FireCatch 1.0

Juan José Dolz Marco
Ingeniería Técnica en Informática de Sistemas

Jordi Bécares Ferrés

12 de Junio del 2012

Resumen

El proyecto que hemos realizado es un detector de incendios implementado mediante una red inalámbrica de sensores (WSN). Los elementos básicos de esta red son las motas: una placa que contiene un microcontrolador, un transceptor (radio) y diversos sensores que funcionarán como nodso sensores de la red. Al tratarse de comunicaciones inalámbricas y debido a que las motas funcionan con batería la gestión de energía se vuelve algo crítico. Utilizaremos por tanto para su programación un sistema operativo especifico para sistemas de bajo consumo TinyOS.

El funcionamiento básico del sistema es: los nodos sensores recogen los datos del entorno, comprueban si se ha producido una alarma y los envían a la estación central por medio de la radio. En caso de que se hubiera producido una alarma, envían un mensaje de alarma automáticamente a la central. Para acceder a la estación central (un PC) lo hacen dirigiendose a la mota base que se encargará de transmitir todo lo que le llegue a la radio al puerto USB del PC.

Desde la estación central se pueden monitorizar los datos de los sensores, confirmar alarmas o varias los parametros de configuración.

Índice de contenidos

1. Introducción 1

- 1.1 Justificación 1
- 1.2 Descripción 2
- 1.3 Requisitos 2
- 1.4 Enfoque y método seguido 5
- 1.5 Planificación 6
- 1.6 Recursos empleados 7
- 1.7 Productos obtenidos 9
- 1.8 Breve descripción del resto de capítulos 9

2. Antecedentes 11

- 2.1 Estado del arte 12
- 2.2 Estudio de mercado 23

3. Descripción funcional 25

- 3.1 Diagrama de bloques de la apliacion 28
- 3.2 Diagrama de casos de uso 30

4. Descripción detallada 31

- 4.1 FireProtectionSystem 31
- 4.2 FireProtectionSystemJava 43

5. Posibilidades de mejora 44

6. Viabilidad técnica 45

7. Valoración económica 46

8. Conclusiones 47

- 8.1 Conclusiones 47
- 8.2 Errores detectados 47
- 8.3 Autoevaluación 47

9. Glosario 50

10. Bibliografía 51

11. Anexos 53

Indice de figuras

Fig 1: Plan previsto 6
Fig 2: Mota que utilizamos COU24 1_2 A2 7
Fig 3: La herramienta SerialForwarder nos permite conectar el PC con la mota 9
Fig 4: Red de sensores inalámbricos 13
Fig 5: Topologías de red 20
Fig 6: Interfaz de usuario de la aplicación de Libelium 24
Fig 7: Interfaz de usuario de la aplicación de PC 26
Fig 8: Alarma de fuego (sensor de temperatura) 27
Fig 9: Alarma de fuego (botón) 27
Fig 10: Alarma confirmada 28
Fig 11: Diagrama de bloques del sistema 29
Fig 12: Casos de uso 30
Fig 13: Caso de uso alguién pulsa el botón de la mota 31
Fig 14: TemperatureSensorC: conexiones 41
Fig 15: BatterySensorC: conexiones 42
Fig 16: GUI realizada en Java con swing y awt 43

Introducción

El objetivo de este proyecto es el desarrollo de un sistema detector de incendios para edificios basado en la tecnología de redes de sensores inalámbricos (WSN).

El sistema consistirá a nivel hardware en:

- conjunto de motas cou24 que funcionarán como nodos sensores.
- 1 mota que funciona como estación base y permite la comunicación entre la red y el ordenador.
- 1 ordenador.

A nivel software consistirá en desarrollar las siguientes aplicaciones:

Una aplicación en las motas de la red que gestionaría los valores de los sensores de batería y temperatura y en caso de ser necesario enviar las correspondientes alarmas.

Otra aplicación en el ordenador que permitiría monitorizar los valores y alarmas provenientes de la red de sensores así como poder configurar los parámetros de los sensores.

En los próximos apartados se detallará como se ha desarrollado el proyecto.

1.1 Justificación:

Una de las principales preocupaciones en grandes edificios es el control y la seguridad de las instalaciones a un coste óptimo. Proveerse de sistemas de prevención garantiza la seguridad de los usuarios de las instalaciones y permite disminuir el coste de los seguros por lo que se convierte en una prioridad para los responsables.

En la actualidad tener un sistema detector de incendios es básico para poder garantizar la

seguridad de los usuarios. En el mercado existen multitud de sistemas: detectores de humo, CO2, luz, temperatura. Estos sistemas ofrecen un nivel óptimo de seguridad en las instalaciones pero presentan un problema económico: su coste de instalación. Todos estos sistemas necesitan de los propios sensores para detectar el incendio, así como todo un sistema de cableado que permita tanto alimentar a los nodos como para transmitir los datos entre los sensores y la estación de control donde se monitorizan los datos y se reciben la alarmas.

1.2 Descripción

A parte del coste directo del cableado existe también el coste asociado de la obra necesaria para realizarlo, ya sea el coste económico como en algunos casos el estético.

La solución que planteamos basándonos en la tecnología de redes de sensores inalámbricas (WSN) resolvería el problema de coste. Este tipo de instalaciones carecen de cableado, ya que se alimentan por baterías comunes (pilas AA) y la transmisión de los datos se realiza mediante comunicación inalámbrica. Ello redunda en una instalación más sencilla y más barata.

1.3 Requisitos

Con el fin de desarrollar un sistema robusto, estable, completo y que contemple todas las necesidades que conlleva implantar este sistema debería dar respuesta a:

1.Detectar incendios

(a) Detección por temperatura. Cuando la temperatura supere un umbral (TEMP_ALARM) el nodo debe enviar una alarma de incendio al centro de control.

(b) TEMP_ALARM tiene que poder configurarse desde la aplicación de usuario.

(c) El sistema pedirá un dato al sensor cada N segundos y comprobará si supera TEMP_ALARM. N ha de ser configurable por el usuario.

2. Notificar automáticamente la alarma de forma remota.

(a) Toda alarma detectada debe ser inmediatamente comunicada al centro de control.

(b) La transmisión se realizará por wifi.

3. Notificar automáticamente la alarma de forma local.

(a) La alarma será visible localmente desde el sensor mediante el uso de los leds.

4. Sistema manual para activar la alarma

(a) El nodo contará con un botón que permitirá activar la alarma de forma manual.

5. Transmisión de alarmas sin perdidas

(a) No se puede perder ninguna señal de alarma.

(b) La comunicación de alarmas se realizará mediante ACK.

6. Reconocimiento de las alarmas desde la aplicación de usuario.

(a) La aplicación debe mostrar por pantalla un aviso cuando se disparé una alarma en algun nodo.

(b) Debería mostrar:

i. Tipo de alarma (manual o automática)

ii. Que sensor detecta la alarma y que temperatura marca.

iii. Hora en la que se ha producido.

(c) El usuario debe poder confirmar la alarma desde la aplicación de escritorio.

7. Monitorizar la batería de los nodos de la red

(a) Cada nodo enviará su carga de batería periódicamente cada L segundos

(b) L ha de ser configurable desde la aplicación

(c) Cuando el nodo detecte que tiene la carga de la batería en situación critica, es decir, por debajo de un umbral (BAT_LVL_ALARM), enviará una alarma notificando su estado.

(d) Este mensaje nos servirá para controlar el correcto funcionamiento de los nodos, si el PC no recibe ningún mensaje de un nodo determinado tras L segundos disparará una alarma de aviso.

8. Mostrar los diferentes estados del sistema en el nodo:

(a) Sensor apagado. De forma periódica puede encenderse un led, conforme el nodo esté funcione correctamente.

(b) No alarma. De forma periódica puede encender un led, conforme el nodo esté funcionando correctamente.

(c) Alarma detectada

(d) Alarma recibida en la estación central

(e) Alarma reconocida.

(f) Nivel bajo de batería

(g) Fuera de cobertura

9. Sistema de protección de caídas. Los nodos no se pueden quedar "colgados" (WatchDog)

(a) El nodo debe hacer uso del WatchDog para evitar quedarse colgado.

(b) Ha de recuperar la configuración de forma automática

10. Sistema de debug de la aplicación.

(a) El usuario debe poder ver las temperaturas de los diferentes sensores.

(b) El usuario puede pedir recibir los datos cada M segundos. M ha de ser configurable.

11.Prueba de cobertura

(a) Proporcionar un sistema per comprovar si on s'instal·la el node hi ha cobertura amb el node base.

12. Sistema de activación

(a) El sensor no estará siempre encendido. Para iniciar el sensor, se hará uso del sensor de efecto Hall. Para encender el sensor deberemos de pasar un imán 2 veces seguidas por encima de éste, para apagarlo 4 veces.

13. Interfície de usuario

(a) Ha de ser simple

(b) Autoinstalable y autoejecutable

  i. script de instalación

  ii. Contiene todas las librerías necesarias

  iii. Manual de instalación

(c) Menu de ayuda en línea.

1.4 Enfoque y método seguido

Después de hacer un análisis del proyecto y sus requisitos se procedió a planificar el proyecto dividiendo los diferentes objetivos en tareas y marcar metas claras que nos permitieran valorar en cualquier momento cual era el camino que nos quedaba por recorrer.

El primer paso fue documentarse sobre el sistema operativo TinyOS con el que se controlarían los nodos sensores y NesC el lenguaje con el que serían programados.

TinyOS nos provee una serie de componentes que nos permiten interactuar con los diferentes componentes de la mota:

- HplAtm128GeneralIOC : componente que permite encender y apagar los sensores.
- AdcReadClientC() : componente que permite recoger datos del sensor.
- ActiveMessageC : componente que permite encender y apagar la radio.
- AMCSender y AMCReceiver : componentes que permiten enviar y recibir mensajes entre motas.
- TimerMilliC: permite utilizar temporizadores.

Finalizado el periodo de documentación procedemos a realizar una primera versión de la aplicación remota que instalaremos en los nodos. La depuración de esta aplicación lo realizaremos mediante la interface printf() de TinyOS que nos permite que la aplicación de la mota pueda escribir en la terminal.

Cuando se consigue una primera versión correcta de la aplicación remota, comenzamos con la aplicación de escritorio realizada en JAVA, que contará con una sencilla interfaz gráfica.

Para poder realizar la comunicación entre los nodos y la central utilizaremos una aplicación de la que nos provee TinyOS: BaseStation. Ésta tiene una única función: servir de puente entre los nodos remotos y el PC y su funcionamiento es muy simple, todo mensaje que le llega al puerto lo envia por la radio y viceversa.

Con el fin de que en la aplicación de escritorio podamos trabajar con los mensajes de TinyOS utilizaremos un programa MIG que nos creará automáticamente las clases necesarias.

El método utilizado para realizar la integración es iterativo

1.5 Planificación:

La planificación en un proyecto es imprescindible sobre todo cuando contamos con un tiempo tan limitado.

Al final el proyecto no ha cumplido los plazos debido a que han surgido varios imprevistos: trabajo nuevo con sus correspondientes cursos, etc... Han sido necesarios, pues, varios reajustes en las fechas finales de cada etapa y prescindir de algunas funcionalidades a la espera de una nueva versión.

Nombre	Fecha de inicio	Fecha de fin
Fase 1: Plan de trabajo	1/03/12	20/03/12
Realización de la propuesta	1/03/12	6/03/12
Documentación sobre TinyOS	5/03/12	10/03/12
Pruebas y tutoriales con las motas	5/03/12	10/03/12
Instalación y configuración del entorno de trabajo	5/03/12	12/03/12
Realización de la planificación de trabajo	14/03/12	20/03/12
Fase 2: Implementacion (1/1)	20/03/12	24/04/12
Programación del comportamiento local	20/03/12	29/03/12
Adaptación del ejemplo BaseStation para realizar Node-b...	2/04/12	8/04/12
Programación de las comunicaciones con el nodo base	11/04/12	24/04/12
Fase 3:_Implementación (2/2)	24/04/12	22/05/12
Comunicación PC-Nodo-base	24/04/12	27/04/12
Comunicación Nodo-base – PC	30/04/12	4/05/12
Programación de la aplicación en modo texto.	4/05/12	11/05/12
Programación del Watchdog	11/05/12	13/05/12
Creación de la interfaz gráfica para la aplicación	13/05/12	17/05/12
Pruebas finales	18/05/12	20/05/12
Diseño del manual de usuario y la ayuda en línea	20/05/12	22/05/12
Fase 4: Memoria	10/04/12	12/06/12
Fase 5: Presentación TFC	7/06/12	14/06/12
Realización del video demostrativo del funcionamento del ...	13/06/12	14/06/12
Presentación en diapositivas	7/06/12	12/06/12

Fig 1: Planificación prevista

1.6 Recursos empleados:

Hardware:

