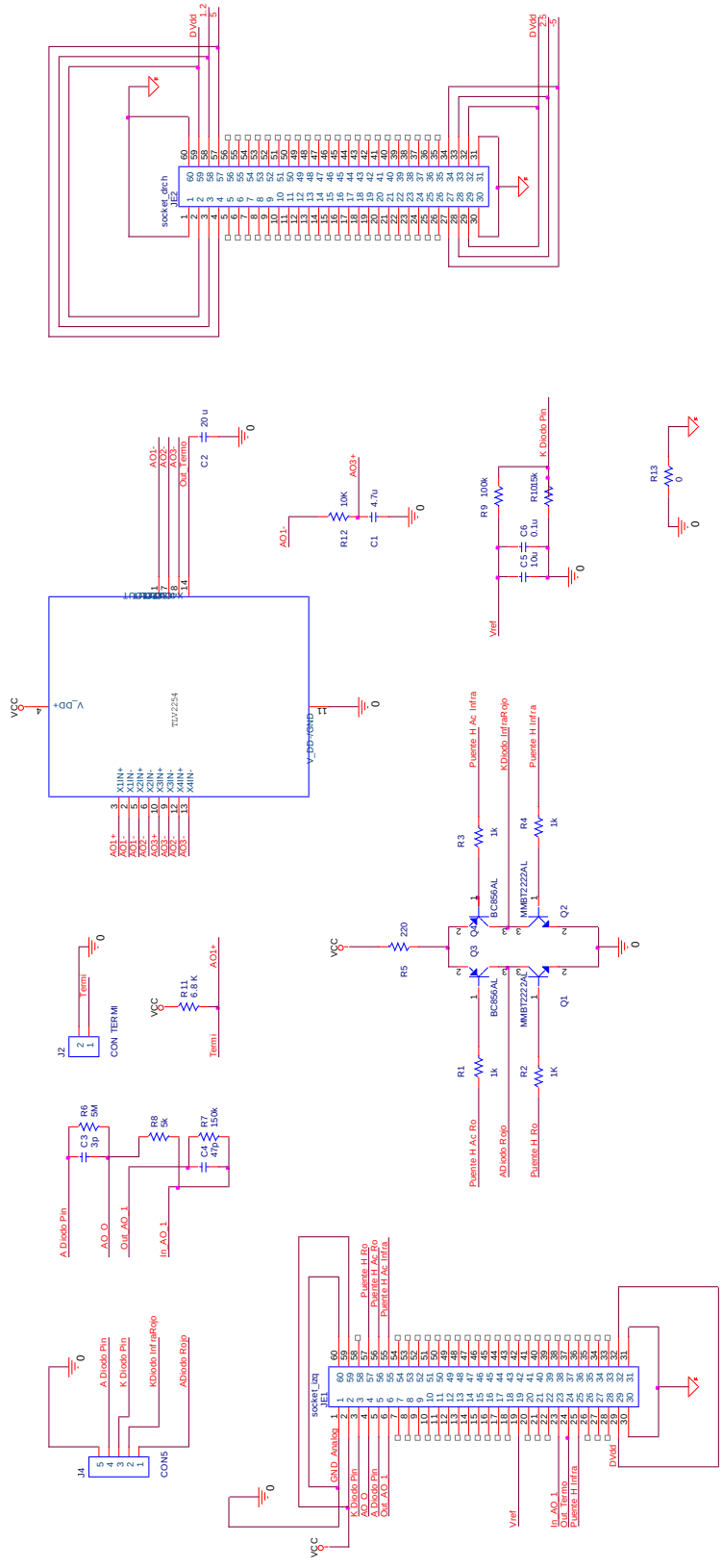

Fig. 1 Esquemático placa de sensado

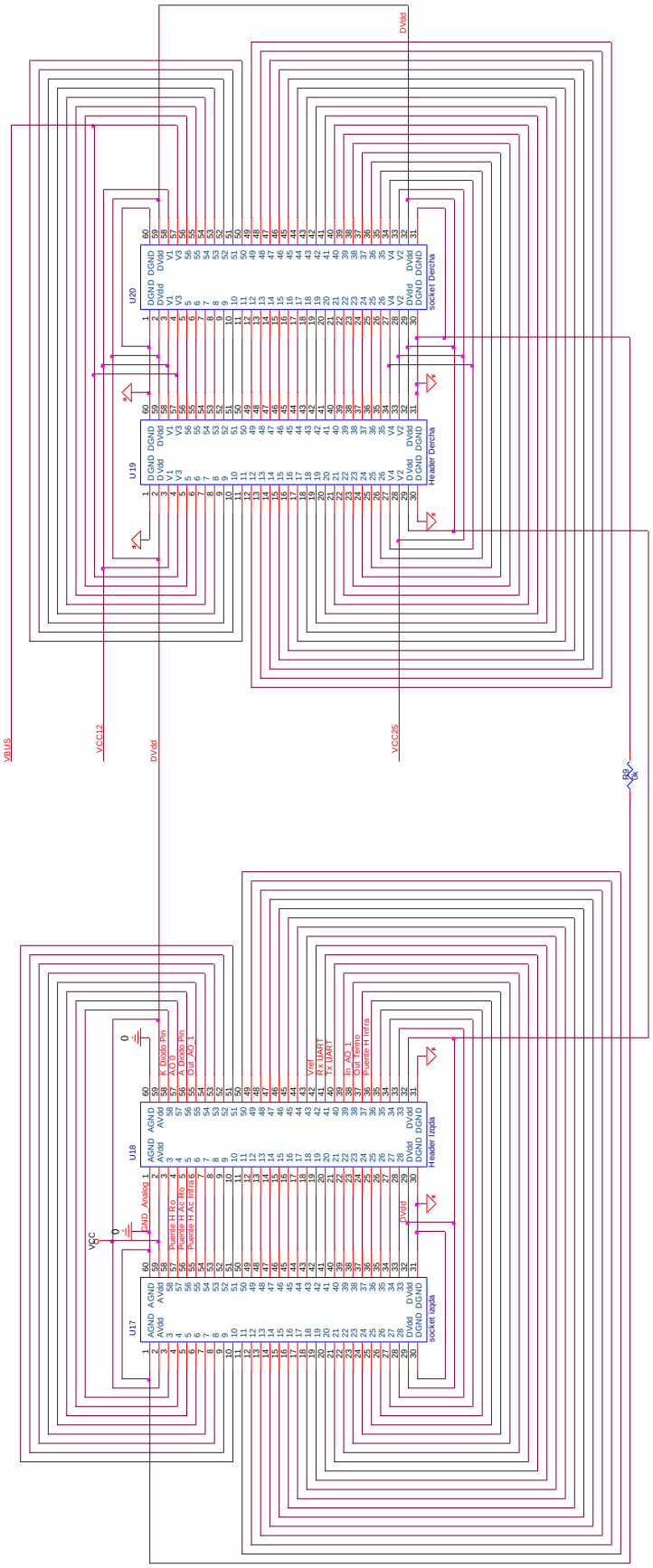

Fig. 2 Esquemático de los concentores de la placa de procesamiento

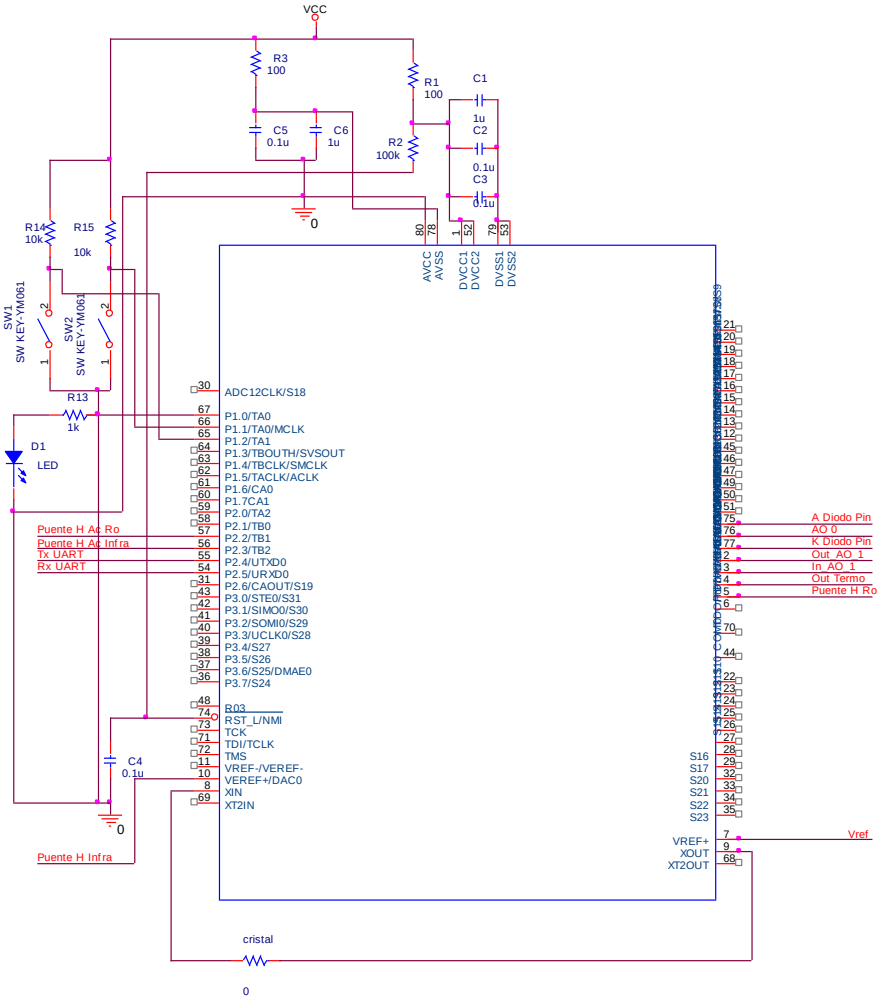

Fig. 3 Esquemático del microcontrolador de la placa de procesamiento

A continuación se presenta el código fuente desarrollado para la WSN de nodos IRIS.

2 Código fuente

En esta sección se presenta el código fuente de los nodos de la red y la aplicación que corre en estación base desarrollada en este trabajo de maestría.

2.1 Nodo

El código desarrollado se divide en cuatro partes:

- El Makefile que aunque este no es un archivo nesC, contiene las reglas de compilación así como las herramientas que se utilizan para poder crear los archivos tipo Java y C; que facilitan el intercambio de datos entre el nodo y el PC.
- Un archivo de cabecera denominado Meris.h, en el cual se definieron todas las estructuras de datos que utiliza la WSN y diversas constantes que utiliza la aplicación, como los tipos de paquetes, tiempos de generación de mensajes, tiempo de la supertrama, tiempo en milisegundos de el periodo de activación e inactivación, entre otros.

- Un archivo de configuración denominado MerisAPPC en este se especifican todos los componentes que se van a utilizar la aplicación y la definición de las relaciones que hay entre las interfaces que utiliza la aplicación y las interfaces que proporcionan los componentes.

