Sistema de Alarma de Incendios

José Ignacio Quintana Ruiz
Ingeniería Técnica en Informática de Sistemas

Jordi Bécares Ferrés

Junio de 2.012

Resumen

Este proyecto ha consistido en el desarrollo de un sistema de detección de alarma de incendios mediante dos sensores de red inalámbrica (comúnmente conocidos como motes o WSN) modelo COU24 y un ordenador.

Las funciones de los motes están diferenciadas: uno de ellos realiza la función de sensor remoto, y el otro, además de la función de sensor, se conecta al ordenador y coordina las comunicaciones entre la red inalámbrica y la aplicación de control.

Como sensor, el mote monitoriza el estado de carga de las baterías, la temperatura ambiente y la pulsación manual de un botón. Cuando se exceden los parámetros configurados o se detecta la pulsación del botón, el sensor envía una alarma al coordinador de red. Los estados del mote y del sistema de alarma se señalizan mediante los leds que los motes incorporan.

Se ha desarrollado una aplicación que muestra las alarmas recibidas y el estado de la red de sensores, controla sus parámetros operativos y permite confirmar la recepción de alarmas.

La entrega de alarmas a la aplicación se garantiza mediante una confirmación de recepción explícita, y en caso de no recibir dicha confirmación, el sensor reenvía la alarma.

Palabras clave:

TinyOS, nesC, WSN, mote, red inalámbrica, 802.15.4,comunicación sin hilos, alarma de incendios, sistemas embebidos, sistemas empotrados, sensor, RS-232, USB, COU24, domótica.

Índice de contenidos

Índice de figuras

1. Introducción

En este trabajo de fin de carrera se desarrollará un sistema de alarma de incendios que comprenderá varios elementos hardware:

- Un mote actuará de sensor inalámbrico.
- Un mote actuará de sensor, coordinador de la red de sensores y puente de comunicación con la aplicación que se ejecuta en el ordenador.
- Un ordenador ejecutará la aplicación de gestión de alarmas.

Respecto al software se desarrollan varias aplicaciones:

- Una aplicación para el sensor remoto que controlará los valores de los sensores de temperatura y batería. En caso de detectar que se han sobrepasado los umbrales configurados o detectar la pulsación de un botón, enviará de forma inalámbrica una alarma al nodo coordinador.
- Un aplicación para el mote coordinador conectado al PC, que adicionalmente a las mediciones y control anteriores, actuará de puente entre la aplicación de control ejecutada en el PC y la red remota inalámbrica.
- Una aplicación para el ordenador que permitirá visualizar las lecturas y alarmas de la red de sensores y confirmar explícitamente cualquiera de las alarmas recibidas. También realizará los cambios operativos de la red de sensores.

1.1. Justificación

Este proyecto se destina a la ampliación de las medidas de protección contra incendios dentro de los edificios, en lugares donde no se puedan realizar cableados, dónde debido a los costes resultaría demasiado caro realizar una instalación convencional o incluso como solución provisional en edificios en construcción dónde la instalación de protección contra incendios definitiva todavía no pueda realizarse.

En cualquier caso, los costes asociados a la instalación de nodos inalámbricos serán menores que los costes de la instalación convencional si tenemos en consideración el coste de los sensores y el coste del cableado destinado a alimentación y datos.

1.2. Descripción del proyecto

Este proyecto está dirigido al control, detección y señalización de alarmas de incendios basados en la detección de temperatura ambiente excesiva.

Cumplirá una serie de requisitos:

- Controlar y notificar los disparos de alarmas manuales, de temperatura o de carga insuficiente de batería. Los nodos sensores proporcionarán medidas periódicas de la temperatura detectada, generarán las alarmas cuando se sobrepase el umbral de temperatura de alarma configurado y también permitirán el disparo manual por botón de alarmas. Para que los nodos funcionen de manera adecuada, se procederá a la monitorización de los niveles de batería así como de la calidad de la señal inalámbrica, informando al nodo central para el tratamiento de estos problemas. Los diferentes problemas que puedan detectar se mostrarán de manera visual en el propio nodo mediante los leds instalados y en el caso de las alarmas de incendio, también de manera sonora.
- Los parámetros operativos podrán actualizarse por el usuario y los sensores remotos recibirán estos parámetros tan pronto se inicien o la configuración cambie.
- El diseño del sistema será fiable, por un lado se garantizará la recepción de las alarmas en la aplicación que se ejecuta en el PC, y por otro monitorizaremos el estado de la carga de las baterías permitiendo su sustitución temprana antes de que se agoten. También proporcionará un disparo manual de alarma para los casos en que la detección se realice por intervención humana previa al disparo automático por exceso de temperatura.
- El sistema proporcionará una aplicación para mostrar el estado de los diferentes nodos, las alarmas recibidas, así como la posibilidad de cambiar los diferentes umbrales configurables y parámetros operativos de la red de sensores.
- El sistema proporcionará un interfaz web para controlar todo el sistema