2 Motas COU24 1_2 24 A2

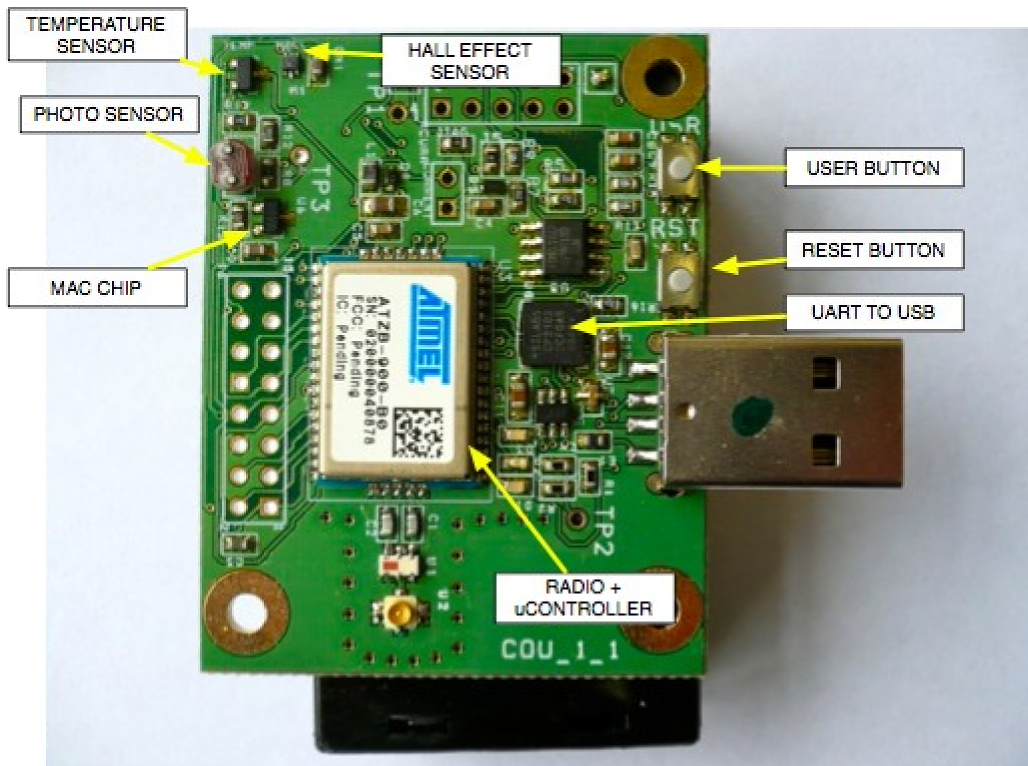

Fig 2: Mota que utilizaremos. COU24 1_2 A2

Microcontrolador:

ATmega1281:

8 bits.

Flash: 128KB

EPROM: 4KB

RAM:8KB

Pins de E/S de próposito general :54

Serial USARTs: 2

Canales ADC: 8

Transceptor AT86RF230 - es la radio, transmite en la banda de 2.4Ghz, compatible con los protocolos IEEE 812.15.4/ZigBee.

Equipada con los siguientes sensores:
Estan optimizados para optimizar el consumo de energía.

Sensor de luz: PDV-P9003-1 (400-700 nm)

Sensor de temperatura: Microchip (0-70º)

Efecto Hall:Rohm Omnipolar (interruptor mágnetico)

3 leds de colores: verde, naranja y rojo

Pilas
Cable USB
PC

Software:
Ubuntu 12.04 - Distribución del sistema operativo Linux.
TinyOS – Sistema operativo diseñado para dispositivos inalámbricos de bajo consumo.
Eclipse - Entorno de desarrollo integrado (sólo para la programación en Java de la aplicación de escritorio.
Meshprog – una herramienta de reflasheo para micros Atmel.
BaseStation: aplicación para la mota que la convierte en un bridge: todo lo que le llega vía radio lo envía al puerto UART y al contrario.

Sublime Text 2.0 – Editor de textos.
SerialForwarer: aplicación desarrollada en Java que recoge los paquetes que le llegan por el puerto serie o USB y los envía a un socket TCP. Gracias a esta aplicación nos podemos comunicar mediante sockets con la mota que esta conectada al PC (mota base).

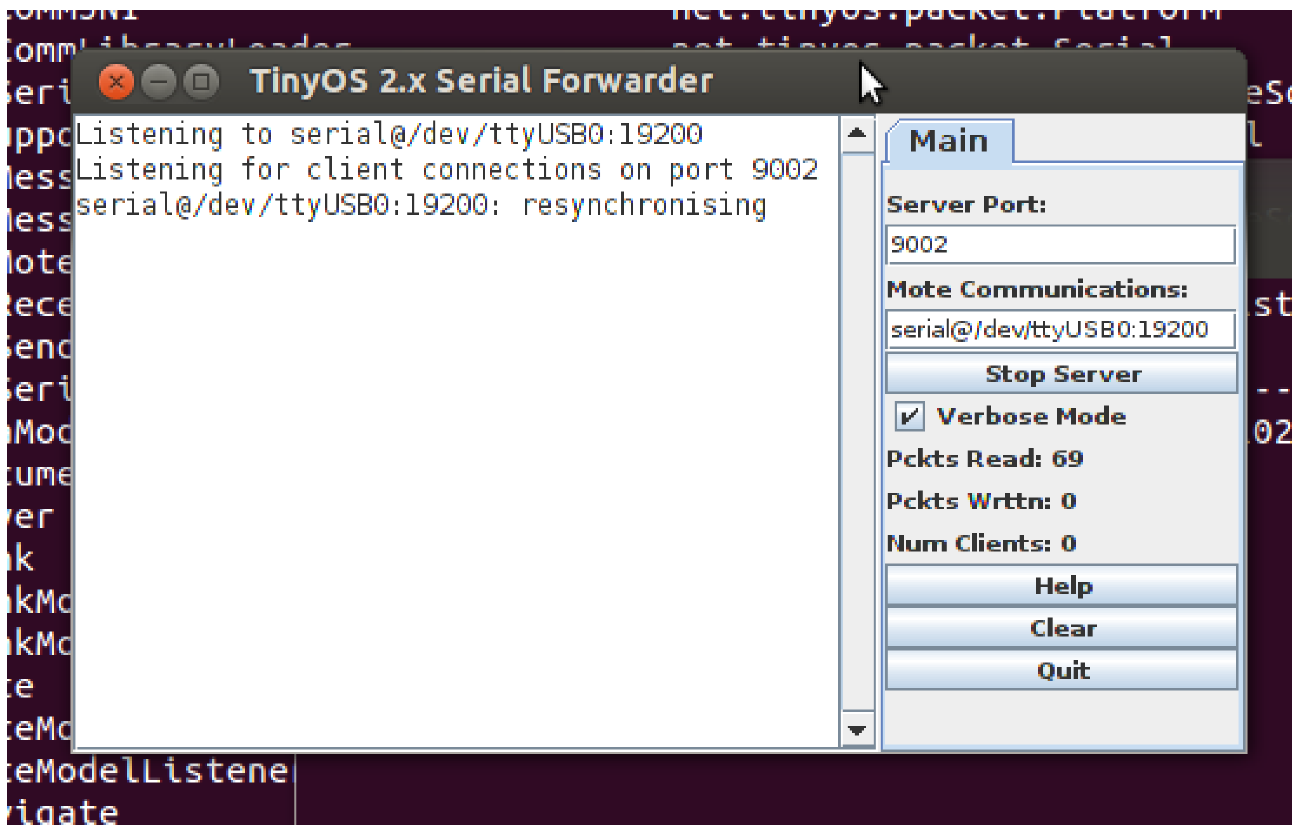

Fig 3: La herramienta SerialForwarder nos permite comunicar el PC con la mota

Git – Sistemas de control de versiones.

Open-jdk 6 – plataforma de desarrollo en java.

Nesc – lenguaje de programación derivado de C para programar WSN

1.7 Productos obtenidos

FireProtectionSystem – aplicación que instalaremos en las motas.

FireProtectionSystemJava – aplicación para el PC que hará las funciones de estación central del sistema.

1.8 Breve descripción del resto de capítulos:

Capítulo 2: Antecedentes

Estado del arte de la tecnología WSN. Motas, protocolos de comunicaciones, sistemas operativos

con especial atención a la evolución de tinyOS.

Capítulo 3: Sistema total

Descripción del funcionamiento del sistema, como interactuan las diferentes piezas, estructura de la red, diagramas de bloques.

Capítulo 4:

Descripción detallada del sistema pero sin incluir código. Se incluyen diagramas de bloques, casos de uso, etc,,.

Capítulo 5:

Posibles ampliaciones y mejoras para la próxima versión.

Capítulo 6:

Viabilidad técnica.

Capítulo 7:

Valoración económica. Presupuesto desglosado.

Capítulo 8:

- Conclusiones.
- Autoevaluación.

Capítulo 9:

Glosario.

Capítulo 10:

Bibliografía.

Capítulo 11:

Anexos: Ejecución y compilación.

2. Antecedentes

Las actuales WSN tienen sus primeros antecedentes a finales de los 70 con el programa de redes de sensores distribuidos (DSN) de la “Defense Advanced Research Projects Agency” (DARPA). En esos momentos ARPANET (Advanced Research Projects Agency Network), la red que derivaría finalmente a lo que hoy conocemos como Internet, ya estaba operativa varios años, con alrededor de 200 puntos en universidades y centros de investigación (Chong & Kumar, 2003).

Las DSNs estaban diseñadas para tener muchos sensores de bajo coste distribuidos en el espacio colaborando entre ellos pero siendo autónomos, la información que captaban se dirigía al nodo que mejor uso podía hacer de ella. Fue un programa muy ambicioso ya que en aquellos tiempos no existían todavía los ordenadores personales o las estaciones de trabajo y Ethernet empezaba a ser popular.

Los componentes de una DSN fueron identificados en un workshop de la época (Proceedings of the Distributed Sensor Nets Work- shop, 1978) e incluían: sensores (acústicos), módulos de comunicación y de proceso y software distribuido.

Los investigadores de la Universidad Carnegie Mellon llegaron incluso a desarrollar un sistema operativo orientado a comunicaciones al que llamaron Accent (Rashid & Robertson, 1981), que permitía el acceso transparente y flexible a recursos distribuidos, necesario para operar una DSN tolerante a fallos.

A pesar de que los investigadores tenían ya en mente de visión de las DSN, la tecnología no estaba preparada y el proyecto no pudo desarrollarse a nivel práctico. Los principales problemas con los que se encontraban eran que el tamaño de los nodos sensores era muy grande, y las comunicaciones sin cables no estaban muy desarrolladas (por lo que generalmente funcionaban con cableado).

Los recientes avances tecnológicos en campos tan diversos como microelectromecánica, computación, comunicaciones han inpulsado increíblemente el campo de la investigación WSN consiguiendo resultados muy cercanos a la visión original.

En 1998 empieza una nueva época en la investigación de WSN y desde entonces el sector no ha parado de captar el interés de la comunidad científica internacional. En esta etapa la investigación se ha centrado en mejorar las tecnologías de red, los sensores y el procesado de información en entornos muy dinámicos y con un uso muy eficiente de la energía. Los avances conseguidos han sido espectaculares: los sensores han llegado a ser mucho más pequeños, eficientes y baratos posibilitando el desarrollo de aplicaciones civiles con WSN en campos tan diversos como monitorización del entorno, automoción o médico.

También en esta época DARPA fue pionera al lanzar un programa de investigación llamado SensIT (2001) que logró resultados espectaculares dotando a las nuevas redes de sensores con nuevas capacidades como crear redes ad hoc, multitarea, …

Al mismo tiempo IEEE se da cuenta de las grandes posibilidades que esta tecnología ofrece y define el estándar IEEE 802.15.4 (IEEE 802.15 WPAN Task Group 4,n.d.) éste trata las necesidades de sistemas con poca transmisión de datos pero vidas útiles muy altas con alimentación limitada (pilas o baterías.) y una complejidad muy baja. La primera revisión se aprobó en mayo de 2003.

Basada en esta especificación ZigBee Alliance ha publicado el ZigBee estándard que especifica un conjunto de protocolos de comunicaciones de alto nivel que pueden ser usados por WSN y que ha llegado a ser muy popular.

El campo de las WSN está en su mejor momento, considerada como una de las tecnologías con más futuro.

2.1 Estado del arte

Una red de sensores inalámbrica (WSN wireless sensor network) consiste en un conjunto de nodos autónomos distribuidos espacialmente que disponen de una serie de sensores de bajo consumo con los que es posible monitorizar diferentes condiciones físicas o medioambientales (temperatura, sonido, vibración, presión del aire, etc...). Estos datos se envían de forma cooperativa mediante la red principal que se encarga de recibirlos y comunicarlos normalmente a un PC.

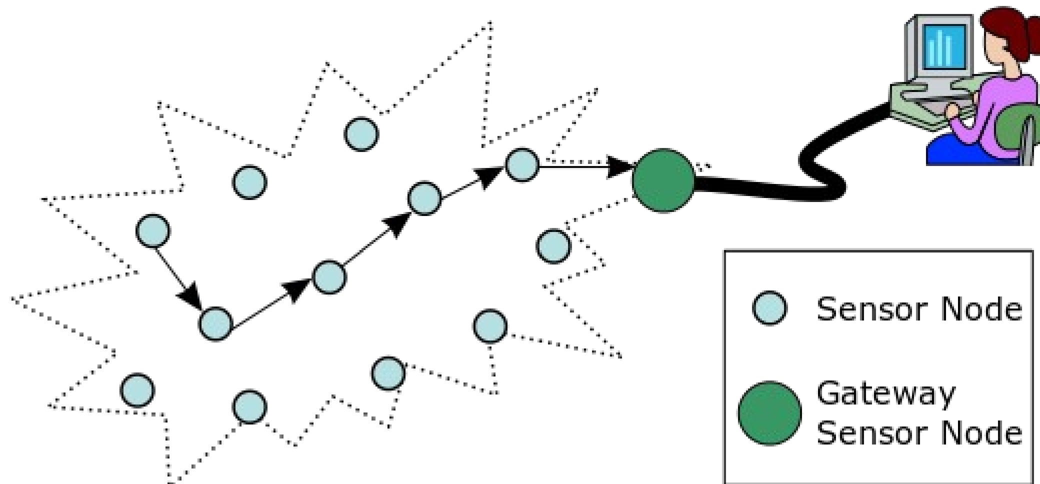

Fig 4: Red de sensores inalámbrica

Se caracterizan por su facilidad de despliegue y por ser autoconfigurables, pueden convertirse en cualquier momento en emisor, receptor, ofrecer servicios de encaminamiento entre nodos sin visión directa, así como registrar datos referentes a los sensores locales de cada nodo, otra de sus principales características es su gestión eficiente de la energía lo que les permite una tasa de autonomía elevada.