- Un archivo módulo llamado MerisC, en el cual se implementa el código de la aplicación y se definen cuales son las interfaces que va a proporcionar el componente.

A continuación se presenta el código para cada uno de estos archivos.

2.1.1 Makefile

```
COMPONENT=MerisAppC
BUILD_EXTRA_DEPS = Neighbor.java GatewayMsg.java ManagmentMsg.java
MerisMonitorPacket.java
CLEAN_EXTRA = *.java

CFLAGS += -I$(TOSDIR)/lib/T2Hack

Neighbor.java: Meris.h
	mig java -target=$(PLATFORM) $(CFLAGS) -java-classname=model.Neighbor
Meris.h Neighbor -o $@

GatewayMsg.java: Meris.h
	mig java -target=$(PLATFORM) $(CFLAGS) -java-classname=model.GatewayMsg
Meris.h GatewayMsg -o $@

ManagmentMsg.java: Meris.h
	mig java -target=$(PLATFORM) $(CFLAGS) -java-
classname=model.ManagmentMsg Meris.h ManagmentMsg -o $@

MerisMonitorPacket.java: Meris.h
	mig java -target=$(PLATFORM) $(CFLAGS) -java-
classname=model.MerisMonitorPacket Meris.h MerisMonitorPacket -o $@

include $(MAKERULES)
```

2.1.2 Meris.h

```
#ifndef MERIS_H
#define MERIS_H

enum {
	MIN_RSSI = 2, /* Valor RSSI Minimo que puede tener el nodo de destino */

	AM_MERIS = 6, /* Indicador de tipos de Paquetes de InformaciÃ³n */
	AM_MANAGMENT = 7,   /* Indicador de tipos de Paquetes de GestiÃ³n */

	MAGICS_LEN = 10,    /* Tamaño del buffer para los numeros mágicos */
	UART_BUFFER_LEN = 20, /* Tamaño del buffer para el UART */
	AM_MERIS_SERIAL_MSG = 0x89, /* Tipo de paquete BROADCAST para UART */

	NEIGHBORHOOD_LEN = 7,/* Tamaño máximo del Vecindario (7 Nodos Iris
disponibles) */
	GATEWAY_ADDR = 0,         /* DirecciÃ³n del nodo Gateway */

	TIMER_PERIOD_MILLI = 2000, /* Perido para el envio de paquetes BEACON */
	CICLES_CONF_UPDATE = 3, /* # beacons generados antes de una reasignación
de tiempos de trama */

	PERCENTAGE_MILLI = 20, /* tiempo en milisegundo por unidad de
porcentaje, en promedio se necesita un 20 msg para el envio exitoso de un
paquete */
```

```
	TIMER_MONITOR_MILLI = 500, /* Perido para el envio de paquetes de
informaciÃ³n a la Red */
	RT_BUFFER_LEN = 20, /* TamaÃto del buffer RT de Radio*/

	AM_NEIGHBOR = 0x70,
	AM_GATEWAYMSG = 0x71,
	AM_MANAGMENTMSG=0x72,/* identificador del tipo de paquete para gestión
*/
	AM_MERISMONITORPACKET=0x73
};

/* Plazos y Prioridades de paquete */
enum {
	HI_PRIORITY = 3,
	HI_PRIORITY_ELAPSE = 4000, /* 99%*Act_Period*#Rangos = 99%*2000*2 = 4000
MILLI */

	MIDDLE_PRIORITY=2,
	MIDDLE_PRIORITY_ELAPSE = 6000,

	LOW_PRIORITY = 1,
	LOW_PRIORITY_ELAPSE = 8000
};

/* Estructura para Paquetes de GestiÃ³n */
typedef nx_struct ManagmentMsg {

  nx_uint16_t magic;	/* Evita la duplicidad de envios en la disiminaciÃ³n
*/
  nx_uint8_t hops;	/* Contador de saltos */
  nx_uint32_t timestamp;/* Tiempo local al envio del paquete */

  nx_uint16_t addr;   /* Nodo a configurar */
  nx_uint8_t start;   /* Orden en la transmisiÃ³n */
  nx_uint8_t stop;    /* Porcentaje del tiempo que puede transmitir */
  nx_uint8_t rssi;  /*Potencia de paquete recibida en la última comunicación */

} ManagmentMsg;

typedef nx_struct Configuration{
	nx_uint8_t start; /* Contador de paquetes de informaciÃ³n recibidos
durante el Ãºltimo hiperperiodo */
	nx_uint8_t stop; /* Porcentaje de aportaciÃ³n del nodo a la carga de la
red  */
} Configuration;

//Estructura para almacenar la informaciÃ³n de Nodos Vecinos
typedef nx_struct Neighbor {
	nx_uint16_t addr;   /* Id Nodo */
	nx_uint8_t hops;    /* Contador de saltos hasta la gateway */
	nx_uint8_t lqi;     /* Indicador de la calidad de enlace con el nodo */
	nx_uint8_t rssi;    /* Indicador de Potencia del Ãºltimo mensaje */
	nx_int32_t offset;  /* Tiempo de recepciÃ³n - estampa del paquete */
	nx_uint16_t evictor; /* Indicador de actividad del Nodo */
	nx_uint16_t cpkg;   /* Contador de paquetes de informaciÃ³n recibidos
durante el Ãºltimo hiperperiodo */
	nx_uint8_t start;   /* Contador de paquetes de informaciÃ³n recibidos
durante el Ãºltimo hiperperiodo */
	nx_uint8_t stop;  /* Porcentaje de aportación del nodo a la carga de la
red  */
} Neighbor;

/*-- Estructura del Paquete Usb (Gateway -(Usb)-> PC ) --*/
typedef nx_struct GatewayMsg {
	nx_uint16_t source;	/* DirecciÃ³n del nodo de procedencia */
```

```
	nx_uint16_t destination;  /* DirecciÃ³n del nodo destino */
	nx_uint32_t counter;
	nx_uint8_t rssi;
	nx_uint8_t quality;
	nx_uint8_t power;
	nx_uint8_t hops;    /* Hops */
} GatewayMsg;

/* Estrutura para Paquetes de InformaciÃ³n */
typedef nx_struct MerisMsg {
	nx_uint16_t nodeid;
	nx_uint16_t counter;
} MerisMsg;

/* Paquetes de Monitoreo */
typedef nx_struct MerisMonitorPacket {
	nx_uint8_t SaO;                  /* SaturaciÃ³n de Oxigeno */
	nx_uint8_t FrC;                  /* Frecuencia Cardiaca */
	nx_uint8_t FrR;                  /* Frecuencia Respiratoria */
	nx_uint8_t Triage; /* EstimaciÃ³n global del estado de Salud "Triage" */
	nx_uint8_t priority;       /* Prioridad del paquete */
	nx_uint32_t timestamp;/* Tiempo actual + offset registrado del vecino */

	nx_uint16_t source;        /* DirecciÃ³n del nodo de origen */
	nx_uint16_t destination; /* DirecciÃ³n del nodo destino */
} MerisMonitorPacket;

#endif
```

2.1.3 MerisAPPC

```
#include <Timer.h>
#include "Meris.h"

configuration MerisAppC {
}

implementation {
	components MerisC as App;
	components MainC;
	App.Boot -> MainC;
	components LedsC;
	App.Leds -> LedsC;
	components new TimerMilliC() as Timer0;
	App.Timer0 -> Timer0;
	components new TimerMilliC() as TimerTxStart;
	App.TimerTxStart -> TimerTxStart;
	components new TimerMilliC() as TimerTxStop;
	App.TimerTxStop -> TimerTxStop;

	/* Componente para el manejo del Envio de Radio*/
	components new AMSenderC(AM_MERIS);
	App.RadioPacket -> AMSenderC;
	App.AMPacket -> AMSenderC;
	App.AMSend -> AMSenderC; /* Interfaz para el envió de paquetes de
Información */

	components new AMReceiverC(AM_MERIS);
	App.Receive -> AMReceiverC; /* Interfaz para el manejo de la Recepcion
de paquetes de InformaciÃ³n */

	components new AMSenderC(AM_MANAGMENT) as AMManagmentSenderC;
	App.AMManagmentSend -> AMManagmentSenderC;    /* Interfaz para el envio
de paquetes de GestiÃ³n */

	components new AMReceiverC(AM_MANAGMENT) as AMManagementReceiverC;
```

```
	App.ManagmentReceive -> AMManagementReceiverC; /* Interfaz para el
manejo de la Recepcion de paquetes de GestiÃ³n */

	/* Captura del RSSI de los paquetes */
	components RF230ActiveMessageC;
	App.PacketRSSI -> RF230ActiveMessageC.PacketRSSI;
	App.PacketLinkQuality -> RF230ActiveMessageC.PacketLinkQuality;
	App.PacketTransmitPower -> RF230ActiveMessageC.PacketTransmitPower;
	App.LowPowerListening -> RF230ActiveMessageC.LowPowerListening;
	components ActiveMessageC;
	App.AMControl -> ActiveMessageC;

	/* Interfaces para el manejo de la comunicacion USB */
	components SerialActiveMessageC as Usb;
	App.UsbPacket -> Usb;
	App.UsbControl -> Usb;
	/* Interfaz de Control USB */
	/* App.UsbSend -> Usb.AMSend[AM_MERIS_SERIAL_MSG];  Interfaz de envio
por USB */
	App.UsbSend -> Usb.AMSend; /* Interfaz de envio por USB */
	App.UartAMPacket -> Usb;

	components LocalTimeMilliC;
	App.LocalTime -> LocalTimeMilliC; /* Permite recuperar el valor del
tiempo del nodo */

	/* INTERFACES DE LA APLICACIÃ‘N: MERIS */
	components new TimerMilliC() as TimerMonitor;
	App.TimerMonitor -> TimerMonitor; /* Genera los eventos de envio de
informaciÃ³n a la Red*/

	components RandomC;
	App.Random -> RandomC; /* Generacion seudoaletoria de valores para
simulación */
}
```