1.3. Objetivos del proyecto

- Diseñar un protocolo de alarmas robusto mediante ACK explícito.
- Crear los módulos de software necesarios a instalar en los sensores inalámbricos de red.
- Permitir la configuración automática de los nodos a través de la red.
- Proporcionar la activación del sensor de temperatura mediante uso de imán sobre el sensor de efecto Hall.
- Control de temperatura cada N segundos (intervalo configurable).
- Disparo de alarma en caso de sobrepasar el umbral TEMP_ALARM.
- Disparo manual de alarma por pulsación de un botón en los sensores.
- Cancelación de alarmas en la red de sensores.
- Avisos luminosos y acústicos que reflejen el estado del sensor.
- Protección contra bloqueos de los nodos haciendo uso de un *watchdog*.
- Monitorización del nivel de carga de las baterías. Disparo de alarma cuando el nivel de batería sea bajo.
- Monitorización de la calidad de la señal y cambio de canal en caso necesario.
- Interface de visualización de alarmas, estado de los sensores y control del sistema con ayuda contextual integrada.
- Interface web.
- Programa de instalación.
- Creación la documentación asociada al software.

1.4. Enfoque y método seguido

En la fase inicial del proyecto se comprueban los recursos disponibles para la realización del proyecto, tanto el hardware, como en referencia a fuentes de información, para realizar una planificación adecuada a los plazos temporales disponibles.

Tras el examen inicial de objetivos y recursos disponibles se toman una serie de decisiones que afectarán a la planificación total de los diferentes hitos del proyecto; estas son:

- Creación modular del software en TinyOS, de forma que expongan sólo los parámetros y funciones necesarias para las capas superiores.
- Dado que la experiencia en programación java que tengo es mínima, descartar las librerías de integración java que TinyOS proporciona con aplicaciones desarrolladas en PC. La capa de integración entre la red de sensores y la aplicación a ejecutar en el PC se basará en texto plano. Todos los comandos y notificaciones que sirvan para interactuar con la red de sensores estarán expuestos en esta capa de integración.
- En una toma de contacto previa con TinyOS observé que la curva de aprendizaje y complejidad del entorno es alta. Cualquier tarea, por simple que parezca puede convertirse en compleja si no se localizan ejemplos similares. TinyOS es un entorno emergente y pese a que la documentación, ejemplos y tutoriales son diversos y múltiples a veces es complicado localizar la información que puede servir de ayuda.
- Para amortiguar en la medida de lo posible los diferentes imprevistos que puedan surgir durante la realización del proyecto, se decide asignar mayor carga de trabajo a la primera fase de creación de código para poder reasignar tareas a la segunda fase en caso necesario.

1.5. Planificación del proyecto

Tras el análisis inicial del proyecto y su descomposición en las diferentes tareas, obtenemos una asignación muy ajustada que nos permite poco margen de maniobra. El tiempo total para la consecución de los objetivos propuestos es poco más de 3 meses y medio; la asignación temporal a cada una de las tareas es muy baja.

Se expone a continuación la descomposición inicial en tareas:

Hito Principal	Actividad			Dependencias
00. Propuesta e investigación TFC				
	00.01		Consulta de fuentes online	
	00.02		Puesta en marcha entorno desarrollo	
	00.03		Petición de muestras	
	00.04		Compra hardware (buzzer)	
01. Plan de trabajo TFC				
	01.01	Doc	Elaboración Plan de Trabajo	
	01.02		Entrega Plan de Trabajo	
02. Entrega del código (1/2) - PAC2				
	02.01	Net	Protocolo de autoconfiguración nodo	
	02.02	Mote	Muestreo sensor temperatura (TEMP_ALARM) N segundos	
	02.03	Mote	Muestreo nivel batería (BATE_LVL_ALARM) L segundos	
	02.04	Mote	Sistema activación sensor efecto Hall	
	02.05	Net	Protocolo ACK notificación alarma temperatura	02.02
	02.06	Net	Protocolo notificación alarma batería/nodo alive	02.03
	02.07	Mote	Notificaciones visuales	02.05
	02.08	Mote	Disparo manual de alarma por botón	02.04; 02.05
	02.09	Mote	Protección watchdog nodo.	02.01
	02.10	Net	Comprobación cobertura con estación base	02.01
	02.11	Net	Protocolo cambio umbral TEMP_ALARM y muestreo N	02.02
	02.12	Net	Protocolo cambio tiempo muestreo nivel de batería L	02.03
	02.13	Net	Protocolo de notificación/cancelación alarma a la red	
	02.14	Doc	Revisión Documentación PAC2	
	02.15		Entrega PAC2	