Son utilizadas habitualmente en entornos militares y cada vez más en aplicaciones civiles. Presentamos aquí algunos usos civiles de esta teconología.

- Eficiencia energética: Red de sensores que se utilizan para controlar el uso eficaz de la electricidad.
- Entornos de alta seguridad: Existen lugares que precisan de altos niveles de seguridad como por ejemplo centrales nucleares, aeropuertos, edificios del Gobierno de paso restringido. Gracias a las redes de sensores es posible detectar situaciones que con una simple cámara sería imposible.
- Sensores ambientales: Se pueden controlar múltiples variables como la temperatura, humedad, fuego, actividad sísmica. Por ejemplo, es posible instalar una red de sensores en el interior de un volcán para que gracias a los datos recogidos los científicos puedan preveer una erupción. O colocar una red en un bosque para que en caso de incendio reaccionar lo más rápidamente posible.-

Ámbito industrial: Hay fábricas que tienen controles de calidad muy complejos en los que este tipo de redes ya tienen uso.

- Automoción: Les redes de sensores son el complemento ideal a las cámaras de tráfico ya que pueden informa del tráfico en ángulos muertos que no cubren y también pueden informar a los conductores de la situación en cas de atasco o accidente y que así puedan escoger una ruta alternativa.

- Medicina: Es un campo muy prometedor ya que se pueden hacer servir para que los pacientes que lo necesiten puedan tener controladas las constantes vitales y así mejorar su calidad de vida.

- Domótica: Su alcance, precio y velocidad de despliegue hacen de este tipo de red una opción muy válida para domotizar la casa.
-

HARDWARE:Motas

Una mota es un nodo de una red de sensores inalámbrica (WSN) que es capaz de realizar algún tipo de procesamiento, recopilar datos de los sensores y comunicarse con otros nodos de la red, realizando todas estas operaciones con el mínimo gasto de energía posible (ya que son alimentados por baterías o pilas) . Para ello constan al menos de un microcontrolador, distintos tipos de sensores (luz, temperatura, …), un ADC y un transceptor (generalmente una radio).

Actualmente existen muchos tipos de motas con distintas características. Presentamos un listado de los nodos presentes en el mercado. Hemos dividido el listado en dos: normales y puerta de enlace (gateway). Por experiencia de nuestro proyecto sabemos que un nodo sensor normal no especializado puede hacer la función de puerta de enlace (gateway).

Sensor Node Name	Microcontroller	Tranceiver	Program+Data Memory	External Memory	Programming
Arago Systems WiSMote Dev	MSP430F5437	CC2520	RAM : 16 Kbytes Flash : 256 Kbytes	up to 8 Mbits	C
Arago Systems WiSMote Mini	ATMEGA128 RFA2	ATMEGA128 RFA2	RAM : 16 Kbytes Flash : 128 Kbytes E²PROM : 4Kbytes		C
AVRraven Atmel AVR#Raven wireless kit	AtMega1284p + ATmega3290p	AT86RF230	128 Kbytes + 16 Kbytes	256 kB?	C
COOKIES	ADUC841, MSP430	ETRX2 TELEGESIS, ZigBit 868/915	4 Kbytes + 62 Kbytes	4 Mbit	C
BEAN	MSP430F169	CC1000 (300-1000 MHz) with 78.6 kbit/s		4 Mbit	
BTnode	Atmel ATmega 128L (8 MHz @ 8 MIPS)	Chipcon CC1000 (433-915 MHz) and Bluetooth (2.4 GHz)	64+180 K RAM	128K FLASH ROM, 4K EEPROM	C and nesC Programming
COTS	ATMEL Microcontroller 916 MHz				
Dot	ATMEGA163		1K RAM	8-16K Flash	weC
EPIC mote	Texas Instruments MSP430 microcontroller	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash	
Egs[1]	ARM Cortex M3	CC2520, Mitsumi's class 2		2 Gbit	

		Bluetooth module			
Eyes	MSP430F149	TR1001		8 Mbit	
EyesIFX v1	MSP430F149	TDA5250 (868 MHz) FSK		8 Mbit	
EyesIFX v2	MSP430F1611	TDA5250 (868 MHz) FSK		8 Mbit	
FlatMesh FM1	16 MHz	802.15.4-compliant		660 sensor readings	Over-air control
FlatMesh FM2	16 MHz	802.15.4-compliant		660 sensor readings	Over-air control
GWnode	PIC18LF8722	BiM (173 MHz) FSK	64k RAM	128k flash	C
IMote	ARM core 12 MHz	Bluetooth with the range of 30 m	64K SRAM	512K Flash	
IMote 1.0	ARM 7TDMI 12-48 MHz	Bluetooth with the range of 30 m	64K SRAM	512K Flash	
IMote 2.0	Marvell PXA271 ARM 11-400 MHz	TI CC2420 802.15.4/ZigBee compliant radio	32 MB SRAM	32 MB Flash	
INDriya_CS_03A14 [2]	Atmel ATmega 128L[3]	IEEE 802.15.4 compliant XBee radios	128 KB FLASH + 4 KB RAM	Expansion available	C-programming & nesC compliant
Iris Mote	ATmega 1281	Atmel AT86RF230 802.15.4/ZigBee compliant radio	8K RAM	128K Flash	nesC
KMote	TI MSP430	250 kbit/s 2.4	10k RAM	48k Flash	

		GHz IEEE 802.15.4 Chipcon Wireless Transceiver			
Mica	ATmega 103 4 MHz 8-bit CPU	RFM TR1000 radio 50 kbit/s	128+4K RAM	512K Flash	nesC Programming
Mica2	ATMEGA 128L	Chipcon 868/916 MHz	4K RAM	128K Flash	
Mica2Dot	ATMEGA 128		4K RAM	128K Flash	
MicaZ	ATMEGA 128	TI CC2420 802.15.4/ZigBee compliant radio	4K RAM	128K Flash	nesC
Monnit WIT	TI CC1110	868/900 MHz	4K RAM		C#
Mulle	Renesas M16C	Atmel AT86RF230 802.15.4 / Bluetooth 2.0	31K RAM	384K+4K Flash, 2 MB EEPROM	nesC, C programming
NeoMote	ATmega 128L	TI CC2420 802.15.4/ZigBee compliant radio	4K RAM	128K Flash	nesC
Nymph	ATMEGA128L	CC1000		64 kB EEPROM	
PowWow	MSP430F1612	TI CC2420 802.15.4/ZigBee compliant radio	55kB Flash + 5kB RAM		C programming : MSPGCC, IAR
Preon32	ARM Cortex M3	Atmel AT86RF231 (2.4 GHz)	64kB RAM + 256kB Flash	8 Mbit	Java
Redbee	MC13224V	2.4 GHz 802.15.4	96 KB RAM + 120 KB Flash		GCC (see mc1322x.devl.org), IAR

Rene	ATMEL8535	916 MHz radio with bandwidth of 10 kbit/s	512 bytes RAM	8K Flash	
SenseNode	MSP430F1611	Chipcon CC2420	10K RAM	48K Flash	C and NesC programming
Shimmer	MSP430F1611	802.15.4 Shimmer SR7 (TI CC2420)	48 KB Flash 10 KB RAM	2 GB microSD Card	nes C and C Programming
SunSPOT	ARM 920T	802.15.4	512K RAM	4 MB Flash	Java
Telos	MSP430		2K RAM		
TelosB	Texas Instruments MSP430 microcontroller	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash	
Tinynode	Texas Instruments MSP430 microcontroller	Semtech SX1211	8K RAM	512K Flash	C Programming
T-Mote Sky	Texas Instruments MSP430 microcontroller	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash	
Waspmote	Atmel ATmega 1281	ZigBee/802.15.4/ DigiMesh/RF , 2.4 GHz/868/900 MHz	8K SRAM	128K FLASH ROM, 4K EEPROM, 2 GB SD card	C/Processing
weC	Atmel AVR AT90S2313	RFM TR1000 RF			
Wireless RS485	Atmega 128L	Chipcon CC2420 + Amplifier 250 kbit/s 2.4 GHz	4k RAM	128k Flash	

		IEEE 802.15.4			
XYZ	ML67 series ARM/THUMB microcontroller	CC2420 Zigbee compliant radio from Chipcon	32K RAM	256K Flash	C Programming
XM1000	TI's MSP430F2618	TI's CC2420	8K RAM	116K FLASH ROM	
Zolertia Z1	Texas Instruments MSP430F2617	Chipcon CC2420 2.4 GHz IEEE 802.15.4 Wireless Transceiver	8 KB RAM	92 KB Flash	C, nesC
FireFly	Atmel ATmega 1281	Chipcon CC2420	8K RAM	128K FLASH ROM, 4K EEPROM	C Programming
Ubimote1	TI's CC2430 SOC based on 8051 Core	TI's CC2430	8K RAM	128K FLASH ROM	C Programming
Ubimote2	TI's MSP430F2618	TI's CC2520	8K RAM	116K FLASH ROM	C Programming
VEmesh	TI MSP430	Semtech SX1211/ 1231, TI TRF6903	512B RAM	8K FLASH	Over-the-air Programming

List of gateway sensor nodes

List of Gateway Nodes

	Microcontroller	**Tranceiver**	**Interface (USB/Serial/ Wifi/Ethernet)**	**Program Memory**	**External Memory**
DWARA_CS_03A30	Atmel ATmega 128L[4]	WPAN:IEEE 802.15.4 compliant	UART based serial	128 KB Flash	---

		XBee radio GSM/GPRS Modem:SIM300S	connection to SIM300S		
Shimmer Span	MSP430F1611	802.15.4 Shimmer SR7 (TI CC2420)	USB Flashdrive - Doesn't block adjacent USB ports	10 KB RAM 48 KB Flash	
Stargate	IntelPXA255	802.11	Serial connection to WSN	64 MB SDRAM	32 MB Flash
FlatMesh FMG-S	16 MHz	802.15.4-compliant	Serial connection to FlatMesh FM1, FM2		660 sensor readings
VEmesh	TI MSP430	Semtech SX1211/1231, TI TRF6903	MODBUS, USB, RS-232/485, TCP/IP	2K RAM + 16K FLASH	Expansion available

Topologías de red:

Las topologías de red más habituales son:

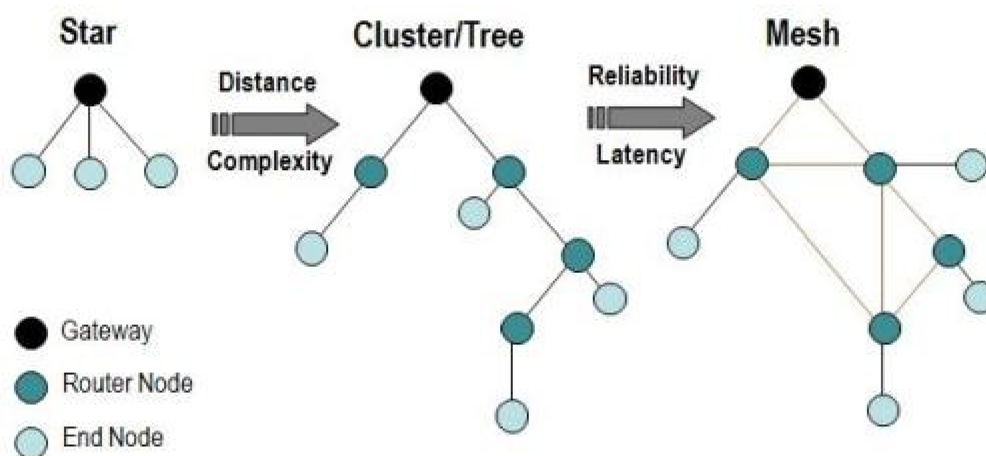

Fig 5: Topologías de red

La **topología de estrella** en que todos los nodos sensores se comunican con un nodo base

(BaseStation) que es el que pasa los datos a la estación central (PC). Es la que utilizaremos en nuestro proyecto.

La **topología cluster/tree** en que los nodos se organizan en forma de arbol, con nodos intermedio de encaminamiento en una estructura descentralizada. Es conveniente cuando las distancias o complejidad aumentan.

Existe también la posibilidad de **redes malla** con una estructura distribuida y más tolerantes a fallos porque la caída de un nodo no repercute en una caída total de sistema.

Estandares y protocolos

En el área de las WSN, varios estándares están aún desarrollándose o consolidándose. Las estandarizaciones se están llevando a cabo por parte de Institute of Electrical and Electronics Engineers (IEEE), el Internet Engineering Task Force (IETF), el International Society for Automation (ISA) y la HART Communication Foundation, etc.

IEEE 802.15.4 es un estándar que define el nivel físico y el control de acceso al medio (MAC) de redes inalámbricas de área personal con tasas bajas de transmisión de datos (*low-rate wireless personal area network*, LR-WPAN). Éste es la base de las especificaciones ZigBee y WirelessHART que se dedicaran a ofrecer una solución completa de las capas superiores que IEEE no cubre.