2.1.4 MerisC

```
#include <Timer.h>
#include "Meris.h"

module MerisC {

	uses interface Boot; /* Manejo primario del flujo de la aplicaciÃ³n */
	uses interface Leds;             /* Manejo de LEDS */
	uses interface Timer<TMilli> as Timer0; /* Timer de mantenimiento de la
red */
	uses interface Timer<TMilli> as TimerTxStart; /* SeÃ±ala el tiempo para
Transmitir a la red */
	uses interface Timer<TMilli> as TimerTxStop;  /* SeÃ±ala el tiempo para
Transmitir a la red */
	uses interface Packet as RadioPacket;         /* GestiÃ³n en memoria del
Paquete de Radio */
	uses interface AMPacket;                                /* Incluye comando
adicionales relacionados con los paquetes de radio */
	uses interface AMSend;                                          /* Clase
control especializada en comandos de radio (e.g send,cancel) */
	uses interface Receive; /* Interfaz para de recpeciÃ³n via radio de
informaciÃ³n */
	uses interface SplitControl as AMControl;     /* Interfaz general de
control para inicio del mÃ³dulo de radio */

	/* Interfaces para el manejo del puerto USB */
	uses interface SplitControl as UsbControl;
	uses interface AMSend as UsbSend[am_id_t id];
	uses interface Packet as UsbPacket; /* Gestión en memoria del Paquetes
USB */
```

```
	uses interface AMPacket as UartAMPacket;


	/* Permite la caputar el RSSI del paquete como indicador de potencia del
paquete */
	uses interface PacketField<uint8_t> as PacketRSSI;
	uses interface PacketField<uint8_t> as PacketLinkQuality;
	uses interface PacketField<uint8_t> as PacketTransmitPower;
	uses interface LowPowerListening;

	/* Permite la caputa del valor del Tiempo Local del Nodo */
	uses interface LocalTime<TMilli> as LocalTime;

	/* Interfaces para el manejo de la informaciÃ³n de Gestion */
	uses interface AMSend as AMManagmentSend;	/* Interfaz de Envio */
	uses interface Receive as ManagmentReceive;/* Interfaz de RecepciÃ³n */

	/* MERIS: Monitoreo en Emergencias */
	uses interface Timer<TMilli> as TimerMonitor; /* Dispara los eventos
para el envio de Datos ciclicos */
	uses interface Random;                                          /*
Genera valores simulados de las seÃ±ales mÃ©dicas */
}

implementation {
	bool isRadioBusy = FALSE; /* Indica cuando esta ocupado el modulo del
radio de Informacion */
	bool isMRadioBusy = FALSE;/* Indica cuando esta ocupado el modulo del
radio de Gestion */

	message_t pkt;	/* Estructura del paquete de radio para gestion */
	message_t infoPkt;/* Estructura del paquete de radio para informacion */

	/* UART: ImplementaciÃ³n de un buffer para el envio de paquetes*/
	message_t usbPackets[UART_BUFFER_LEN]; /* Paquete para el manejo USB
	*/
	uint8_t uartIn = 0; /* Puntero a la primera posicion del registro del
buffer UART */
	uint8_t uartOut = 0;/* Puntero a la Ãºltima posicion del registro del
buffer UART */
	GatewayMsg gPayload; /* registro temporal de la informaciÃ³n a
transmitir */
	bool isUsbBusy;/* Indica si esta en proceso el envio de un paquete */

	bool isUartBufferFull();  /* Indica si el buffer esta lleno */
	bool isUartBufferEmpty(); /* Inica si el buffer esta vacio */
	void removeUartBuffer();  /* libera la Ãºltima posiciÃ³n del buffer */
	void addUartBuffer();	/* reserva una nueva posiciÃ³n del buffer */

	/*-- TABLA DE NODOS VECINOS --*/
	Neighbor Neighborhood[NEIGHBORHOOD_LEN];/* Tabla de Nodos Vecinos */
	bool Exclude[NEIGHBORHOOD_LEN];/* Tabla de Nodos Vecinos */

	Neighbor* getNeighbor(uint16_t source);/* Retorna el registro para el
vecino */
	uint8_t EvictorNeighborhood(); /* Libera el registro de los vecinos
inactivos */
	void SendFeedbackNeighbor(Neighbor* neighbor);/* Envia la informaciÃ³n
del Vecino por UART */
	uint8_t neighborPointer = 0; /* Puntero a una posiciÃ³n libre */

	uint8_t timePointer = 0; /* Puntero a la posiciÃ³n del vecino para
sincronizaciÃ³n en transmisiÃ³n */

	/*-- NUMEROS MAGICOS PARA DISIMINACIÃ“N --*/
	uint16_t magics[MAGICS_LEN]; /* Ultimos 3 Numeros magicos utilizados */
```

```
	uint8_t magicPointer = 0; /* Puntero circular a las posiciones del
registro de los nÃºmeros mÃ¡gicos */
	bool isMagicFree(uint16_t magic); /* Valid si el numero magico ya fue
utilizado */
	void updateMagics(uint16_t magic); /* Registra el uso de un numero
mÃ¡gico */

	/*-- MERIS --*/
	task void GenMonitorPacket();	/* Genera Paquetes de monitoreo, su
prioridad depende del estado de salud */
	task void SendMonitorPacket();	/* Envia los Paquetes de monitoreo que
esten en buffer */
	void SendFeedbackInfo(uint16_t bridge, MerisMonitorPacket* info); /*
Reenvia un paquete de informaciÃ³n a travÃ©s del UART */

	/*-- MERIS::BUFFER DE RADIO --*/
	MerisMonitorPacket rtBuffer[RT_BUFFER_LEN];	/* Buffer para el manejo
de paquetes de radio con prioridad */
	MerisMonitorPacket * getRTBuffer();
	uint8_t EvictorRTBuffer();	/* retorna la posiciÃ³n del paquete de menor
prioridad */
	uint32_t getMaxElapse(uint8_t priority);/* Retorna el plazo mÃ¡ximo de
paquete segun su prioridad */
	void EvalueElapse();/* Libera las posiciones que no cumplan con el plazo
*/
	uint8_t FirstPosition(); /* Primera posiciÃ³n no libre del buffer */
	bool isTimeToTransmision = FALSE; /* Indica si esta en fase de
transmisiÃ³n */
	uint16_t GetDestination(uint16_t addr,uint16_t source); /* Utiliza la
tabla de vecinos para identificar la ruta adecuada */

	void SendFeedbackSynchronize(ManagmentMsg* btrpkt);
	void SynchronizeTransmition();	/* Inicia el ciclo de transmisiÃ³n de
forma sincrona */
	Configuration myConfiguration;	/* almacena la informaciÃ³n de
configuraciÃ³n para la sincronizaciÃ³n de la transmisiÃ³n */
	uint16_t cCicles = 0;	/* Contador de Paquetes enviados */
	void UpdateSlotsDistribution(); /* Actualiza la redistribuciÃ³n de slots
segun el numero de paquetes recibidos */
	uint8_t GetNextConf();
	uint8_t GetNextNeighbor(); /* retorna el siguiente vecino de la tabla
para ser enviado por UART (DepuraciÃ³n) */
	uint8_t nextPointer=0; /* Puntero al proximo registro de la tabla
vecinos */

	/*-- DefiniciÃ³n de Variable Globales --*/
	uint8_t rCounter=0;		/* contador de paquetes recibidos */
	uint8_t sCounter=0;		/* Contador de Paquetes emitidos */

	uint8_t gCounter=0;		/* Contador de Paquetes Generados */

	/*-----------------------------------------------------------
		Evento de inicio de aplicaciÃ³n
	-----------------------------------------------------------*/
	event void Boot.booted() {
		uint8_t i = 0;

		rCounter=0;		/* contador de paquetes recibidos */
		sCounter=0;		/* Contador de Paquetes emitidos */
		gCounter=0;  /* contador de paquetes generados */

		/* Inicializa las variables necesarios para el buffer */
		for(; i<RT_BUFFER_LEN; i++){
			rtBuffer[i].destination = 0xFFFF;/* Hace disponible las
posiciones del buffer */
		}
```

```
		for(i=0;i<NEIGHBORHOOD_LEN;i++){
			Neighborhood[i].addr=0xFFFF; /* Todas las posiciones del
vecindario libres */
			Neighborhood[i].evictor=0;
			Neighborhood[i].rssi=0;
			Neighborhood[i].start=0;
			Neighborhood[i].stop=0;
			Neighborhood[i].cpkg=0;
		}

		myConfiguration.start = 98;/* espera el beacon y transmite al
final de la supertrama durante 10% */
		myConfiguration.stop = 100;/* El nodo inicia su transmisiÃ³n
utilizando el 10% final de la supertrama */

		call  LowPowerListening.setLocalSleepInterval (100);
		call AMControl.start();
		call UsbControl.start();
	}

	/*-----------------------------------------------------------
		Envia un paquete de establecimiento a la Red
	-----------------------------------------------------------*/
	task void SendManagmentPacket(){
		uint32_t magic = call LocalTime.get();

		if(cCicles<CICLES_CONF_UPDATE){
			cCicles +=1; /* Incrementar los ciclos */
		}else{
			UpdateSlotsDistribution();/* Actualiza la distribuciÃ³n de
slots en funciÃ³n de la cantidad de paquetes enviados,*/
			cCicles = 0;
		}

		if (!isMRadioBusy) {
			ManagmentMsg* btrpkt = (ManagmentMsg*)(call
RadioPacket.getPayload(&pkt,  sizeof(ManagmentMsg)));

			uint8_t pConf = GetNextConf();   /* retorna la posiciÃ³n de
nodo que debe ser actualizada */

			btrpkt->magic = magic;
			btrpkt->hops = 0;
			btrpkt->timestamp = call LocalTime.get();

	/* Envia la informaciÃ³n de configuraciÃ³n para un nodo de la red */
			btrpkt->addr = Neighborhood[pConf].addr;
			btrpkt->start = Neighborhood[pConf].start;
			btrpkt->stop = Neighborhood[pConf].stop;
			btrpkt->rssi = Neighborhood[pConf].rssi;
			SendFeedbackNeighbor(&Neighborhood[GetNextNeighbor()]);


			if (call AMManagmentSend.send(AM_BROADCAST_ADDR, &pkt,
sizeof(ManagmentMsg)) == SUCCESS) {
				isMRadioBusy = TRUE;
				updateMagics(magic);/* Actualiza el registro de
numeros magicos */
			}
		}
	}

	/*-----------------------------------------------------------
		Disparo del TIMER
	-----------------------------------------------------------*/
```

```
	event void TimerMonitor.fired() {
		post GenMonitorPacket();  /* Genera paquetes de informaciÃ³n Red
*/
	}

	/*-----------------------------------------------------------
		Disparo del TIMER
	-----------------------------------------------------------*/
	event void Timer0.fired() {
		call Leds.led2Toggle();    /* Conmuta el LED AMARILLO */
		post SendManagmentPacket();/* Envia paquetes de establecimiento y
mantenimiento de Red */
	}

	/*-----------------------------------------------------------------
		Arranque el modulo de Radio
	-----------------------------------------------------------------*/
	event void AMControl.startDone(error_t err) {
		if (err == SUCCESS) {
			if(TOS_NODE_ID==GATEWAY_ADDR){/* DefiniciÃ³n estÃ¡tica del
nodo Gateway */
				call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
	/* Eventos para el envio de paquetes de gestiÃ³n de la red  */
			}else{
				call
TimerMonitor.startPeriodic(TIMER_MONITOR_MILLI); /* Eventos para la
generaciÃ³n de paquetes de informaciÃ³n */
				/* call TimerTx.startPeriodic(TIMER_TX_MILLI); */
	/* Eventos para el envio de paquetes de informaciÃ³n */
			}
		}
		else {
			call AMControl.start();
		}
	}

	/*-----------------------------------------------------------
		Evento luego que el modulo de radio a enviado un paquete
	-----------------------------------------------------------*/
	event void AMSend.sendDone(message_t* msg, error_t error) {
		isRadioBusy = FALSE;
		post SendMonitorPacket(); /* Activa la tarea e envio */
	}

	/*-----------------------------------------------------------
		Evento luego que el modulo de radio a enviado un paquete
	-----------------------------------------------------------*/
	event void AMManagmentSend.sendDone(message_t* msg, error_t error) {
		isMRadioBusy = FALSE;
	}

	/*-----------------------------------------------------------
		Evento para cuando el modulo de radio se ha detenido
	-----------------------------------------------------------*/
	event void AMControl.stopDone(error_t err) {}

	bool isUartBufferFull(){
		return (uartIn>uartOut && (UART_BUFFER_LEN<=(uartIn-uartOut+1)))
|| (uartOut>uartIn && ((uartOut-uartIn)==1));
	}

	bool isUartBufferEmpty(){
		return uartIn==uartOut;
	}

	void removeUartBuffer(){
```

```
		uartOut++;
		if(uartOut>=UART_BUFFER_LEN){
			uartOut=0;
		}
	}

	void addUartBuffer(){
		uartIn++;
		if(uartIn>=UART_BUFFER_LEN){
			uartIn=0;
		}
	}

	/*-----------------------------------------------------------
		Evalua el numero mÃ¡gico del paquete para disiminarlo o no
	-----------------------------------------------------------*/
	bool isMagicFree(uint16_t magic){
		uint8_t i=0;
		for(;i<MAGICS_LEN;i++){
			if(magics[i]==magic){
				return FALSE;
			}
		}
		return TRUE;
	}

	/*-----------------------------------------------------------
		Actualiza el registro de Numeros MÃ¡gicos de forma circular
	-----------------------------------------------------------*/
	void updateMagics(uint16_t magic){
		magics[magicPointer]=magic;/* Actualiza la posiciÃ³n a la que
apunta el puntero */
		magicPointer++;
		if(magicPointer >= MAGICS_LEN){ /* una vez alcanza el mÃ¡ximo
valor, reinicia su valor a cero */
			magicPointer = 0;
		}
	}
	/**-----------------------------------------------------------
		Este evento se dispara cuando se recibe un nuevo paquete a
travÃ©s del modulo de Radio
	-----------------------------------------------------------*/
	event message_t* Receive.receive(message_t* msg, void* payload, uint8_t
len) {
		uint16_t source = call AMPacket.source(msg);
		/*uint16_t destination = call AMPacket.destination(msg);*/
		Neighbor * neighbor = getNeighbor(source);    /* Busca en el
registro en el vecindario */

		/*if(call PacketRSSI.get(msg)<MIN_RSSI){return msg;}*/


		if(len == sizeof(MerisMonitorPacket)){ /* Paquete de InformaciÃ³n
de Monitoreo */
			MerisMonitorPacket* meris = (MerisMonitorPacket*)payload;

			neighbor->cpkg += 1; /* Incrementa el contador de mensajes
de la fuente */
			meris->timestamp  +=neighbor->offset ;  /* Actualizo la
estampa de tiempo utilizando el offset */

			meris->SaO = neighbor->cpkg;

			SendFeedbackInfo(source,meris); /* Lo Consumo, comparmos el
origen real, con el marcado en la info para ver si hubo puente */
```

```
                        call Leds.led1Toggle();

                        if(meris->destination!=TOS_NODE_ID){ /* Si no es el nodo
destino, lo renvio segÃ°n el enrutamiento */
                                MerisMonitorPacket * mmp = getRTBuffer(); /* Busca
un espacio en buffer para el mensaje */

                                mmp->SaO = meris->SaO;
                                mmp->FrC = meris->FrC;
                                mmp->FrR = meris->FrR;
                                mmp->Triage = meris->Triage;
                                mmp->priority = meris->priority;
                                mmp->timestamp = meris->timestamp; /* Hace cuanto se
genero el mensajes */
                                mmp->destination = meris->destination;
                                mmp->source = meris->source;
                                post SendMonitorPacket(); /* Activa la tarea e envio
*/
                        }
                }

                return msg;
        }

        /*-----------------------------------------------------
                Evento inicio del control USB
        -----------------------------------------------------*/
        event void UsbControl.startDone(error_t err) {
                /* En caso de fallo, reintenta hasta lograr el correct arranque
*/
                if(err !=SUCCESS){
                        call UsbControl.start();
                }
        }

        /*-----------------------------------------------------
                Evento Paquete enviad por USB
        -----------------------------------------------------*/
        event void UsbSend.sendDone[am_id_t id](message_t* bufPtr, error_t
error) {
                if(error==SUCCESS){
                        atomic{
                                removeUartBuffer(); /* Libera la posiciÃ³n del
buffer */
                                isUsbBusy = FALSE;
                        }

                        if(!isUartBufferEmpty()){
                                message_t* msg = &usbPackets[uartOut];
                                uint8_t len =  call UsbPacket.payloadLength(msg);
                                am_id_t i = call UartAMPacket.type(msg);

                                if(call UsbSend.send[i](AM_BROADCAST_ADDR, msg, len)
== SUCCESS){
                                        isUsbBusy = TRUE;
                                }
                        }
                }
        }

        /*-----------------------------------------------------
                USB Stoped
        -----------------------------------------------------*/
        event void UsbControl.stopDone(error_t err) {}

        /*-- NEIGHBORHOOD --*/
```

```
/*-------------------------------------------------------------------------
	Retorna el registro donde se almacena la informaciÃ³n de un nodo vecino
-------------------------------------------------------------------------*/
	Neighbor* getNeighbor(uint16_t source){
		uint8_t i = 0;
		for(;i<NEIGHBORHOOD_LEN;i++){/* Buscamos un registro anterior
para la direcciÃ³n dada */
			if(Neighborhood[i].addr==source){
				return &Neighborhood[i];
			}
		}

		for(i=0;i<NEIGHBORHOOD_LEN;i++){ /* Buscamos un registro vacio
(direcciÃ³n 0xFFFF) */
			if(Neighborhood[i].addr==0xFFFF){
				return &Neighborhood[i];
			}
		}

		return &Neighborhood[EvictorNeighborhood()]; /* De no encontrar,
sobreescribe al vecino menos activo <<Patron Evictor>> */
	}/* getNeighbor */

	/* Liber la posiciÃ³n del nodo menos activo << PatrÃ³n Evictor -
criterio de recurrencia >> */
	uint8_t EvictorNeighborhood(){
		uint8_t i = 0;
		uint8_t nodeDrop = 0;/* Posicion del vecino a liberar */

		for(;i<NEIGHBORHOOD_LEN;i++){
			if(Neighborhood[i].evictor<Neighborhood[nodeDrop].evictor){
/* Buscamos el minimo evictor */
				nodeDrop = i;
				continue;
			}else
if(Neighborhood[i].evictor==Neighborhood[nodeDrop].evictor){/* Recurrimos a un
segundo criterio... salto  */

	if(Neighborhood[nodeDrop].hops<Neighborhood[i].hops){/* el de mas saltos
a la gateway es sustituido */
					nodeDrop = i;
					continue;
				}else
if(Neighborhood[nodeDrop].hops==Neighborhood[i].hops){	/* Tercer
criterio... calidad del enlace */

	if(Neighborhood[nodeDrop].lqi>Neighborhood[i].lqi){  /* la menor calidad
es sustituido */
						nodeDrop = i;
						continue;
					}
				}
			}
		}

		Neighborhood[nodeDrop].cpkg = 0; /* limpia el contador de
mensajes */
		Neighborhood[nodeDrop].evictor = 0;/* limpia el contador de
actividad */
		Neighborhood[nodeDrop].addr = 0xFFFF;   /* Se libera el registro
del nodo perdedor */
		Neighborhood[nodeDrop].offset = 0;
		return nodeDrop;
	}/* EvictorNeighborhood */

	/*-- END NEIGHBORHOOD--*/
```

```
/*-- MERIS --*/

/*-----------------------------------------------------------------
      Genera un paquete de informaciÃ³n
-----------------------------------------------------------------*/
task void GenMonitorPacket(){
      MerisMonitorPacket * mmp = getRTBuffer();
      uint8_t SaO;
      uint8_t FrC;
      uint8_t FrR;
      uint8_t Triage=1;

      gCounter +=1;

      SaO = 89+call Random.rand16() % 15;
      FrC = 40+call Random.rand16() % 80;
      FrR = 5+call Random.rand16() % 35;

      /*Calculo del Triage */
      if(FrR<=13){
            SaO = 90;
            FrC = 50;
            Triage = 3;
      }else if (FrR>13 && FrR<=20 ){ /* se evalua el pulso */
            FrC = 70;
            if(SaO<95){
                  Triage = 2;
            }else{
                  Triage = 1;
            }
      }else if (FrR>20 && FrR<=30 ){
            if(FrC<60){
                  SaO=93;
                  Triage = 3;
            }else if(FrC>=60 && FrC<=100) {
                  SaO = 95;
                  Triage = 2;
            }else{
                  SaO=94;
                  Triage = 3;
            }
      }else if(FrR>30){
            SaO = 89;
            FrC = 100;
            Triage = 3;
      }else{
            Triage = 1;
      }

      /* Forzar un paquete rojo */
      /*if(TOS_NODE_ID==6){
            FrR=35;
            SaO = 89;
            FrC = 100;
            Triage = 3;
      }*/

    /* Forzar Amarillo */
    /*if(TOS_NODE_ID==5){
      FrR=25;
            SaO = 95;
            FrC = 70;
            Triage = 2;
      }*/
```

```
            /* Forzar Verde */
            /*
          if(TOS_NODE_ID==4){
            FrR=15;
                  SaO = 100;
                  FrC = 70;
                  Triage = 1;
            }
            */
            mmp->SaO = SaO;
            mmp->FrC = FrC;
            /*mmp->FrR = FrR;*/
            mmp->FrR = gCounter;
            mmp->Triage = Triage;
            mmp->priority = Triage;                  /* Determina un criterio
de prioridad */
            mmp->timestamp = call LocalTime.get(); /* Marca el paquete con el
tiempo actual local */
            mmp->destination = GATEWAY_ADDR;       /* El Nodo de destino es
la Gateway */
            mmp->source = TOS_NODE_ID;             /* DirecciÃ³n del Nodo
Origen  */
            post SendMonitorPacket(); /* Dispara la tarea e envio */

      }/* GenMonitorPacket */

      /*-----------------------------------------------------------------
            Genera un paquete de informaciÃ³n
      -----------------------------------------------------------------*/
      task void SendMonitorPacket(){
            MerisMonitorPacket* payload;
            uint8_t i = 0;
            uint8_t top;

            atomic if(!isTimeToTransmision || isRadioBusy){ /* Si no es
tiempo de transmisiÃ³n o el radio esta ocupado */
                  return; /* descarta la tarea */
            }

            EvalueElapse();      /* Descarta paquetes fuera de plazo */
            top = FirstPosition(); /* indicador del paquete con mayor
prioridad */

            if(top!=0xFF){
                  uint16_t destination;     /* Destino del paquete */

                  for(;i<RT_BUFFER_LEN;i++){ /* contendrÃ¡ la posiciÃ³n en el
buffer del paquete a enviar */
                        if(rtBuffer[i].destination!=0xFFFF){

      if(rtBuffer[i].priority>rtBuffer[top].priority){ /* Primer criterio: la