Las características de IEEE 802.15.4 son (IEEE 802.15 WPAN Task Group 4, n.d.):

• Tasas de transmisión de 250 kbps, 40 kbps, y 20 kbps.
• Dos modos de direccionamiento; 16-bit short y 64-bit IEEE direcccionamiento.
• CSMA-CA Control de acceso al medio.
• Establecimiento automático de la red por el coordinador.
• Protocolo full handshake para una transferencia segura.
• Gestión de energía, para asegurar bajos consumos de energía.
• 16 canales en la banda 2.4GHz ISM, 10 canales en la banda 915MHz ISM un canal en la banda 868MHz.

ZigBee: Es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicaciones inalámbricas para su utilización en radiodifusión digital de bajo consumo, basada en el estándard IEEE 802.15.4 de redes inalámbricas de área personal (WPAN).
Su objetivo son las aplicaciones que requieren comunicaciones seguras con una tasa de envío de datos baja y maximizar la vida útil de sus baterías.
El ámbito en el que se prevee tenga más éxito esta tecnología es la domótica gracias a las siguientes características:

- Bajo consumo.
- Topología de red en malla (mesh).
- Su fácil integración (se pueden fabricar nodos con muy poca electrónica).

WirelessHART: Es una tecnología de red en malla que opera en la banda de radio ISM (Industria, Científica y Médica) de 2,4GHz. Utiliza radios compatibles con el estándar IEEE 802.15.4, modulación DSSS y saltos en frecuencia basados en envíos paquete a paquete, la topología está diseñada para a que opere como malla pero también puede funcionar con topologías tipo estrella o ramificada.

Sistemas operativos para motas

En la actualidad hay variedad de sistemas operativos para motas que describimos a continuación:
Contiki: Ha estado diseñado para cubrir las necesidades de los sistemas empotrados con poca memoria. Consiste en un núcleo orientado a eventos el cual hace uso de procesos sobre los cuales los programas son cargados y descargados dinámicamente, se comunica entre procesos enviando mensajes a través de eventos..
Puede funcionar en una variedad de plataformas, desde microcontroladores empotrados como el MSP430 y el AVR como con Pcs.

Erika Enterprise: Es un sistema operativo de código abierto, incluye RT-Druid el cual es un entorno de desarrollo distribuido como un conjunto de plugins de Eclipse.
La idea principal es poder obtener un sistema básico y utilizable para aplicaciones en el ámbito de la automoción así como para sistemas empotrados que requieran de un soporte de tiempo real, es ideal para redes de sensores.

Nano-RK: És un (RTOS) Sistema operativo de tiempo real desarrollado en la Universidad Carnegie Mellon y diseñado para ser ejecutado en microcontroladores de redes de sensores. Soporta tareas de prioridad fija, Nano significa que consume pocos recursos, y RK (Resource Kernel) indica al sistema como ha de consumir los recursos, está escrito en C.

TinyOS: Es un sistema operativo de código abierto basado en componentes para redes de sensores inalámbricas, escrito con el lenguaje NesC como un conjunto de tareas y procesos que colaboran entre si, diseñado para incorporar novedades rápidamente y para funcionar bajo las importantes restricciones de memoria que se dan en redes de sensores.
Está desarrollado por un consorcio liderado por la Universidad de California en Berkeley en cooperación con Intel Research. Las aplicaciones para Tinyos se escriben en NesC, un dialecto del lenguaje C.
Proporciona interfícies, módulos y configuraciones específicas que permiten a los programadores construir programas con una serie de módulos que hacen tareas específicas.
Los módulos de TinyOS ofrecen interfícies para los tipos estándares de entradas i salidas de hardware y sensores.
Como ya se ha comentado previamente en este proyecto utilizamos el sistema operativo TinyOS.

LiteOS: Es un sistema operativo de tiempo real (RTOS) desarrollado por la Universidad de Illinois, se ajusta a los nodos de sensores de memoria limitada, permite a los usuarios operar redes de sensores inlámbricas como lo hace Unix, proporciona un entorno de programación familiar basado en Unix y C, es de código abierto y está escrito en C.

2.2 Estudio de mercado

Al analizar el mercado nos encontramos con diversas soluciones para la detección de incendios en edificios, pero la mayoría de ellos pensados para uso industrial no son inalámbricos como por ejemplo los de la empresa NetSafety Monitoring (http://www.netsafety.com) con lo que el tema de la gestión de energía deja de ser crítico.

La detección de incendios mediante redes de sensores inalámbricas si está presente en el mercado de detección de incendios forestales donde podemos encontrar una solución similar a la planteada en este proyect realizada por la empresa Liberium
http://www.libelium.com/wireless_sensor_networks_to_detec_forest_fires/.

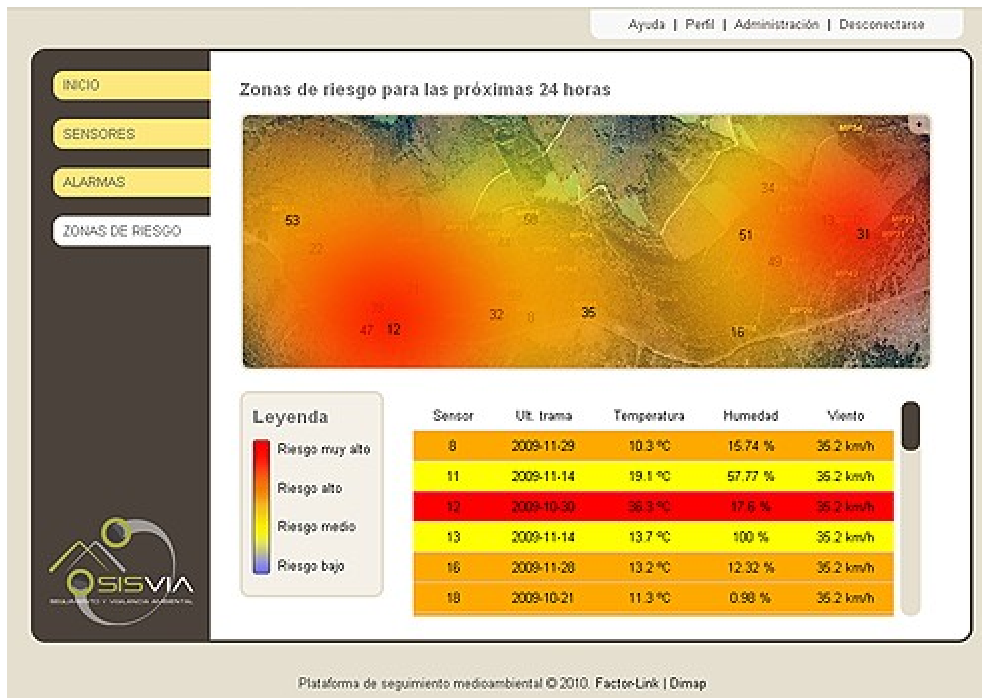

Fig 6: Interface de usuario de la aplicación central de Liberium

En edificios sigue habiendo una preferencia por sistemas cableados y las WSN se utilizan cuando éstos no son posibles. Nuestra solución, por cierto, no podría utilizarse en exteriores porque nuestras motas no son las indicadas.

Por último destacar que nuestra solución aún no está suficientemente madura para su comercialización y necesitaría, al menos, depurar errores esporádicos en la recepción de paquetes y mejorar la gestión de energía antes de poder acceder al mercado.

3. Descripción funcional

El funcionamiento de este sistema esta compuesto de varios procedimientos que pasamos a detallar:

EN LA MOTA:

- Cada 500 ms el sensor de temperatura recoge los datos y el sistema verifica si se ha superado el umbral TEMP_ALARM. En caso de que asi sea se crea una alarma y se envía inmediatamente al sistema central via la mota base con ACK. (El valor inicial 500 ms puede ser variado por el usuario).
- Cada 1000 ms los sensores de carga de batería y temperatura recogen los datos y los envían al sistema central vía la mota base.
- Si la carga de batería es menor que un valor umbral BAT_LIMIT se crea una alarma de carga de batería baja y se envía automáticamente al sistema central vía la mota base con ACK.
- En caso de que en cualquier momento un usuario pulse el botón de usuario de la mota definido en el sistema como alarma manual se crea una alarma MANUAL_ALARM y se envía automáticamente al sistema central vía la mota base con

EN EL PC:

Desde la aplicación de podemos monitorizar los valores enviados por las motas.

Fig 7: Interfaz de usuario de la aplicación de PC

Esta es la pantalla inicial cuando arrancamos la aplicación. Se nos ofrece una pequeña información de como funciona. También podemos ver la información del último paquete recibido, aparece la mota, la batería y la temperatura.

Podemos variar los valores de los temporizadores o el valor umbral a partir del que saltará la alarma de temperatura.Si variamos el limite de temperatura a 30º saltará la alarma de temperatura:

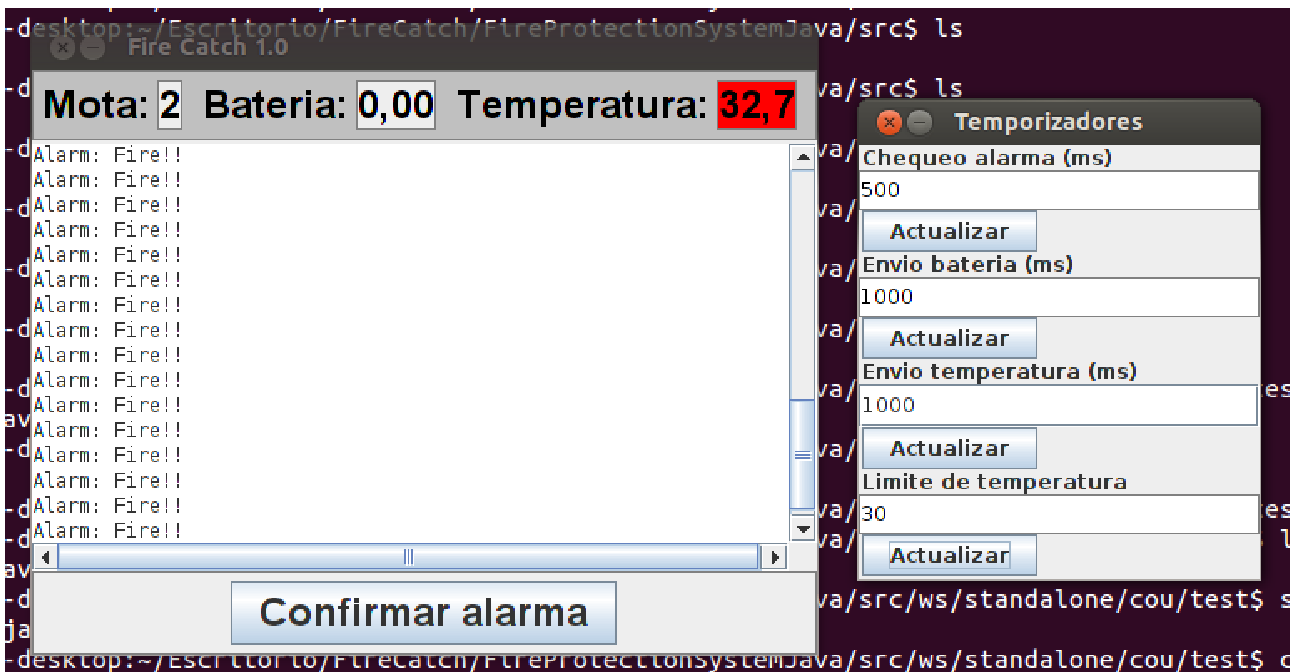

Fig 8: Alarma de fuego (sensor de temperatura)

Si alguién pulsa el botón de usuario salta la alarma manual:

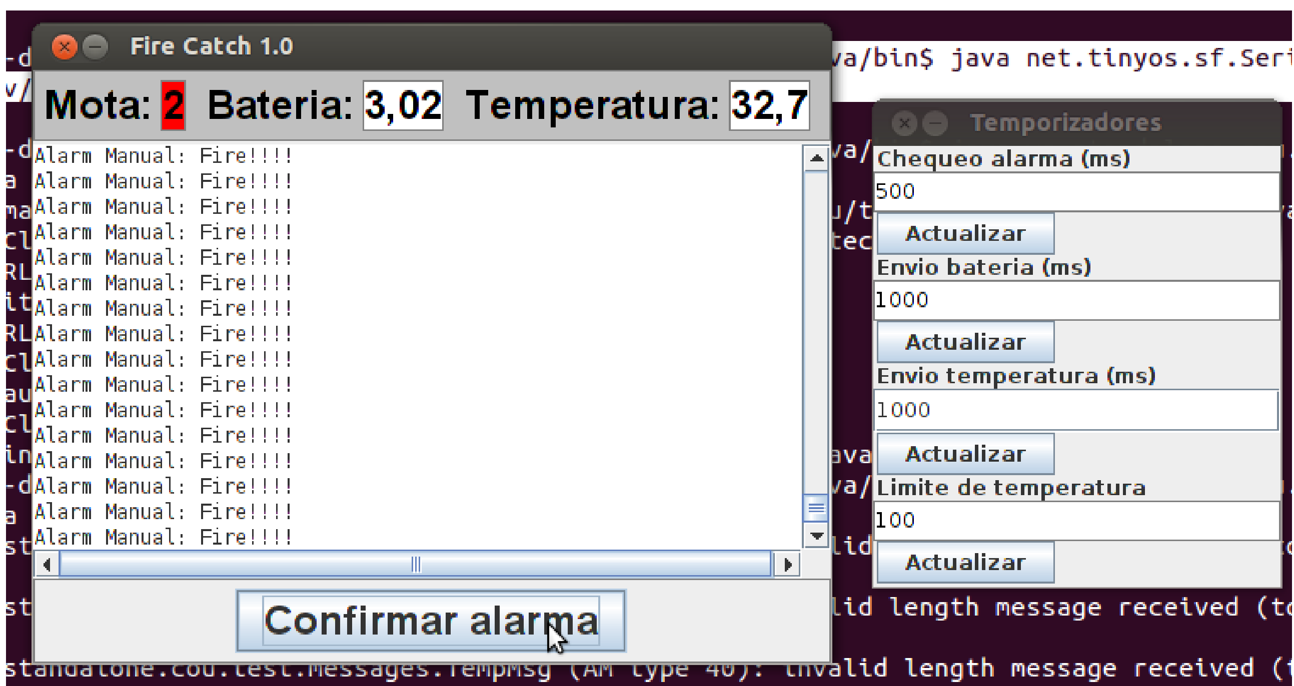

Fig 9: Alarma de fuego (botón)