prioridad */
                                    top=i;
                              }else
if(rtBuffer[i].priority==rtBuffer[top].priority){ /* Segundo criterio, el
plazo */

      if(rtBuffer[i].timestamp>rtBuffer[top].timestamp){
                                          top=i;

                                    }//if
                              }//elseif
                        }//if
                  }//for

                  /* Proceder a enviarlo */
```

```
                    payload = (MerisMonitorPacket*)(call
RadioPacket.getPayload(&infoPkt, sizeof(MerisMonitorPacket)));
                    payload->SaO = rtBuffer[top].SaO;
                    payload->FrC = rtBuffer[top].FrC;
                    payload->FrR = rtBuffer[top].FrR;
                    payload->Triage = rtBuffer[top].Triage;
                    payload->priority = rtBuffer[top].priority;

                    /*payload->timestamp = rtBuffer[top].timestamp;*/
                    payload->timestamp = call LocalTime.get(); /* tiempo en el
que se envia el paquete */

                    payload->source = rtBuffer[top].source;
                    payload->destination = rtBuffer[top].destination;

                    destination = GetDestination(payload->destination,payload-
>source); /* Utiliza la tabla de vecinos para seleccionar la ruta adecuada */

                    /* if(TOS_NODE_ID==6 ||TOS_NODE_ID==5){destination = 4; }
*/
                    if(call AMSend.send(destination, &infoPkt,
sizeof(MerisMonitorPacket)) == SUCCESS){
                          atomic{
                                isRadioBusy = TRUE;
                                rtBuffer[top].destination=0xFFFF; /* Libera
la posiciÃ³n del buffer */
                          }
                    }
             }
      }/* SendMonitorPacket */

      /*-------------------------------------------------------------------------
---
            Retorna la direcciÃ³n de destino mediÃ¡tica para los mensajes
      -------------------------------------------------------------------------
---*/
      uint16_t GetDestination(uint16_t endAddr, uint16_t source){
            uint8_t i = 0;
            uint8_t bridge = 0xFF;                              /* En caso de
encontrarse entre los vecinos, utiliza un nodo cercano a la gateway  */

            for(i=0;i<NEIGHBORHOOD_LEN;i++){              /* Buscamos si
aparece en el vecindario */
                    if(Neighborhood[i].addr!=0xFFFF) {

                          if(Neighborhood[i].rssi<MIN_RSSI ||
Neighborhood[i].rssi==source){ /* Rechazar paquetes de nodos muy distantes */
                                continue;
                          }

                          if(Neighborhood[i].addr==endAddr){

                                return Neighborhood[i].addr;

                          }

                          if(bridge==0xFF){ bridge=i;      } /* primer
registro opcional */

                          if(Neighborhood[i].hops<Neighborhood[bridge].hops){
/*Primer criterio, el mas cercano a la gateway */
                                bridge = i;
                          }else
if(Neighborhood[i].hops==Neighborhood[bridge].hops){
```

```
	if(Neighborhood[i].evictor>Neighborhood[bridge].evictor){ /* Segundo
criterio, el mÃ¡s activo */
                                        bridge = i;
                                    }else
if(Neighborhood[i].evictor==Neighborhood[bridge].evictor){

	if(Neighborhood[i].lqi>Neighborhood[bridge].lqi){ /* Tercer criterio,
mejor lqi */
                                                bridge = i;

                                        }

                                    }
                                }
                        }
                }
                if(bridge==0xFF)bridge=0;
                return Neighborhood[bridge].addr;
        }/* GetDestination */

        /* Retorna la posiciÃ³n de primer paquete que encuentre */
        uint8_t FirstPosition(){
                uint8_t i = 0;
                for(;i<RT_BUFFER_LEN;i++){ /* contendrÃ¡ la posiciÃ³n en el
buffer del paquete */
                        if(rtBuffer[i].destination!=0xFFFF){
                                return i;
                        }
                }
                return 0xFF;
        }/* FirstPosition */

        /*-----------------------------------------------------------------
                Retorna una posiciÃ³n en el buffer
        -----------------------------------------------------------------*/
        MerisMonitorPacket * getRTBuffer(){
                uint8_t i = 0;
                EvalueElapse();     /* Aplica el criterio de descarte por plazos
*/

                for(;i<RT_BUFFER_LEN;i++){
                        if(rtBuffer[i].destination==0xFFFF){    /* Buscamos una
posiciÃ³n libre en el buffer */
                                return &rtBuffer[i];
                        }
                }
                return &rtBuffer[EvictorRTBuffer()]; /* sino, sacrifica el
registro del nodo 0 */
        }/* getRTBuffer */

        /*-----------------------------------------------------------------
                Libera la posiciÃ³n en el buffer que tenga menor prioridad
        -----------------------------------------------------------------*/
        uint8_t EvictorRTBuffer(){
                uint8_t i = 0;
                uint8_t minPriority = rtBuffer[0].priority;   /* valor evictor
mÃ-nimo, por defecto el del i = 0 */
                uint8_t drop = 0;          /* Posicion del vecino a liberar */

                for(;i<RT_BUFFER_LEN;i++){
                        if(rtBuffer[i].destination!=0xFFFF){ /* Descartar
posiciones vacias */
                                uint32_t elapse = call LocalTime.get() -
rtBuffer[0].timestamp;
                                if(elapse>getMaxElapse(rtBuffer[i].priority)){
```

```
                                    rtBuffer[i].destination=0xFFFF;
                                    drop=i;
                                    return drop;
                              }else{
                                    if(rtBuffer[i].priority<minPriority){   /*
Utilizamos el de mÃ-nima prioridad como posiciÃ³n libre en el buffer */
                                          minPriority=rtBuffer[i].priority;
                                          drop=i;
                                    }
                              }
                        }
                  }
                  return drop;
            }/* EvictorRTBuffer */

            /*-----------------------------------------------------------------
                  Descarta los paquetes de que no cumplen el plazo
            -----------------------------------------------------------------*/
            void EvalueElapse(){
                  uint32_t time = call LocalTime.get();
                  uint32_t hiTime = time -HI_PRIORITY_ELAPSE;
                  uint32_t loTime = time -LOW_PRIORITY_ELAPSE;
                  uint32_t miTime = time -MIDDLE_PRIORITY_ELAPSE;
                  uint8_t i = 0;
                  for(;i<RT_BUFFER_LEN;i++){
                        if(rtBuffer[i].destination!=0xFFFF){ /* Descartar
posiciones vacias */
                              if( (rtBuffer[i].priority==HI_PRIORITY && hiTime >
rtBuffer[i].timestamp) ||
                                    (rtBuffer[i].priority==MIDDLE_PRIORITY
&& miTime > rtBuffer[i].timestamp)  ||

            (rtBuffer[i].priority==LOW_PRIORITY && loTime > rtBuffer[i].timestamp)){
                                    rtBuffer[i].destination=0xFFFF;  /* Libera la
posiciÃ³n */
                              }
                        }
                  }
            }/* EvalueElapse */

            /*-----------------------------------------------------------------
                  Retorna el plazo segun la prioridad
            -----------------------------------------------------------------*/
            uint32_t getMaxElapse(uint8_t priority){
                  if(priority==HI_PRIORITY)return HI_PRIORITY_ELAPSE;
                  if(priority==LOW_PRIORITY)return LOW_PRIORITY_ELAPSE;
                  return MIDDLE_PRIORITY_ELAPSE;
            }/* getMaxElapse */

            /*-----------------------------------------------------------
                  Envia la informaciÃ³n Meris a travÃ©s de la interfaz UART
            -----------------------------------------------------------*/
            void SendFeedbackInfo(uint16_t bridge, MerisMonitorPacket* info){
                  message_t* usbPacket;
                  MerisMonitorPacket* payload;

                  if(isUartBufferFull()){ /* Si el buffer esta lleno, solo se puede
descartar el mensaje */
                        return;
                  }

                  atomic {
                        usbPacket = &usbPackets[uartIn]; /* Recupera el registro */
                        addUartBuffer();    /* Reserva sobre el buffer esa
posiciÃ³n */
                  }
```

```
            payload = (MerisMonitorPacket*)(call
UsbPacket.getPayload(usbPacket,  sizeof(MerisMonitorPacket)));

            payload->SaO = info->SaO;
            payload->FrC = info->FrC;
            payload->FrR = info->FrR;
            payload->Triage = info->Triage;
            payload->priority=info->priority;
            payload->timestamp=call LocalTime.get() - info->timestamp;

            payload->source=info->source;

            if(bridge!=info->source){
                  payload->destination=bridge;
            }else{
                  payload->destination=info->destination;
            }

            call UartAMPacket.setType(usbPacket,AM_MERISMONITORPACKET);


            if(!isUsbBusy){
                  if(call
UsbSend.send[AM_MERISMONITORPACKET](AM_BROADCAST_ADDR, usbPacket,
sizeof(MerisMonitorPacket)) == SUCCESS){
                        isUsbBusy = TRUE;
                  }
            }
      } /* SendFeedbackInfo */

      /*-- END MERIS --*/

      /* Envia la informaciÃ³n de un nodo vecino por UART */
      void SendFeedbackSynchronize(ManagmentMsg* btrpkt){
            message_t* usbPacket;
            ManagmentMsg* payload;

            if(isUartBufferFull()){ /* Si el buffer esta lleno, solo se puede
descartar el mensaje */
                  return;
            }

            atomic {
                  usbPacket = &usbPackets[uartIn]; /* Recupera el registro */
                  addUartBuffer();    /* Reserva sobre el buffer esa
posiciÃ³n */
            }

            payload = (ManagmentMsg*)(call UsbPacket.getPayload(usbPacket,
sizeof(ManagmentMsg)));

            payload->magic=btrpkt->magic;
            payload->hops=btrpkt->hops;
            payload->timestamp=btrpkt->timestamp;

            payload->addr=btrpkt->addr;
            payload->start=btrpkt->start;
            payload->stop=btrpkt->stop;
            call UartAMPacket.setType(usbPacket,AM_MANAGMENTMSG);
            if(!isUsbBusy){
                  if(call UsbSend.send[AM_MANAGMENTMSG](AM_BROADCAST_ADDR,
usbPacket, sizeof(ManagmentMsg)) == SUCCESS){
                        isUsbBusy = TRUE;
                  }
            }
      } /*  SendFeedbackSynchronize */
```

```
	/* Envia la informaciÃ³n de un nodo vecino por UART */
	void SendFeedbackNeighbor(Neighbor* neighbor){
		message_t* usbPacket;
		Neighbor* payload;

		if(isUartBufferFull()){ /* Si el buffer esta lleno, solo se puede
descartar el mensaje */
			return;
		}

		atomic{
			usbPacket = &usbPackets[uartIn]; /* Recupera el registro */
			addUartBuffer();    /* Reserva sobre el buffer esa
posiciÃ³n */
		}

		payload = (Neighbor*)(call UsbPacket.getPayload(usbPacket,
sizeof(Neighbor)));

		payload->addr = neighbor->addr;
		payload->hops = neighbor->hops;
		payload->rssi = neighbor->rssi;
		payload->lqi = neighbor->lqi;

		/*payload->offset = neighbor->offset;*/
		payload->offset = call LocalTime.get(); /* Utilizamos el offset
de los paquetes vecinos para sincronizar los relojes de Gateway y PC*/

		payload->evictor = neighbor->evictor;
		payload->cpkg = neighbor->cpkg; /* Contador de mensajes de
informaciÃ³n recibidos */
		payload->start = neighbor->start;
		payload->stop = neighbor->stop; /* informaciÃ³n en porcentaje */
		call UartAMPacket.setType(usbPacket,AM_NEIGHBOR);
		if(!isUsbBusy){
			if(call UsbSend.send[AM_NEIGHBOR](AM_BROADCAST_ADDR,
usbPacket, sizeof(Neighbor)) == SUCCESS){
				isUsbBusy = TRUE;
			}
		}
	} /*  SendFeedbackNeighbor */

	/**-------------------------------------------------------------
		Este evento se dispara cuando se recibe un nuevo paquete a
travÃ©s del modulo de Radio
	-------------------------------------------------------------*/
	event message_t* ManagmentReceive.receive(message_t* msg, void* payload,
uint8_t len) {

		if(call PacketRSSI.get(msg)<MIN_RSSI){ /* Rechazar paquetes de
sincronizaciÃ³n de nodos muy distantes */
			return msg;
		}

		if (len == sizeof(ManagmentMsg)) {
			Neighbor * neighbor;

			uint16_t source = call AMPacket.source(msg);


			ManagmentMsg* btrpkt = (ManagmentMsg*)payload;
			uint16_t magic = btrpkt->magic;

			uint8_t hops =  btrpkt->hops;
```

```
                    /* Rutas de PropagaciÃ³n */
                    neighbor = getNeighbor(source);  /* reserva un nuevo
registro o actualiza uno anterior */
                    neighbor->addr = source;
                    neighbor->hops = hops;
                    neighbor->lqi = call PacketLinkQuality.get(msg);

                    neighbor->offset = call LocalTime.get() - btrpkt-
>timestamp; /* EstimaciÃ³n del offset entre relojes */
                    neighbor->evictor += 1;    /* Incrementa el contador de
actividad del nodo */

                    if(btrpkt->addr == TOS_NODE_ID){ /* Sincronizo mi
comunicaciÃ³n si el mensaje es para mi */

                          myConfiguration.start = btrpkt->start; /* porcentaje
en que inicia la transmisiÃ³n */
                          myConfiguration.stop = btrpkt->stop; /* porcentaje
de la trama en que finaliza la transmisiÃ³n */
                          neighbor->rssi = (call PacketRSSI.get(msg)+btrpkt-
>rssi)/2; /* promedia el valor actual con el recibido desde el padre */

                    }else{
                          neighbor->rssi = (call PacketRSSI.get(msg)+neighbor-
>rssi)/2; /* promedia el valor actual con el recibido */

                    }

                    if(isMagicFree(magic)){

                          uint8_t pConf = 0;

                          /* Disimina hacia otros nodos */
                          updateMagics(magic);      /* Actualiza el registro
de numeros magicos */
                          btrpkt->hops ++;          /* Incrementa el numero de
saltos */
                          btrpkt->timestamp = call LocalTime.get(); /*
actualizamos la estampa de tiempo para el paquete */

                          if(cCicles<CICLES_CONF_UPDATE){ /* Luego de N ciclos
*/
                                cCicles +=1; /* Incrementar los ciclos */

                          }else{
                                UpdateSlotsDistribution();/* Actualiza la
distribuciÃ³n de slots en funciÃ³n de la cantidad de paquetes recibidos */
                                cCicles = 0;
                          }

                          pConf = GetNextConf();
                          neighbor = &Neighborhood[pConf];

                          btrpkt->addr =  neighbor->addr;
                          btrpkt->start = neighbor->start;
                          btrpkt->stop = neighbor->stop;

                          btrpkt->rssi = neighbor->rssi;

                          /*Envia por USB la informaciÃ³n del vecino */

      /*SendFeedbackNeighbor(&Neighborhood[GetNextNeighbor()]);*/

      SendFeedbackNeighbor(&Neighborhood[GetNextNeighbor()]);

                          /*SendFeedbackSynchronize(btrpkt);*/
```

```
                        /* Paquete de SincronizaciÃ³n */

                        if (call AMManagmentSend.send(AM_BROADCAST_ADDR,
msg, sizeof(ManagmentMsg)) == SUCCESS) {      /* Propago el mensaje */
                              isMRadioBusy = TRUE;

                        }

                        call Leds.led2On();
                        SynchronizeTransmition(); /* BEACON DE
SINCRONIZACION, INICIO PROCEDIMIENTOS DE ESPERA Y TRANSMISIÃ"N */
                  }
            }
            return msg;
      }

      void SynchronizeTransmition(){
            call TimerTxStart.stop(); /* por si estaba activo */
            call TimerTxStop.stop(); /* por si estaba activo */

            if(myConfiguration.start>98)myConfiguration.start=98;
            if(myConfiguration.stop>100)myConfiguration.stop=100;

            if(myConfiguration.start==myConfiguration.stop){ /* margen de
seguridad de envio de datos */
                  myConfiguration.start=98;
                  myConfiguration.stop=100;
            }

            call TimerTxStart.startOneShot(PERCENTAGE_MILLI *
myConfiguration.start); /* convierte el porcentage en tiempo de espera antes
de cancelar la transmisiÃ³n */
            call TimerTxStop.startOneShot(PERCENTAGE_MILLI *
myConfiguration.stop); /* convierte el porcentage en tiempo de espera antes de
cancelar la transmisiÃ³n */
      }

      /*-----------------------------------------------------------
            TIMER de transmisiÃ³n START
      -----------------------------------------------------------*/
      event void TimerTxStart.fired() {
            isTimeToTransmision = TRUE;
            call Leds.led0On();
            post SendMonitorPacket(); /* Envia paquetes de informaciÃ³n Red
*/
            call Leds.led2Off();
      }

      /*-----------------------------------------------------------
            TIMER de transmisiÃ³n STOP
      -----------------------------------------------------------*/
      event void TimerTxStop.fired() {
            isTimeToTransmision = FALSE;
            call Leds.led0Off();
      }

      /*-----------------------------------------------------------
            Actualiza los periodos de actualizaciÃ³n de los nodos en funciÃ³n
de la cantidad de mensajes que se emiten,
            retorna la posiciÃ³n del nodo que debe ser actualizado con esta
informaciÃ³n, este proceso es cÃ-clico
      -----------------------------------------------------------*/
      void UpdateSlotsDistribution() {
            uint8_t i = 0;
            uint16_t total = 0;
            uint16_t start = 0;
```

```
		for(;i<NEIGHBORHOOD_LEN;i++){
			if(Neighborhood[i].addr!=0xFFFF){
				total += Neighborhood[i].cpkg;	/* Contador del
total de paquetes recibidos */
			}
		}

		atomic for(i=0;i<NEIGHBORHOOD_LEN;i++){
			if(Neighborhood[i].addr!=0xFFFF){
				if(total==0){
					Neighborhood[i].stop = Neighborhood[i].start
= 0x0;
				}else{
					Neighborhood[i].start = start;

					Neighborhood[i].stop = start +
Neighborhood[i].cpkg * 98 / total; /* Porcentaje de la carga recibida */
					start = Neighborhood[i].stop;
				}

				if(Neighborhood[i].cpkg==0){
					Neighborhood[i].start = 0;
					Neighborhood[i].stop = 0; /* indica que
posiblemente el nodo esta fuera de covertura */
				}

				Neighborhood[i].cpkg = 0; /* restablece el contador
de paquetes */

				if(Neighborhood[i].evictor == 0){
					Neighborhood[i].addr=0xFFFF; /* Borrar el
registro de nodos que no responden */
					Neighborhood[i].start = 0;
					Neighborhood[i].stop = 0; /* indica que
posiblemente el nodo esta fuera de covertura */
					Neighborhood[i].rssi = 0;

				}
				Neighborhood[i].evictor = 0;

			}
		}

	}/* UpdateSlotsDistribution */

	uint8_t GetNextConf(){
		uint8_t i;
		/* busca el siguiente nodo al cual debe ser realizada la
actualizaciÃ³n del periodo de sincronizacion */
		for(i=0;i<NEIGHBORHOOD_LEN;i++){
			if(Neighborhood[timePointer].addr!=0xFFFF &&
Neighborhood[timePointer].stop>0){
				uint8_t ret = timePointer;
				timePointer += 1;	/* dispone para que sea la
siguiente posiciÃ³n en la tabla */
				if(timePointer>=NEIGHBORHOOD_LEN){
					timePointer=0;
				}
				return ret;
			}else{
				timePointer += 1;	/* dispone para que sea la
siguiente posiciÃ³n en la tabla */
				if(timePointer>=NEIGHBORHOOD_LEN){
					timePointer=0;
				}
```

```
                }
            }
            return 0;
        }

        /* Retorna el siguiente vecino de la tabla */
        uint8_t GetNextNeighbor(){
            uint8_t i;
            /* busca el siguiente nodo */
            for(i=0;i<NEIGHBORHOOD_LEN;i++){
                if(Neighborhood[nextPointer].addr!=0xFFFF){
                    uint8_t ret = nextPointer;
                    nextPointer += 1;   /* dispone para que sea la
siguiente posiciÃ³n en la tabla */
                    if(nextPointer>=NEIGHBORHOOD_LEN){
                        nextPointer=0;
                    }
                    return ret;
                }else{
                    nextPointer += 1;   /* dispone para que sea la
siguiente posiciÃ³n en la tabla */
                    if(nextPointer>=NEIGHBORHOOD_LEN){
                        nextPointer=0;
                    }
                }
            }
            return 0;
        }
}
```