Cuando confirmemos la alarma, volveremos al estado normal:

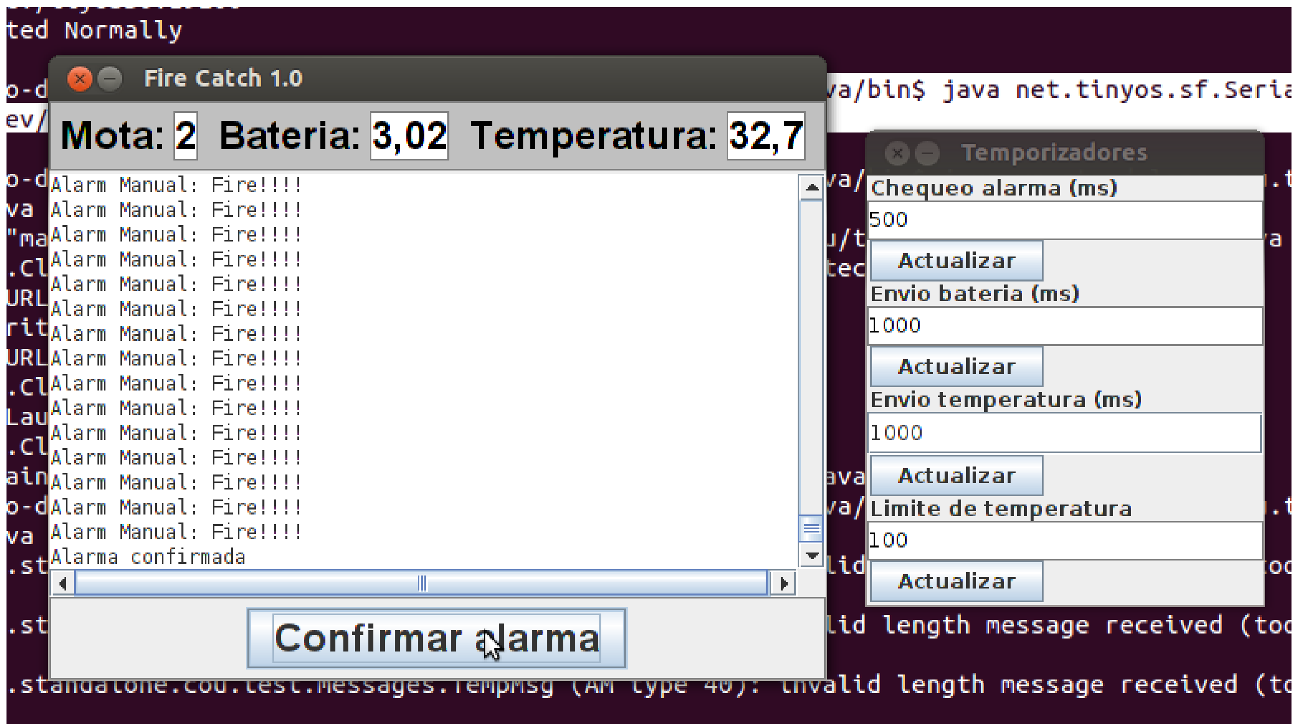

Fig 10: Alarma confirmada

3.1 Diagrama de bloques de la aplicación:

Los sensores de temperatura y batería recogen los datos y el microcontrolador los procesa y mediante la radio los envía. La mota base, que tiene cargado un programa especial llamado BaseStation lo único que hace es pasar la información que le llega por la radio al puerto UART.

De esta manera llega la información hasta la aplicación del escritorio. Desde la aplicación de escritorio si se necesita comunicar algo a alguna mota se hace el camino inverso. Envía el dato indicando como dirección la id de la mota con la quiere comunicarse y lo envía por el puerto UART. Alli la mota base lo pasa a su radio y lo transmite.

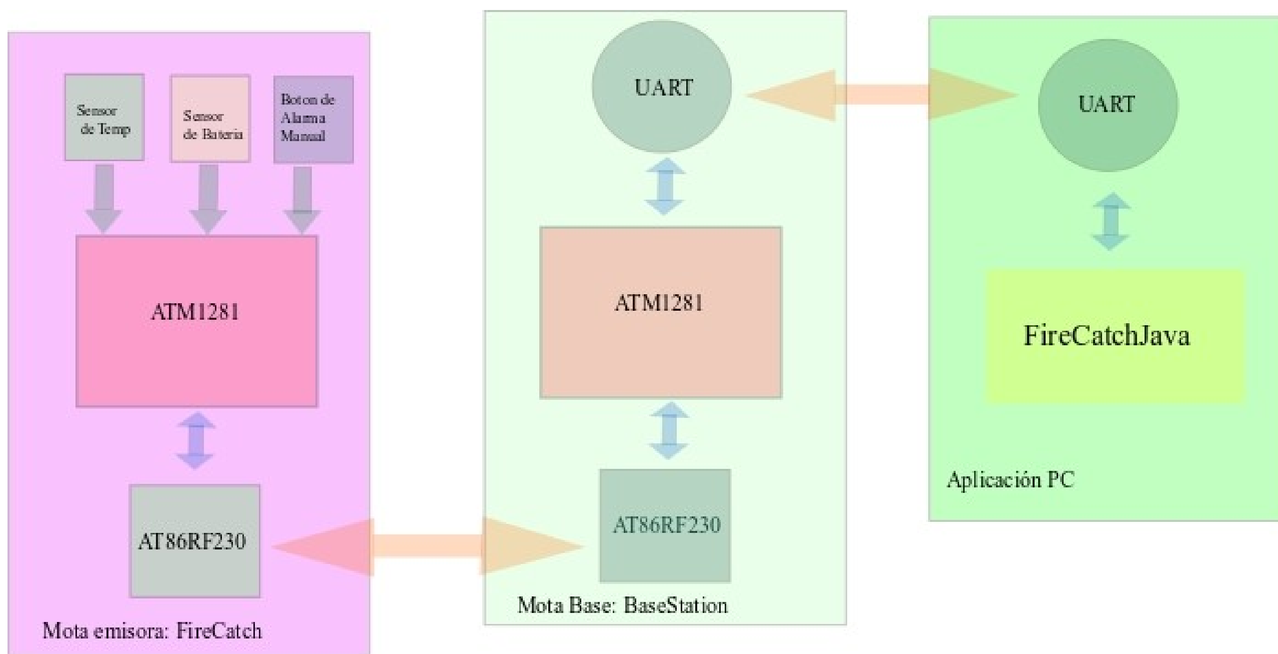

Fig 11: Diagrama de bloques del sistema

Tal como esta planteada la aplicación la red es una red totalmente centralizada (topología de red: estrella). Todas las motas comunican con la mota base directamente si quieren enviar información a la central.

3.2 Diagrama de casos de uso

Aquí tenemos los casos de uso con los usuarios FireProtectionSystem, FireProtectionSystemJava y el usuario humano en el PC.

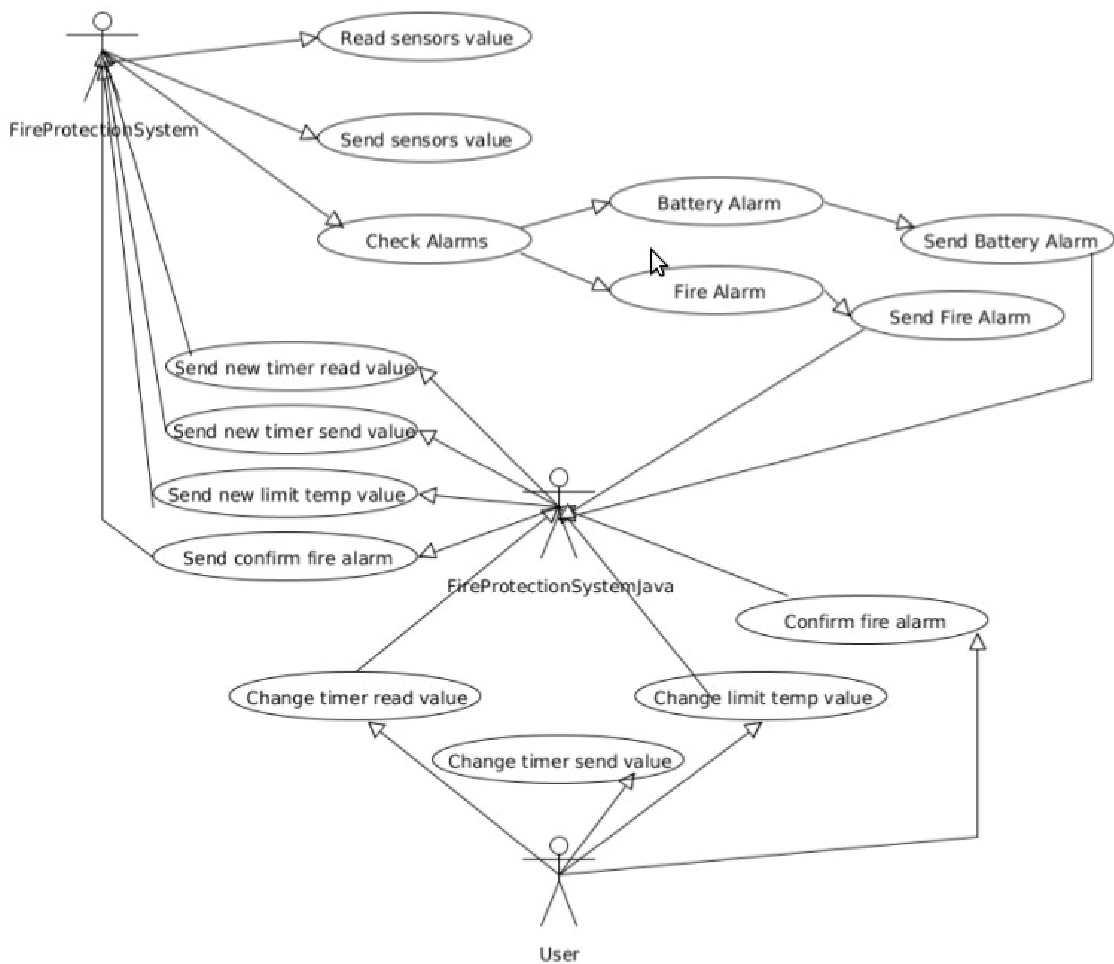

Fig 12: Casos de uso 1

Aquí tenemos el caso de uso para la alarma manual de emergencia. El usuario seria un usuario local a la mota

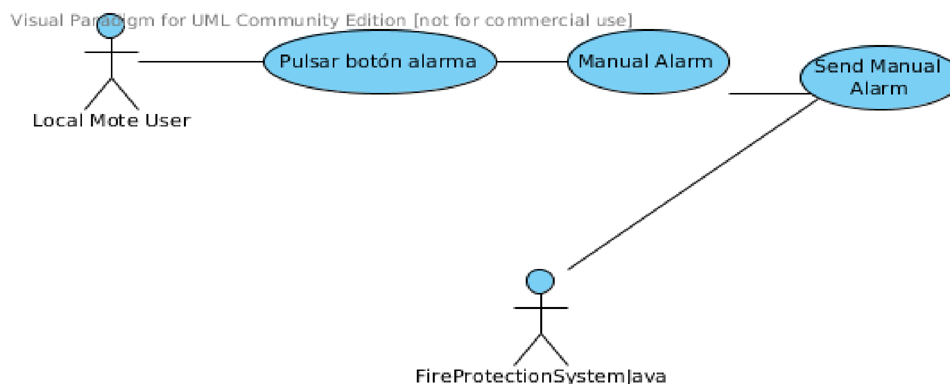

Fig 13: Caso de uso: alguién pulsa el botón de la mota

4. Descripción detallada

Pasamos a describir detalladamente la construcción de las dos aplicaciones que forman FireCatch: la aplicación que ejecutamos en la mota FireProtectionSystem y la aplicación de escritorio FireProtectionSystemJava.

4.1. FireProtectionSystem

Hay 3 temporizadores definidos que se arrancan al inicio.

El primer paso que realiza la aplicación es encender la radio de la mota y automática arranca los 3 temporizadores siguientes:

- **TimerRead:** Temporizador con un período de 500 ms (se puede modificar desde la aplicación Java del PC FireProtectionSystemJava), en cada iteración si el estado es READY o READ_TEMP hace una lectura del sensor de temperatura y se comprueba si se ha producido una alarma de incendio gracias a la interfície ReturnSensor con la que conjuntamente con el componente TemperatureSensorC podemos leer los valores de los sensores, comprobar si se ha producido alguna alarma y modificar el límite de temperatura.
- **TimerSendBat:** Timer con un período de 1000ms (se puede modificar desde la aplicación Java del PC FireProtectionSystemJava), en cada iteración se lee la batería se comprueba si hay alarma y se envía un mensaje a la mota receptora gracias al componente que proporciona TinyOS: AMSender que nos permite enviar una estructura creada por nosotros (en este caso la estructura SensorMsg) encapsulada dentro de un message_t i además enciende el Led0.

- **TimerSendTemp:** con un período de 1000ms (se puede modificar desde la aplicación Java del PC FireProtectionSystemJava), en cada iteración se lee la temperatura y se envía un mensaje a la mota receptora gracias al
componente que proporciona TinyOS: AMSender que nos permite enviar una estructura creada por nosotros (en este caso la estructura SensorMsg) encapsulada dentro de un message_t i además enciende el Led1.

Ademas contamos con una serie de eventos programados para cuando la mota recibe mensajes:

Receive_Timer_Read.receive: Cuando se recibe el nuevo valor de temporizador de iteración de lectura/chequeo de alarma del sensor de temperatura, procede a actualizarlo.
Receive_Timer_Send_Bat.receive: Cuando se recibe el nuevo valor de temporizador de iteración de lectura/chequeo de alarma/envío del sensor de batería, procede a actualizarlo.
Receive_Timer_Send_Temp.receive: Cuando se recibe el nuevo valor de temporizador de iteración de lectura/envío del sensor de temperatura, procede a actualizarlo.
Receive_Limit_Temp.receive: Cuando se recibe el nuevo valor del limite de temperatura, se procede a actualizarlo.
Receive_Alarm_Temp.receive: Cuando se recibe la confirmación de alarma se procede a apagar el equipo.

A continuación detallaremos los archivos que componen la aplicación de la mota:

FireProtectionSystemC.nc: archivo de configuración principal de la aplicación, es donde se especifican que componentes se hacen servir así como las conexiones (wiring) definidas para conectarlos.

Componentes:

FireProtectionSystemP: Componente donde está la gestión principal de la aplicación. También se encarga de controlar la activación de los sensores mediante efecto Hall y el botón de alarma manual.
MainC: Usamos este componente porque nos proporciona la interfície Boot necesaria para arrancar la aplicación.

TimerMilliC: Componente necesario para crear los diversos temporizadores que necesitamos.
LedsC: Componente necesario para poder interacturar con los leds de las motas (encenderlos, apagarlos...).
HplAtm128GeneralIOC: Componente que enciende y apaga los sensores.
ActiveMessageC: Componente que enciende y apaga la radio.
AMSenderC: Componente necesario para poder enviar mensajes por la radio.
AMReceiverC: Componente necesario para poder recibir mensajes por la radio.
BatterySensorC: Componente desarrollado por nosotros que lee el sensor de batería y comprueba si hay que lanzar la alarma de baja batería.
TemperatureSensorC: Componente desarrollado por nosotros que lee el sensor de temperatura y comprueba si hay que lanzar la alarma de fuego.
PacketAknowledgements: Componente que nos permite acuses de recibo en los mensajes de alarma (baja batería, fuego, alarma manual).
WatchdogC: Componente que nos permite evitar que el sistema de la mota se quede "colgado", ya que en ese caso procede a hacer un reset.

Connexiones (wirings):

FireProtectionSystemP.ReturnBatterySensor -> BatterySensorC.ReturnSensor
Nos indica que el componente FireProtectionSystemP usa la interfície ReturnBatterySensor y el componente BatterySensorC se la proporciona. Lo utilizamos para leer los datos de la batería y detectar la alarma de batería baja.

FireProtectionSystemP.ReturnTemperatureSensor -> TemperatureSensorC.ReturnSensor
Nos indica que el componente FireProtectionSystemP usa la interfície ReturnTemperatureSensor y el componente TemperatureSensorC se la proporciona. Lo utilizamos para leer los datos del sensor de temperatura y detectar la alarma de incendio.

FireProtectionSystemP.Boot -> MainC
Nos indica que el component FireProtectionSystemP usa interfície Boot y el componente MainC se la proporciona. Se usa para arrancar la aplicación (Boot).

FireProtectionSystemP.TimerRead -> TimerMilliC

Nos indica que el componente FireProtectionSystemP usa la interfície TimerRead y el componente TimerMilliC se la proporciona. La usamos para iterar las lecturas al sensor de temperatura para comprobar si hay que hacer saltar la alarma de incendio.

FireProtectionSystemP.TimerSendBat -> TimerMilliC

Nos indica que el componente FireProtectionSystemP usa la interfície TimerSendBat y el componente TimerMilliC se la proporciona. La usamos para iterar los envíos de datos del sensor de batería y la comprobación de si hay alarma.

FireProtectionSystemP.TimerSendTemp -> TimerMilliC

Nos indica que el componente FireProtectionSystemP usa la interfície TimerSendTemp y el componente TimerMilliC se la proporciona. La usamos para iterar los envíos de datos del sensor de temperatura.

FireProtectionSystemP.TimerStabilize -> TimerMilliC

Nos indica que el componente FireProtectionSystemP usa la interfície TimerStabilize y el componente TimerMilliC se la proporciona. Lo usamos cada vez que encendemos los sensores como retardo que les permita inicializarse para que las primeras lecturas que nos den sean correctas.

FireProtectionSystemP.Leds -> LedsC

Nos indica que el componente FireProtectionSystemP usa la interfície Leds y el componente LedsC se la proporciona. La usamos para gestionar el encendido y apagado de los leds de las motas.

FireProtectionSystemP.SinkPowerSensors -> IO.PortG1

Nos indica que el componente FireProtectionSystemP usa la interfície SinkPowerSensors y el componente IO se la proporciona con la interfície PortG1. La usamos para gestionar el encendido y apagado de los sensores,

FireProtectionSystemP.ButtonPin -> IO.PortE7

Nos indica que el componente FireProtectionSystemP usa la interficie ButtonPin y que el componente IO se la proporciona con la interfice PortE7. La usamos para gestionar el botón de

usuario (alarma manual).

FireProtectionSystemP.powerSerie -> ActiveMessageC

Nos indica que el componente FireProtectionSystemP usa la interfície powerSerie y el componente ActiveMessageC se la proporciona. La usamos para gestionar el encendido y apagado de la radio de las motas.

FireProtectionSystemP.Packet_Bat -> AMSenderC_Bat

Nos indica que el componente FireProtectionSystemP usa la interfície Packet_Bat y el componente AMSenderC_Bat se la proporciona. La usamos par poder encapsular los mensajes con los valores del sensor de batería y así poder enviarlos.

FireProtectionSystemP.AMSend_Bat -> AMSenderC_Bat

Nos indica que el componente FireProtectionSystemP usa la interfície AMSend_Bat y el componente AMSenderC_Bat se la proporciona. La usamos para poder enviar los mensajes con los valores por la radio de la mota.

FireProtectionSystemP.Packet_Temp -> AMSenderC_Temp

Nos indica que el componente FireProtectionSystemP usa la interfície Packet_Temp y el componente AMSenderC_Temp se la proporciona. La usamos par poder encapsular los mensajes con los valores del sensor de temperatura y así poder enviarlos.

FireProtectionSystemP.AMSend_Temp -> AMSenderC_Temp

Nos indica que el componente FireProtectionSystemP usa la interfície AMSend_Temp y el componente AMSenderC_Bat se la proporciona. La usamos para poder enviar los mensajes con los valores por la radio de la mota.

.FireProtectionSystemP.Packet_Temp_Alarm -> AMSenderC_Temp_Alarm

Nos indica que el componente FireProtectionSystemP usa la interfície Packet_Temp_Alarm y el componente AMSenderC_Temp_Alarm se la proporciona. La usamos para poder encapsular los mensajes con la alarma de incendio y así poder enviarlos.

FireProtectionSystemP.AMSend_Temp_Alarm -> AMSenderC_Temp_Alarm

Nos indica que el componente FireProtectionSystemP usa la interfície AMSend_Temp_Alarm y el componente AMSenderC_Temp_Alarm la proporciona. La usamos para poder enviar los mensajes con la alarma de incendio por la radio.

FireProtectionSystemP.Packet_Bat_Alarm -> AMSenderC_Bat_Alarm

Nos indica que el componente FireProtectionSystemP usa la interfície Packet_Bat_Alarm y el componente AMSenderC_Bat_Alarm se la proporciona. La usamos para poder encapsular los mensajes con la alarma de batería baja y así poder enviarlos.

FireProtectionSystemP.AMSend_Bat_Alarm -> AMSenderC_Bat_Alarm

Nos indica que el componente FireProtectionSystemP usa la interfície AMSend_Bat_Alarm y el componente AMSenderC_Bat_Alarm la proporciona. La usamos para poder enviar los mensajes con la alarma de batería baja por la radio.

FireProtectionSystemP.Packet_Manual_Alarm -> AMSenderC_Manual_Alarm

Nos indica que el componente FireProtectionSystemP usa la interfície Packet_Manual_Alarm y el componente AMSenderC_Manual_Alarm se la proporciona. La usamos para poder encapsular los mensajes con la alarma manual de fuego y así poder enviarlos.

FireProtectionSystemP.AMSend_Manual_Alarm -> AMSenderC_Manual_Alarm

Nos indica que el componente FireProtectionSystemP usa la interfície AMSend_Manual_Alarm y el componente AMSenderC_Manual_Alarm se la proporciona. La usamos para poder enviar los mensajes de alarma manual pulsada por radio.

FireProtectionSystemP.Receive_Timer_Read -> SerialAMReceiver_Timer_Read.Receive

Nos indica que el componente FireProtectionSystemP usa la interfície Receive_Timer_Read y el componente SerialAMReceiver_Timer_Read se la proporciona con la interfície Receive. La usamos para poder recibir por radio los mensajes con el nuevo valor del tiempo de iteración de lectura y comprobación del sensor de temperatura.

FireProtectionSystemP.Receive_Timer_Send_Bat ->

SerialAMReceiver_Timer_Send_Bat.Receive
Nos indica que el componente FireProtectionSystemP usa la interfície Receive_Timer_Send_Bat y el componente SerialAMReceiver_Timer_Send_Bat se la proporciona con la interfície Receive. La usamos para poder recibir por radio los mensajes con el nuevo valor del tiempo de iteración de lectura, comprobación de alarma y envío del sensor de carga de batería.

FireProtectionSystemP.Receive_Timer_Send_Temp-> SerialAMReceiver_Timer_Send_Temp.Receive
Nos indica que el componente FireProtectionSystemP usa la interfície Receive_Timer_Send_Temp y el componente SerialAMReceiver_Timer_Send_Temp se la proporciona con la interfície Receive. La usamos para poder recibir por radio los mensajes con el nuevo valor del tiempo de iteración de lectura, comprobación de alarma y envío del sensor de temperatura.

FireProtectionSystemP.Receive_Limit_Temp -> SerialAMReceiver_Limit_Temp.Receive
Nos indica que el componente FireProtectionSystemP usa la interfície Receive_Limit_Temp y el componente SerialAMReceiver_ Limit_Temp la proporciona con la interfície Receive. La usamos para poder recibir por radio los mensajes con el nuevo valor del límite de temperatura.

FireProtectionSystemP.Receive_Alarm_Temp -> SerialAMReceiver_Alarm_Temp.Receive
Nos indica que el componente FireProtectionSystemP usa la interfície Receive_Alarm_Temp y el componente SerialAMReceiver_ Alarm_Temp se la proporciona con la interfície Receive. La usamos para poder recibir por radio los mensajes de confirmación de alarma una vez recibido procederemos a parar el sistema de la mota emisora (FireProtectionSystem).

FireProtectionSystemP.nc: Archivo módulo que contiene el código fuente del main() de la aplicación.

FireProtectionSystem.h: Archivo que contiene las constantes y estructuras que se necesitan en la aplicación.
Constantes:
READY: Constante que indica que ya podemos comenzar el proceso de lectura y control de alarmas.

ERROR: Constante que usaremos para indicar que se ha producido un error.

READ_BAT: Constante para indicar al sistema que realice una lectura del valor del sensor de batería y además que compruebe si se produce una alarma de carga de batería baja.

READ_TEMP: Constante para indicar al sistema que realice una lectura del valor del sensor de temperatura y compruebe si se produce la alarma de incendio.

ALARM_BAT: Constante para indicar al sistema que se ha producido la alarma de carga de batería baja.

ALARM_TEMP: Constante para indicar al sistema que se ha producido la alarma de incendio.

TIMER_MILI_SEND_BAT: Constante para indicar el tiempos de iteración en el que se enviarán los datos de la batería (por defecto 1000).

TIMER_MILI_SEND_TEMP: Constante para indicar el tiempo de iteración en el que se enviarán los datos del sensor de temperatura (por defecto 1000).

TIMER_MILI_READ: Constante para indicar el tiempo de iteración en el que se leerá el valor del sensor de temperatura y se comprobará si existe alarma (por defecto 500)

AM_TEMPMSG: Constante para diferenciar los diferentes Active Message que tiene la aplicación. Es necesario porque sino el receptor no sabría que está recibiendo. Para los mensajes del sensor de temperatura vale 40.