2.2 Aplicación de la estación base

La aplicación de despliegue que corre en la estación base está divida en cuatro paquetes:

- Control: el cual contiene las clases: controlAPP.java que recibe los paquetes desde la interfaz USB, actualiza las tablas de datos de los signos vitales y de los vecinos; y la clase NeiborhoodListener.java que es una interfaz que permite notificar los cambios sucedidos sobre la tabla de vecinos.

- Model: que contiene las clases que generan las estructuras de datos de información (MerisMonitorPacket.java) y de vecinos (Neighbor.java).

- View: paquete que contiene las clases: Network.java que es la interfaz donde se presenta todo el monitoreo de la red y StartView.java que es la interfaz principal donde se despliegan los datos de los signos vitales.

A continuación, se presenta el código de las clases de los paquetes control y view, las de modelo no se muestran dado que estas las genera automáticamente la herramienta MIG (Message Interface Generator) que permite convertir las estructuras definidas dentro del código NesC en clases java que son utilizadas para interpretar los paquetes que vienen a través del puerto USB.

2.2.1 Control

En esta sección se presenta las clases controlAPP.java y NeiborhoodListener.java

2.2.1.1 controlAPP.java

```
package control;
```

```
import java.awt.Component;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.BitSet;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;

import javax.swing.table.DefaultTableModel;

import model.ManagmentMsg;
import model.MerisMonitorPacket;
import model.extention.Neighbor;
import net.tinyos.message.Message;
import net.tinyos.message.MessageListener;
import net.tinyos.message.MoteIF;
import net.tinyos.packet.BuildSource;
import net.tinyos.packet.PacketListenerIF;
import net.tinyos.packet.PhoenixSource;
import net.tinyos.util.Dump;
import net.tinyos.util.PrintStreamMessenger;
import view.StartView;

public class ControlApp extends Thread implements MessageListener,
PacketListenerIF{

	private static final long WATCH_DOG_MILLI = 5000; /* verifica que si no
se ha recibido informaciÃ³n de nodo luego de ese tiempo se elimina*/
	private static final int MAX_WARNINGS = 2;

	String source = "serial@/dev/ttyUSB1:iris";
	public long offset = 0;
	public long time = 0;

	PrintStream stream;
	PhoenixSource phoenix;
	MoteIF mif;

	DefaultTableModel model;

	HashMap<String, Neighbor> neighborhood = new HashMap<String,
Neighbor>();

	public ControlApp() {
	}

	public boolean connect(PrintStream stream) {
		try {
			this.stream = stream;
			if (phoenix != null) {
				disconnect();
			}
			phoenix = BuildSource.makePhoenix(source, new
PrintStreamMessenger(stream));
			phoenix.registerPacketListener(this);

			mif = new MoteIF(phoenix);
			mif.registerListener(new MerisMonitorPacket(), this);
			mif.registerListener(new Neighbor(), this);

			this.stream.println("[" + new Date() + "] Conectado\n");
			stream.flush();

			start();	/* Watch Dog */

			return true;
```

```
			} catch (Exception ex) {
				ex.printStackTrace(stream);
				return false;
			}
		}

		public void run(){
			for(;;){
				try {
					boolean lessOne = false; /* almenos un nodo esta en
la mira */

					Iterator<Neighbor> it =
neighborhood.values().iterator();
					while (it.hasNext()) {
						Neighbor neighbor = it.next();
						if(!neighbor.isAlive()){ /* si no cambio su
estado alive...*/
							if(neighbor.getWarning()>MAX_WARNINGS){
/* y supero el maximo permitido de advertencias */
								remove(neighbor); /* removemos
el nodo */
							}else{
								neighbor.incrementWarning(); /*
sino  incrementamos la advertencias */
							}
							lessOne = true; /* almenos un nodo fue
actualizado */
						}else{
							neighbor.setAlive(false); /* si es la
primera advertencia del nodo, no pasa nada */
						}

					}

					if(lessOne){
						updateNeighbordhood();

					}
					fireNeighborhoodChanged(null); /* notificamos un
cambio en los nodos */

					sleep(WATCH_DOG_MILLI);
				} catch (Exception e) {
					e.printStackTrace();
					if (stream != null)
						e.printStackTrace(stream);
				}
			}
		}

		private void updateNeighbordhood() {
			for(int i=0;i<rm.size();i++){
				neighborhood.remove(rm.get(i));
			}
			rm.clear();
		}

		ArrayList<String> rm = new ArrayList<String>();
		private void remove(Neighbor neighbor) {
			rm.add(neighbor.get_addr()+"");
		}

		public static void main(String[] args) {
			StartView start = new StartView(new ControlApp());
			start.setVisible(true);
```

```
	}

	public void messageReceived(int id, Message msg) {

		if(stream!=null){
			Dump.printPacket(stream, msg.dataGet());
			stream.println("\n"+msg);
			stream.flush();
		}

		if (msg instanceof MerisMonitorPacket) {
			process((MerisMonitorPacket) msg);
		} else if (msg instanceof ManagmentMsg) {
			process((ManagmentMsg)msg);
		} else if (msg instanceof Neighbor) {
			process((Neighbor)msg);
		}
	}
	private void process(ManagmentMsg msg) {

	}
	public String getSource() {
		return source;
	}
	public void setSource(String source) {
		if (source != null && !source.trim().equals("")) {
			this.source = source;
		}
	}
	public void disconnect() {
		try {
			if (phoenix != null) {
				mif.deregisterListener(new MerisMonitorPacket(),
this);
				mif.deregisterListener(new Neighbor(), this);
				mif=null;
				phoenix.shutdown();
				phoenix = null;
			}
			this.stream.println("[" + new Date() + "] Desconectado");
			this.stream.flush();
		} catch (Exception e) {	}
	}

	public HashMap<String, Neighbor> getNeighborhood() {
		return neighborhood;
	}

	/**
	 * Procesar Paquetes de InformaciÃ³n
	 * */
	private void process(MerisMonitorPacket info) {
		Neighbor n = null;

		String source = ""+info.get_source();
		if(!neighborhood.containsKey(source)){
			n = new Neighbor();
			n.set_addr(info.get_source());
			n.set_hops((short)2);
			n.setParent(info.get_destination());
			neighborhood.put(source, n);
			fireNeighborhoodChanged(n);
		}else{
			n = neighborhood.get(source);
		}
```

```
                n.incrementParent(info.get_destination());

                n.setParent(info.get_destination());
                n.setAlive(true);
                n.setTriage(info.get_Triage());

                //if(n.get_addr()!=1)return;

                long time = info.get_timestamp();
                if(time>100000){ /* Es un valor negativo !!! */
                        time= 4294967295L - time;
                }

                if(model!=null){
                        boolean updated = false;
                        if(!continuo){
                                for(int i=0;i<model.getRowCount() && !updated;i++){
                                        String node = model.getValueAt(i,
0).toString();
                                        if(node.equals(""+info.get_source())){
                                                /* Actualizar el registro */
                                                model.setValueAt(time, i,1);
                                                model.setValueAt(info.get_Triage(),
i,3);

                                                model.setValueAt(info.get_FrR(), i, 4);
                                                model.setValueAt(info.get_FrC(), i, 5);
                                                model.setValueAt(info.get_SaO(), i, 6);

                                                updated = true;
                                        }
                                }
                        }
                        if(!updated){
                                model.addRow(new
Object[]{""+info.get_source(),time,""+System.currentTimeMillis(),info.get_Tria
ge(),info.get_FrR(),info.get_FrC(),info.get_SaO()});

                        }
                        model.fireTableDataChanged();
                }
        }

        /**
         * Procesar paquetes Vecinos
         * */
        private void process(Neighbor neighbor) {
                if(neighbor.get_addr()==0xFFFF)return; /* Paquete de informaciÃ³n
vacio */

                int triage = 0;
                int parent = 0;

                if (neighborhood.containsKey(neighbor.get_addr() + "")) {
                        triage = neighborhood.get(neighbor.get_addr() +
"").getTriage();
                        parent = neighborhood.get(neighbor.get_addr() +
"").getParent();
                        neighborhood.remove(neighbor.get_addr() + "");
                }

                neighbor.setParent(parent);
                neighbor.setTriage(triage);

                neighborhood.put(neighbor.get_addr() + "", neighbor);
                neighbor.setAlive(true);
```

```
                time = System.currentTimeMillis();
                offset = neighbor.get_offset();

                neighbor.set_offset((int)offset);

                fireNeighborhoodChanged(neighbor);
        }

        private void fireNeighborhoodChanged(Neighbor neighbor) {
                Iterator<NeighborhoodListener> it =
neighborhoodListeners.iterator();
                while(it.hasNext()){
                        it.next().fire(neighbor);
                }
        }

        public void setDataListener(DefaultTableModel model) {
                this.model=model;
        }

        ArrayList<NeighborhoodListener> neighborhoodListeners = new
ArrayList<NeighborhoodListener>();
        public void addNeighborhoodListener(NeighborhoodListener lister) {
                neighborhoodListeners.add(lister);
        }

        public DefaultTableModel getDataListener() {
                return model;
        }

        boolean continuo=false;
        public void setContinue(boolean b) {
                this.continuo=b;
        }

        public boolean toggleContinue() {
                continuo  = !continuo;
                return continuo;
        }

        public void packetReceived(byte[] data) {
                //Dump.printPacket(System.out, data);   System.out.println();

        }

}
```

2.2.1.2 NeiborhoodListener.java

```
package control;

import model.extention.Neighbor;
public interface NeighborhoodListener {
        public void fire(Neighbor neighbor);
}
```