AM_BATMSG: Constante para diferenciar los diferentes Active Message que tiene la aplicación. Es necesario porque sino el receptor no sabría que está recibiendo. Para los mensajes del sensor de carga de batería vale 41.

AM_LIMITTEMP: Constante para diferenciar los diferentes Active Message que tiene la aplicación. Es necesario porque sino el receptor no sabría que está recibiendo. Para los mensajes del nuevo valor de temperatura límite vale 42.

AM_ALARMTEMP: Constante para diferenciar los diferentes Active Message que tiene la aplicación. Es necesario porque sino el receptor no sabría que está recibiendo. Para los mensajes del sensor de alarma de incendio vale 43.

AM_ALARMBATTERY: Constante para diferenciar los diferentes Active Message que tiene la aplicación. Es necesario porque sino el receptor no sabría que está recibiendo. Para los mensajes de alarma por baja carga de batería vale 44.

AM_TIMERNMSG: Constante para diferenciar los diferentes Active Message que tiene la aplicación. Es necesario porque sino el receptor no sabría que está recibiendo. Para los mensajes de nuevo valor para el temporizador de lectura y comprobación de alarma del sensor de temperatura

vale 45.

AM_TIMERLMSG: Constante para diferenciar los diferentes Active Message que tiene la aplicación. Es necesario porque sino el receptor no sabría que está recibiendo. Para los mensajes del nuevo valor del temporizador de lectura y envío de datos de carga de batería vale 46.

AM_TIMERMMSG: Constante para diferenciar los diferentes Active Message que tiene la aplicación. Es necesario porque sino el receptor no sabría que está recibiendo. Para los mensajes del nuevo valor del temporizador de lectura y envío de datos del sensor de temperatura vale 47.

AM_ALARMMANUAL: Constante para diferenciar los diferentes Active Message que tiene la aplicación. Es necesario porque sino el receptor no sabría que está recibiendo. Para los mensajes del sensor de alarma manual de incendio vale 48.

VREFMILIVOLTS: Constante para indicar al sistema los voltios con los que trabaja la mota vale 2560.

LIMIT_BAT: Constante para indicar al sistema el límite con el que se comprobará si se ha producido alarma de batería baja, vale 345 que equivale a 2 Volts.

LIMIT_TEMP: Constante para indicar al sistema el límite con el que comprobará si se ha producido
alarma de incendio, val 600 que equivale a 100oC.

ALARMBAT_OK: Constante para indicar al sistema que se ha producido alarma de batería baja.

ALARMBAT_KO: Constante para indicar al sistema que no se ha producido alarma de batería baja.

ALARMFIRE_OK: Constante para indicar al sistema que se ha producido alarma de incendio.

ALARMFIRE_KO: Constante para indicar al sistema que no se ha producido alarma de incendio.

Estructuras:

Estructura que sirve para enviar al receptor los datos de los sensores de batería y de temperatura

```
typedef nx_struct BatMsg {
	nx_uint8_t moteId;
	nx_uint16_t countsBat;
	nx_uint16_t vrefmilivolts;
} BatMsg;
typedef nx_struct TempMsg {
	nx_uint8_t moteId;
```

```
        nx_uint16_t countsTemp;
        nx_uint16_t vrefmilivolts;
} TempMsg;
```

Estructura que sirve para que el receptor pueda enviar el límite de alarma de incendio

```
typedef nx_struct LimitTemp {
        nx_uint16_t countsTemp;
} LimitTemp;
```

Estructura que sirve para indicar al receptor la alarma de incendio

Si se recibe del receptor un mensaje de este tipo será el mensaje de confirmación de la alarma

```
typedef nx_struct AlarmTemp {
        nx_uint8_t moteId;
} AlarmTemp;
```

Estructura que sirve para indicar al receptor la alarma de batería baja.

```
typedef nx_struct AlarmBattery {
        nx_uint8_t moteId;
} AlarmBattery;
```

Estructura que sirve para indicar al receptor la alarma de incendio manual.

```
typedef nx_struct AlarmManual {
        nx_uint8_t moteId;
} AlarmManual;
```

Estructura que sirve para actualizar el período en el que se leen los datos del sensor de temperatura y se comprueba si hay alarma de incendio

```
typedef nx_struct TimerNMsg{
        nx_uint32_t timerVal;
} TimerNMsg;
```

Estructura que sirve para actualizar el período en el que se leen los datos del sensor de carga de batería,se comprueba si hay alarma de incendio y se envían

```
typedef nx_struct TimerLMsg{
        nx_uint32_t timerVal;
} TimerLMsg;
```

Estructura que sirve para actualizar el período en el que se leen los datos del sensor de temperatura, se comprueba si hay alarma de incendio y se envían

```
typedef nx_struct TimerMMsg{
        nx_uint32_t timerVal;
} TimerMMsg;
```

ReturnSensor.nc: Interfície creada por nosotros que proporciona los siguientes métodos:

command uint16_t getValue() : Devuelve el valor del sensor

command uint16_t ReturnAlarm(): Devuelve si se ha producido una alarma o no

command void updateLimit(uint16_t val): Actualiza el valor del límite del sensor.

TemperatureSensorC.nc: Archivo de configuración del componente TemperaturaSensorP.

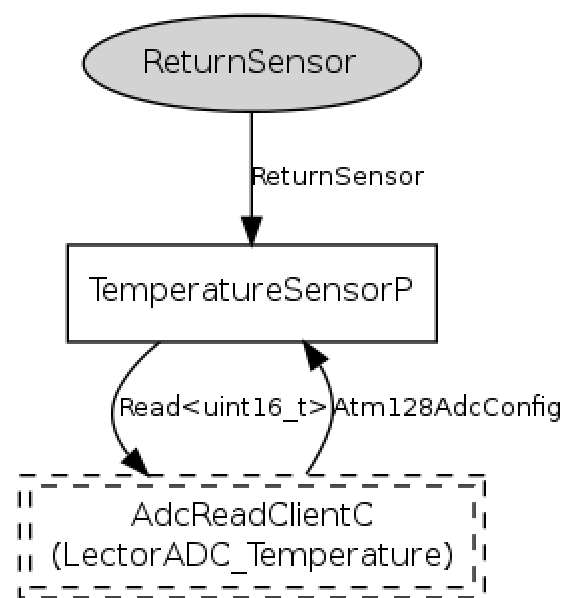

Fig 14: TemperatureSensorC : conexiones

TemperatureSensorP.nc: Archivo con el código fuente para el módulo TemperatureSensorP que nos sirve para leer los valores del sensor de temperatura, comprobar si hay alarma y actualizar el valor umbral LIMIT_TEMP.

BatterySensorC.nc: Archivo de configuración del modulo BatterySensorP.

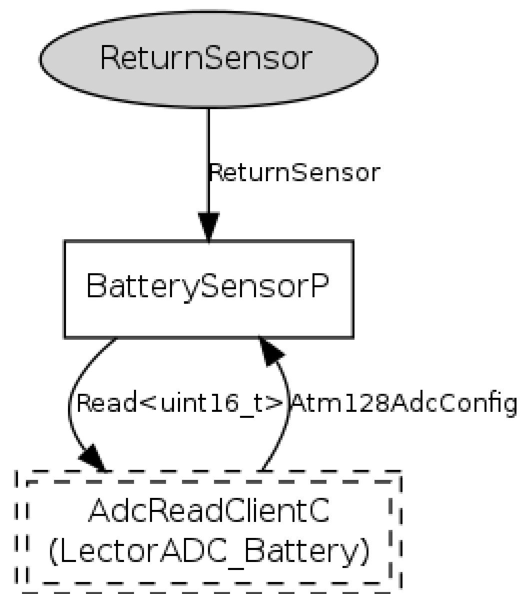

Fig 15: BatterySensorC: conexiones

BatterySensorP.nc: Archivo con el código fuente del módulo BatterySensorP que usamos para leer el sensor de batería y comprobar si se ha producido una alarma.
WatchDogC.nc: archivo con la configuración del módulo WatchDogP.
WatchDogP.nc: archivo con el código fuente del modulo WatchDogP. Se encarga de reiniciar la mota si está se "cuelga".
Watchdog: archivo del interfaz de Watchdog. Ofrece los siguiente métodos:
stop() - inicia Watchdog
start (uint32_t time) - habilita WatchDog para resetearse en el tiempo designado por time
reset() - detiene el reinicio si se ejecuta dentro del tiempo asignado

MakeFile: Archivo que configura una serie de parámetros necesarios para compilar la aplicación.
JmakeFile: Archivo en el que se especifican una serie de parámetros para que se puedan generar las clases necesarias por parte de Java para poder manejar los mensajes con la mota desde FireProtectionSystemJava. Nos ayudaremos de la herramienta mig (Message Inteface Generator) que proporciona TinyOS.

4.2. FireProtectionSystemJava

TinyOS proporciona una librería para Java MoteIF con la que podemos recibir y enviar mensajes a la mota conectada al socket que ha creado la aplicación SerialForwarder.

Se crean dos instáncias de esta clase para poder monitorizar los valores de los sensores provinientes de la mota así como las alarmas y poder enviar mensajes a la mota emisora como por ejemplo valores nuevos de los temporizadores o confirmar una alarma.

Está compuesta los siguientes archivos:

FireProtectionSystemJava.java: Es la aplicación encargada de recibir los mensajes que envía la aplicación FireProtectionSystem con los valores de los sensores o la indicación de existencia de alarma de incendio, de baja batería o manual- También se encarga de permitir al usuario cambiar los valores de los temporizadores de lectura y detección de alarmas y llegado el caso permitirle enviar la confirmación de alarma.

GraphicalUserInterface.java: Código fuente perteneciente a la interfaz gráfica de la aplicación de escritorio. Se ha hecho uso de las librerias de Java swing y awt. Adjunto captura de pantalla:

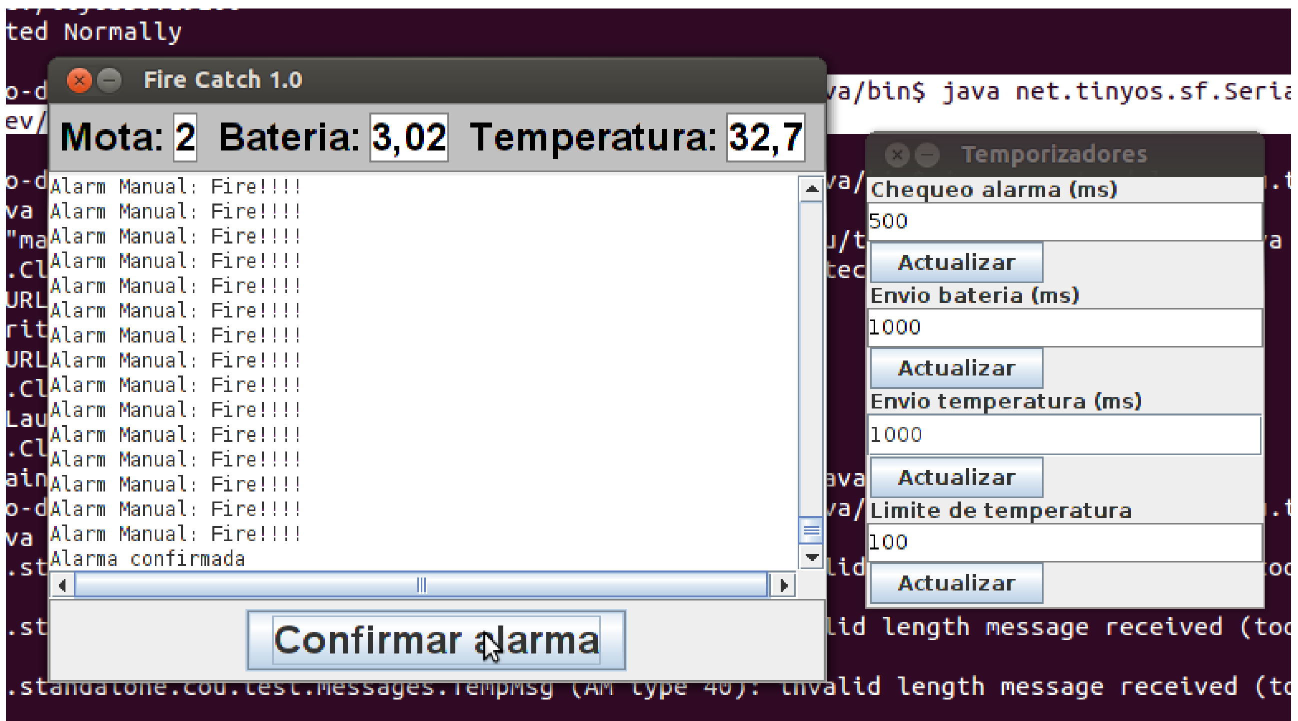

Fig 16: GUI realizado en Java usando swing y awt

Ademas contamos con todo un conjunto de archivos que son las clases generadas por el mig para poder recibir y enviar mensajes a la mota. Se generaron automáticamente al realizar el make -f jmakefile

AlarmBattery.java: basada en la estructura de AlarmBattery definida en FireProtectionSystem.h
AlarmTemp.java: basada en la estructura de AlarmTemp definida en FireProtectionSystem.h
AlarmManual.java: basada en la estructura AlarmManual definida en FireProtectionSystem.h
LimitTemp.java:basada en la estructura de LimitTemp definida en FireProtectionSystem.h
BatMsg.java: basada en la estructura de BatMsg definida en FireProtectionSystem.h
TempMsg.java:basada en la estructura de TempMsg definida en FireProtectionSystem.h
TimerNMsg.java: basada en la estructura de TimerNMsg definida en FireProtectionSystem.h
TimerLMsg.java: basada en la estructura de TimerLMsg definida en FireProtectionSystem.h
TimerMMsg.java: basada en la estructura de TimerMMsg definida en FireProtectionSystem.h

5. Posibilidades de mejora

Las posibilidades de mejora son infinitas:
En la mota:

- Optimización del uso de energía.
- Utilización del sensor de luz para la detección de incendio.
- Añadir un beeeper a la mota que suene cuando la alarma salte.
- Controlar la cobertura WIFI de la mota.
- Depurar la gestión de ACKs de las alarmas.
- Guardar el estado de la mota cuando se reinicie por cuelgue para poder volver donde estaba.

En la mota base:

- En la versión actual solo funciona como mota base pero no como nodo sensor. Era un requisito que no se ha podido implementar.

En la aplicación de escritorio:

- Gestionar la cobertura de las motas.
- Controlar las cargas de batería de las motas.
- Mejorar las interfaz gráfica: el aspecto de la apliación de Libelium que aparece en la

gráfica que adjunte sería un objetivo interesante.

- La visualización de los datos que nos llegan podría mejorarse mucho, usando Processing y algunas técnicas de visualización de datos.
- Manejar más de 1 mota.

En el sistema en general:

- Cambiar la estructura de la red de nodos de estrella a algo más descentralizado como estructura cluster/arbol.

6. Viabilidad técnica

El sistema en su estado actual precisa unas cuantas mejoras para ser viable en el entorno de producción ya que adolece de a

- Fer el sistema de comunicació sense fils més robust a les col·lisions produïdes pel medi, implementant
un sistema d'acks.
- Modificar el codi per poder monitoritzar més d'una mota a la vegada.
- Desenvolupar un sistema d'interconnexió entre múltiples nodes, perquè per exemple es pogués monitoritzar diferents sales d'una planta d'un edifici.
El punts forts d'aquest projecte son les eines utilitzades pel seu desenvolupament, Java i TinyOS ja que
actualment són projectes mantinguts i en fase emergent, així ens assegurem en un futur que aquest projecte no estigui desfasat.
El punts febles serien tots els objectius que falta per implementar.

7. Valoración económica

El proyecto aún no está lo suficientemente madura para comercializarlo pero podemos empezar a analizar los costes.

Costes en hardware:

Mota:

Elemento	Precio
Atm1281 (microcontrolador)	10,737 €
AT86RF230 (Tranceptor)	5,511 €
MCP9700E (sensor temperatura)	0,29€
BU52001GULE2 (sensor efecto Hall)	0,73€
PDV-P9G03-1 (sensor de luz)	1,96€
TOTAL	19,228€

Es decir sólo los componentes básicos de la mota ya casi son 20€.

O sea que la mota nos vendrá a salir por 20-22€ dependiendo de donde la fabriquemos y en que cantidad.

Necesitaremos una cantidad importante de pilas (2xmota).

Mano de obra:

Se tiene que tener en cuenta que en este caso:

- A pesar de que el plazo con el que teníamos para realizar el proyecto eran 4 meses en realidad el tiempo dedicado a él ha sido mucho menor.
- Se dedico bastante tiempo a la investigación, creo que demasiado, lo que ha abierto muchos frentes de mejora desde la gestión de energía en la mota, a la visualización de los datos en la aplicación de escritorio o desde la estructura de red del sistema a la posibilidad de utilizar el sensor de luz para ayudar en la detección de un incendio.
- Por supuesto hubo un período intensivo de documentación sobre TinyOS, NesC, meshprog, SerialForwarder.
- Tiempo dedicado a desarrollar el código.
- Tiempo de depuración de errores.
- Pruebas de rendimiento.

- Se ha dedicado un tiempo importante a realizar la memoria.

8. Conclusiones

8.1. Conclusiones

Desconocía absolutamente el mundo de los sistemas empotrados y precisamente por ello elegí como TFC éste. Prefería realizar el proyecto en algo que desconocía lo que me permitiría descubrir “un nuevo mundo”. No me equivoqué, así ha sido.

Las redes de sensores inalámbricas son un campo con mucho presente y aún mucho más futuro conforme la tecnología aumente las capacidades de los nodos reduciendo su consumo de energía. Sus campos de aplicación son diversos: domótica, monitorización del entorno físico, monitorización de enfermos, etc...

Por diversas circunstancias personales: nuevo trabajo, diversos cursos realizados por cuestiones laborales,etc... no le he podido dedicar el tiempo que merecía y el resultado dista de haberme dejado satisfecho.

La curva de aprendizaje es alta pues carecía de conocimientos básicos para el campo como interpretar correctamente la información técnica (esquemas del microcontrolador), conocimiento de tinyOS o NesC etc...

8.2. Errores detectados

En **FireProtectionSystem**:

- Se han detectado errores esporádicos en la recepción y envío de paquetes que aparecen como Bad_Packet.
- La detección por ACK no acaba de funcionar bien.
- Al arrancar la mota nos da lecturas erróneas a pesar del temporizador de retardo.

En **FireProtectionSistemJava.Java**:

- Problemas de perdidas de paquetes en los envíos. Sería conveniente implementar algún tipo de ACKs.

8.3. Autoevaluación

Para poder valorar el resultado del proyecto haremos una comprobación sobre los requisitos solicitados y los resultados obtenidos. En azul lo implementado, en naranja lo mejorable y en rojo

no implementado:

1.Detectar incendios

(a) Detección por temperatura. Cuando la temperatura supere un umbral (TEMP_ALARM) el nodo debe enviar una alarma de incendio al centro de control.

(b) TEMP_ALARM tiene que poder configurarse desde la aplicación de usuario.

(c) El sistema pedirá un dato al sensor cada N segundos y comprobará si supera TEMP_ALARM. N ha de ser configurable por el usuario.

2. Notificar automáticamente la alarma de forma remota.

(a) Toda alarma detectada debe ser inmediatamente comunicada al centro de control.

(b) La transmisión se realizará por wifi.

3. Notificar automáticamente la alarma de forma local.

(a) La alarma será visible localmente desde el sensor mediante el uso de los leds.

4. Sistema manual para activar la alarma

(a) El nodo contará con un botón que permitirá activar la alarma de forma manual.

5. Transmisión de alarmas sin perdidas

(a) No se puede perder ninguna señal de alarma.

(b) La comunicación de alarmas se realizará mediante ACK.

6. Reconocimiento de las alarmas desde la aplicación de usuario.

(a) La aplicación debe mostrar por pantalla un aviso cuando se disparé una alarma en algun nodo.

(b) Debería mostrar:

1. Tipo de alarma (manual o automática)
2. Que sensor detecta la alarma y que temperatura marca.
3. Hora en la que se ha producido.

(c) El usuario debe poder confirmar la alarma desde la aplicación de escritorio.

7. Monitorizar la batería de los nodos de la red

(a) Cada nodo enviará su carga de batería periódicamente cada L segundos

(b) L ha de ser configurable desde la aplicación

(c) Cuando el nodo detecte que tiene la carga de la batería en situación critica, es decir, por debajo de un umbral (BAT_LVL_ALARM), enviará una alarma notificando su estado.

(d)Este mensaje nos servirá para controlar el correcto funcionamiento de los nodos, si el PC no recibe ningún mensaje de un nodo determinado tras L segundos disparará una alarma de aviso.

8. Mostrar los diferentes estados del sistema en el nodo:

(a) Sensor apagado. De forma periódica puede encenderse un led, conforme el nodo esté funcione correctamente.

(b) No alarma. De forma periódica puede encender un led, conforme el nodo esté funcionando correctamente.

(c) Alarma detectada

(d) Alarma recibida en la estación central

(e) Alarma reconocida.

(f) Nivel bajo de batería

(g) Fuera de cobertura (No se realizó por temas de gestión de energía)

9. Sistema de protección de caídas. Los nodos no se pueden quedar "colgados" (WatchDog)

(a) El nodo debe hacer uso del WatchDog para evitar quedarse colgado.

(b) Ha de recuperar la configuración de forma automática

10. Sistema de debug de la aplicación.

(a) El usuario debe poder ver las temperaturas de los diferentes sensores.

(b) El usuario puede pedir recibir los datos cada M segundos. M ha de ser configurable.

11.Prueba de cobertura

(a) Proporcionar un sistema para comprobar si donde se instala el nodo hay cobertura con el nodo base.

12. Sistema de activación

(a) El sensor no estará siempre encendido. Para iniciar el sensor, se hará uso del sensor de efecto Hall. Para encender el sensor deberemos de pasar un iman 2 veces seguidas por encima de éste, para apagarlo 4 veces.

13. Interfície de usuario

(a) Ha de ser simple

(b) Autoinstalable y autoejecutable

i. script de instalación

ii. Contiene todas las librerías necesarias

iii. Manual de instalación

(c) Menu de ayuda en línea.

9. Glosario

TinyOS: És un sistema operativo de código abierto basado en componentes para redes de sensores inalámbricos.

MIG: Message Interface Generator, herramienta para generar las clases Java a partir de las estructuras en NesC.

COU_1_2 24: Motas con las que se ha realizado el proyecto.

802.15.4:Estándar que define la capa física y MAC de redes

ZigBee: Especificación de un conjunto de protocolos de alto nivel para redes inalámbricas de bajo consumo.

Meshprog: Programa para cargar en las motas COU24 el código NesC compilado.

GUI: (Graphical User Interface), Interfície gráfica de usuario.

JDT: Java Development Tools.

Eclipse IDE: Plataforma de código abierto de programación multilenguaje.

WSN: Red de sensores inalámbricos..

ADC: Convertidor Analógico a Digital

UART: Transmisor/Receptor Universal Asíncrono.

Java: Lenguaje de programación orientado a objetos.

UML: Lenguaje de modelado unificado..

ATmega1281: Microcontrolador de la marca ATMEL.

MCP9700: Transistor especialmente diseñado para medir la temperatura. También se caracteriza por su bajo consumo.

CP2102: Circuito integrado que hace de puente entre un bus USB y la mota.

NesC: Lenguaje de programación C optimitzado pensando en las limitaciones de memoria de las redes de sensores.

Processing: lenguaje de programación para realizar visualizaciones de datos profesionales basado en Java.

Git: sistema de control de versiones distribuido.

10. Bibliografia

TinyOS Programming ,Philip Levis & David Gay, Cambridge University Press, 2009

Building Wireless Sensor Network, Robert Faludi, O'reilly 2010

Java SE 6 -Manual Imprescindible, F.Javier Moldes Teo, ANAYA

Wireless Sensor Networks - An Introduction , Qinghua Wang & Ilangko Balasingham ,inTech,2010

http://es.wikipedia.org/wiki/Protecci%C3%B3n_contra_incendios

http://cv.uoc.es/app/mediawiki14/wiki/P%C3%A0gina_principal

http://tinyos.net/

http://es.wikipedia.org/wiki/UART

http://edison.upc.edu/curs/llum/fotometria/magnitud.html

http://tos-ide.ethz.ch/wiki/index.php

www.eclipse4you.com/?q=en/eclipse_plugins/yeti02

http://pervasive.researchstudio.at/portal/projects/meshprog

http://wikipedia.orange.es/wiki/NesC

http://es.wikipedia.org/wiki/Red_de_sensores

http://en.wikipedia.org/wiki/Sensor_node

http://en.wikipedia.org/wiki/Sensor_node

http://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes

http://en.wikipedia.org/wiki/Contiki

http://en.wikipedia.org/wiki/ERIKA_Enterprise

http://en.wikipedia.org/wiki/Nano-RK

http://en.wikipedia.org/wiki/TinyOS

http://en.wikipedia.org/wiki/LiteOS

http://es.wikipedia.org/wiki/ZigBee

https://github.com/cdwilson/nesC.tmbundle

http://nescc.sourceforge.net/

http://processing.org/

http://www.libelium.com/wireless_sensor_networks_to_detec_forest_fires/

11. Anexos

11.1. Ejecución y compilación

Todas las aplicaciones aparecen compiladas en el directorio bin correspondiente.

Pero si queremos compilarlas no tendremos más que seguir los siguientes pasos:

- Irnos al directorio de FireProtectionSystem y ejecutar: **make cou24**
- La aplicación Java es conveniente compilarla desde Eclipse como proyecto importado.

Una vez tenemos todo compilado pasamos a ejecutarlo:

1. Instalar la aplicación BaseStation en la mota base: **meshprog -t/dev/ttyUSB0 -f./build/cou24/main.srec** on ttyUSB0 es la mota que esta conectada al USB. (podemos comprobar donde esta la mota mediante motelist).
2. Instalar la aplicación FireProtectionSystem en las motas nodos sensores: **meshprog -t/dev/ttyUSB0 -f./build/cou24/main.srec** (si tenemos más de uno compilaremos la segunda como **make cou24 install**, 2 o 3 o 4 antes de instalarla (cambiaremos main.srec por main.srec.out-2)
3. Conectar la mota base al USB del sistema y ejecutar **SerialForwarder: java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB0:19200**
4. Ponerle las pilas a las motas nodo sensores (las que no son base)
5. Ejecutar la aplicación de escritorio: desde el directorio bin de FireProtectionSystemJava: **java ws.standalone.cou.test.FireProtectionSystemJava** para poder empezar a visualizar las lecturas de los sensores y recibir las alarmas que se produzcan.