2.2.2 View

A continuación se presenta las clases Network.java y StartView.java

2.2.2.1 Network.java

```
package view;

import java.io.FileOutputStream;
import java.io.InputStream;
import java.util.HashMap;
import java.util.Iterator;
```

```
import javax.swing.JComponent;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.table.DefaultTableModel;

import model.extention.Neighbor;

import att.grappa.Graph;
import att.grappa.GrappaAdapter;
import att.grappa.GrappaPanel;
import att.grappa.Parser;
import control.ControlApp;
import control.NeighborhoodListener;

public class Network implements NeighborhoodListener {

	DefaultTableModel model;
	ControlApp control;

	public Network(ControlApp control){
		this.control=control;
		model = new DefaultTableModel();
		model.addColumn("DirecciÃ³n");
		model.addColumn("Saltos");
		model.addColumn("Lqi");
		model.addColumn("Rssi");
		model.addColumn("Evictor");
		model.addColumn("Offset");
		model.addColumn("% InicioTx");
		model.addColumn("% FinTx");
		control.addNeighborhoodListener(this);
	}

	JTabbedPane tab;
	private void updateTopology(JTabbedPane tab2){

		if(tab2!=null){
			tab = tab2;
		}

		if(tab==null)return;

		/* Utilizar Graphviz para desplegar la topologÃ-a */
		HashMap<String, Neighbor> neighborhood =
control.getNeighborhood();

		String sample = "digraph \"Test Graph\" {\nrankdir=BT;\nnode
[shape=ellipse,style=filled,color=\"0.5,0.56,0.6\"];\n";
		sample += "N0 [label=\"Gateway\", shape=box,fontcolor=\"white\",
style=filled,color=\"0.5,0,0.3\"] \n";

		Iterator<Neighbor> it = neighborhood.values().iterator();
		while (it.hasNext()) {
			Neighbor neighbor = it.next();

			int id = neighbor.get_addr();
			int parent = neighbor.getParent();

			String color =
(neighbor.getWarning()>1)?"gray":neighbor.getColor();

			sample += "N" + id + " [label=\"Nodo " + id
				+ "\", color=\""+color+"\"] \n";
```

```
                    /* Fijar el ancho segun el numero de mensajes recibidos por
cada nodo padre indentificado */
                    neighbor.printParents();
                    neighbor.clearParents();

                    sample += "N" + id + " -> N"+parent+"\n";
             }

             sample += "}";

             try {
                    FileOutputStream fos = new
FileOutputStream("topology.dot");
                    fos.write(sample.getBytes());
                    fos.close();
                    InputStream input = Runtime.getRuntime().exec("dot
topology.dot")
                                  .getInputStream();

                    Parser program = new Parser(input);
                    program.parse();

                    Graph graph = program.getGraph();
                    GrappaPanel gp = new GrappaPanel(graph);
                    gp.addGrappaListener(new GrappaAdapter());
                    gp.setScaleToFit(true);

                    JScrollPane jsp = new JScrollPane();
                    jsp.setViewportView(gp);

                    if(tab.getTabCount()>0){
                           tab.setComponentAt(0, jsp);
                    }else{
                           tab.addTab("Topologia", jsp);
                    }
             } catch (Exception e) {
                    e.printStackTrace();
             }
      }

      public JComponent getView() {
             JTabbedPane tab = new JTabbedPane();
             /* Tabla de Vecinos */
             JTable table = new JTable(model);
             updateTopology(tab );

             tab.addTab("DescripciÃ³n", new JScrollPane(table));

             tab.addTab("Traza de Mensajes", new JScrollPane(log));
             return tab;
      }

      public void fire(Neighbor neighbor) {

             if(neighbor==null){ // Pudieron cambiar muchos nodos !!!
                    updateAll();
                    return;
             }

             boolean isNew = true;
             for (int i = 0; i < model.getRowCount(); i++) {
                    if (model.getValueAt(i, 0).equals(neighbor.get_addr())) {
                           isNew = false;
                           // actualizar datos;
                           model.setValueAt(neighbor.get_addr(), i, 0);
```

```
                        model.setValueAt(neighbor.get_hops(), i, 1);
                        model.setValueAt(neighbor.get_lqi(), i, 2);
                        model.setValueAt(neighbor.get_rssi(), i, 3);
                        model.setValueAt(neighbor.get_evictor(), i, 4);
                        model.setValueAt(neighbor.get_offset(), i, 5);
                        model.setValueAt(neighbor.get_start(), i, 6);
                        model.setValueAt(neighbor.get_stop(), i, 7);

                }
            }
            if (isNew) {
                model.addRow(new Object[] { neighbor.get_addr(),
                            neighbor.get_hops(), neighbor.get_lqi(),
                            neighbor.get_rssi(), neighbor.get_evictor(),
                            neighbor.get_offset(), neighbor.get_start(),
                            neighbor.get_stop() });
            }
            model.fireTableDataChanged();
        }

        boolean isUpdating=false;
        private void updateAll() {
            isUpdating=true;

            for(int i=0, rows = model.getRowCount();i<rows;i++){
                model.removeRow(0);
            }

            Iterator<Neighbor> it =
control.getNeighborhood().values().iterator();
            while (it.hasNext()) {
                Neighbor neighbor = it.next();
                fire(neighbor);
            }
            updateTopology(null);
            isUpdating=false;
        }

        JTextArea log;
        public void setLog(JTextArea log) {
            this.log = log;
        }
}
```

2.2.2.2 StartView.java

```
package view;

import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.io.PrintWriter;

import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
```

```
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.filechooser.FileFilter;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.TableColumn;

import util.CellRenderColor;
import util.CellRenderDateAgo;
import util.CellRenderImage;
import util.CellRenderText;
import util.OutputStreamAdapter;
import control.ControlApp;

public class StartView extends JFrame {
	private static final long serialVersionUID = 1L;
	ControlApp control;

	public StartView(ControlApp control){
		super("MonitorizaciÃ³n de Pacientes con Redes InalÃ¡mbricas de
Sensores - MERIS");
		this.control = control;

		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(800, 600);
		setLocationRelativeTo(null);

		InitializeComponents();
		network = new Network(control);

		network.setLog(log);
	}

	private void InitializeComponents() {
		menuBar = new JMenuBar();
		menu = new JMenu("AplicaciÃ³n");
		menuBar.add(menu);

		menuItemConnect = new JMenuItem("Conectar");
		menuItemConnect.setIcon(new ImageIcon("run.png"));
		menuItemConnect.addActionListener(new ActionListener() {

			public void actionPerformed(ActionEvent evt) {
				if(menuItemConnect.getText().equals("Desconectar")){
					menuItemConnect.setText("Conectar");
					control.disconnect();
				}else{
					if(control.connect(new PrintStream(new
OutputStreamAdapter(log))))
						menuItemConnect.setText("Desconectar");
				}
			}
		});
		menu.add(menuItemConnect);
		menu.add(new JSeparator());


		menuItem = new JMenuItem("Guardar");
		menuItem.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				save();
			}
		});
		menu.add(menuItem);
```

```
		menu.add(new JSeparator());

		menuItem = new JMenuItem("Cerrar");
		menuItem.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				exit();
			}
		});
		menu.add(menuItem);

		menu = new JMenu("EdiciÃ³n");
		menuBar.add(menu);

		menuItem = new JMenuItem("Borrar");
		menuItem.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				delete();
			}
		});
		menu.add(menuItem);

		menu = new JMenu("ConfiguraciÃ³n");
		menuBar.add(menu);

		menuItem = new JMenuItem("ConexiÃ³n");
		menuItem.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				configuration();
			}
		});
		menu.add(menuItem);

		menuItemContinue = new JMenuItem("Despliegue contÃ-nuo");
		menuItemContinue.addActionListener(new ActionListener() {

			public void actionPerformed(ActionEvent evt) {
				if(control.toggleContinue()){
					menuItemContinue.setText("Desplegar
Ã°ltimos");
				}else{
					menuItemContinue.setText("Despliegue
contÃ-nuo");
				}
			}
		});
		menu.add(menuItemContinue);

		menu = new JMenu("Ver");
		menuBar.add(menu);

		menuItem = new JMenuItem("Monitorizar Pacientes");
		menuItem.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				monitor();
			}
		});
		menu.add(menuItem);

		menuItem = new JMenuItem("Monitorizar Red");
		menuItem.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				topology();
			}
		});
		menu.add(menuItem);
```

```
		menu = new JMenu("Ayuda");
		menuBar.add(menu);
		menuItem = new JMenuItem("Acerca de");
		menuItem.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				about();
			}
		});
		menu.add(menuItem);

		setJMenuBar(menuBar);

		log=new JTextArea();
		log.setFont(new Font("Verdana", Font.BOLD, 14));

		/* Interfaz Inicial */
		monitor();
	}

	protected void continuo() {
		control.setContinue(true);
	}

	protected void save() {

		JFileChooser jfc = new JFileChooser();
		jfc.setDialogType(JFileChooser.SAVE_DIALOG);
		jfc.setFileSelectionMode(JFileChooser.FILES_ONLY);

		jfc.setDialogTitle("Guadar Datos - Meris");

		jfc.setFileFilter(new FileFilter() {
			@Override
			public String getDescription() {
				return "Archivos XLS";
			}

			@Override
			public boolean accept(File f) {
				return f.canWrite();
			}
		});

		if(jfc.showSaveDialog(this)==JFileChooser.APPROVE_OPTION){
			try {
				FileOutputStream fos = new
FileOutputStream(jfc.getSelectedFile());
				PrintWriter print = new PrintWriter(fos);

	print.println("Nodo,Plazo,Paciente,Triage,SaO,FrR,FrC");
				DefaultTableModel model = control.getDataListener();
				for(int i = 0; i< model.getRowCount()-1; i++){

					for(int col = 0; col<model.getColumnCount();
col++){
						print.print(model.getValueAt(i,
col)+",");
					}

					print.println();
				}
				print.flush();
				fos.close();
			} catch (Exception e) {
				e.printStackTrace();
```

```
                }

            }
        }

        protected void delete() {

            DefaultTableModel model = control.getDataListener();

            for(int i = 0,rows=model.getRowCount(); i<rows-1; i++){
                model.removeRow(1);
            }
        }

        protected void monitor() {
            JPanel panel = new JPanel();
            panel.setBackground(new Color(0xFEFEFE));

            DefaultTableModel model = control.getDataListener();

            if(model==null){
                model = new DefaultTableModel();
                model.addColumn("");
                model.addColumn("dt");
                model.addColumn("Nombre");
                model.addColumn("Triage");
                model.addColumn("FrR");
                model.addColumn("FrC");
                model.addColumn("SaO");
                model.addRow(new
Object[]{"id","Plazo","Paciente","Triage","RespiraciÃ³n","Cardica","SaturaciÃ³
n"});
            }

            JTable table = new JTable(model){
                private static final long serialVersionUID = -
8291222048038722118L;

                public TableCellRenderer getCellRenderer(int row, int
column) {
                    if(row == 0)return new CellRenderColor(true);
                    if(column == 3)return new CellRenderColor(false);
                    if(column == 1)return new CellRenderDateAgo();
                    if(column == 2)return super.getCellRenderer(row,
column);
                  return (column != 0)?new
CellRenderText():super.getCellRenderer(row, column);
                }
            };

            TableColumn col = table.getColumn("");
            col.setHeaderRenderer(new CellRenderColor(Color.white));
            col.setMaxWidth(30);


            col = table.getColumn("dt");
            col.setHeaderRenderer(new CellRenderColor(Color.white));
            col.setMaxWidth(120);

            col = table.getColumn("Nombre");
            col.setHeaderRenderer(new CellRenderColor(Color.white));
            col.setPreferredWidth(200);

            table.getColumn("SaO").setHeaderRenderer(new CellRenderImage(new
ImageIcon("sao.png")));
```

```
            table.getColumn("FrC").setHeaderRenderer(new CellRenderImage(new
ImageIcon("frc.png")));
            table.getColumn("FrR").setHeaderRenderer(new CellRenderImage(new
ImageIcon("frr.png")));
            table.getColumn("Triage").setHeaderRenderer(new
CellRenderImage(new ImageIcon("triage.png")));

            control.setDataListener(model);

            //panel.add(new JScrollPane(table),BorderLayout.CENTER);

            setContentPane(new JScrollPane(table));
            setVisible(true);
      }

      protected void log() {
            setContentPane(new JScrollPane(log));
            setVisible(true);
      }

      Network network;

      protected void topology() {
            setContentPane(network.getView());
            this.setVisible(true);
      }

      protected void disconnect() {
            control.disconnect();
      }

      protected void connect() {
            control.connect(new PrintStream(new
OutputStreamAdapter(this.log)));
      }

      protected void configuration() {
            control.setSource(JOptionPane.showInputDialog(this,"Parametros de
conexiÃ³n",control.getSource()));
      }

      protected void about() {
            JOptionPane.showMessageDialog(this, "MERIS\nCarolina Rios Fuentes
(crios@unicauca.edu.co)\nTodos los derechos reservados - 2009\nVersiÃ³n
1.0","Acerca de MERIS",JOptionPane.INFORMATION_MESSAGE);
      }

      private void exit() {
            this.setVisible(false);
            this.dispose();
            System.exit(0);
      }

      private JMenuBar menuBar;
      private JMenu menu;
      private JMenuItem menuItem;
      private JMenuItem menuItemConnect;
      private JMenuItem menuItemContinue;
      private JTextArea log;
}
```