(Continua en la página siguiente)

Ilustración 1 – Planificación. Descomposición en tareas (primera parte)

Hito Principal			Actividad	Dependencias
Continuación				
03. Entrega del código (2/2) - PAC3				
	03.01	Base	Interface estación base	
	03.02	Base	Modo debug de aplicación	
	03.03	App	Modelado BBDD	
	03.04	Base	Enlace estación base / BBDD	03.01; 03.03
	03.05	App	Aplicación usuario – mostrar alarma, temperatura, hora	02.05; 03.04
	03.06	App	Notificación/Cancelación alarma	02.06; 03.04
	03.07	App	Cambio de muestreo TEMP_ALARM	02.11; 03.04
	03.08	App	Ayuda contextual de la aplicación	
	03.09	App	Manual de instalación	
	03.10	App	Script instalación aplicación usuario	
	03.11	Net	*OPT- Protocolo notificación/cancelación alarma a la red*	*03.04; 03.06*
	03.12	Mote	*OPT- Notificaciones sonoras*	*00.04; 02.07*
	03.13	Base	*OPT- Medición de la potencia prueba de cobertura (dB) .*	*02.06*
	03.14	Net	*OPT- Protocolo de cambio dinámico o manual del canal RF*	*03.13*
	03.15	Web	*OPT- Interface web del sistema*	*03.04*
	03.16	Doc	Revisión Documentación PAC3	
	03.17		Entrega PAC3	
04. Memoria y Presentación TFC				
	04.01	Doc	Elaboración Memoria	
	04.02	Doc	Elaboración Presentación	
	04.03		Entrega TFC	

Ilustración 2 – Planificación. Descomposición en tareas (segunda parte)

El diagrama de Gantt obtenido es:

	Nombre de tarea	uración	Comienzo	Fin	Predec
1	**TFC Sistema de Alarma de incendios**	**92 días?**	**29/02/12**	**14/06/12**	
2	**00. Propuesta e investigación TFC**	**7 días?**	**29/02/12**	**07/03/12**	
3	00.01 Consulta de fuentes online	7 días?	29/02/12	07/03/12	
4	00.02 Puesta en marcha entorno desarrollo	7 días?	29/02/12	07/03/12	
5	00.03 Petición de muestras	7 días?	29/02/12	**07/03/12**	
6	00.04 Compra hardware (buzzer)	7 días?	29/02/12	**07/03/12**	
7	**01. Plan de trabajo TFC**	**14 días?**	**05/03/12**	**20/03/12**	
8	01.01 Elaboración Plan de Trabajo	13 días?	05/03/12	19/03/12	
9	**01.02 Entrega Plan de Trabajo**	0 días	20/03/12	20/03/12	
10	**02. Entrega del codigo (1/2) - PAC2**	**30 días**	**20/03/12**	**24/04/12**	
11	02.01 Protocolo de autoconfiguración nodo	3 días	**20/03/12**	22/03/12	
12	02.02 Muestreo sensor temperatura (TEMP_ALARM) N segundos	3 días	23/03/12	26/03/12	
13	02.03 Muestreo nivel batería (BATE_LVL_ALARM) L segundos	3 días	27/03/12	29/03/12	
14	02.04 Sistema activación sensor efecto Hall	2 días	30/03/12	01/04/12	
15	02.05 Protocolo ACK notificación alarma temperatura	2 días	02/04/12	03/04/12	12
16	02.06 Protocolo notificación alarma batería/nodo alive	2 días	04/04/12	05/04/12	13
17	02.07 Notificaciones visuales	1 día	06/04/12	06/04/12	15
18	02.08 Disparo manual de alarma por botón	1 día	07/04/12	08/04/12	14;15
19	02.09 Protección watchdog nodo.	1 día	09/04/12	09/04/12	11
20	02.10 Comprobación cobertura con estación base	3 días	10/04/12	12/04/12	11
21	02.11 Protocolo cambio umbral TEMP_ALARM y muestreo N	3 días	13/04/12	16/04/12	12
22	02.12 Protocolo cambio tiempo muestreo nivel de batería L	2 días	17/04/12	18/04/12	13
23	02.13 Protocolo de notificación/cancelación alarma a la red	2 días	19/04/12	20/04/12	
24	02.14 Revisión Documentación PAC2	2 días	21/04/12	23/04/12	
25	**02.15 Entrega PAC2**	0 días	24/04/12	24/04/12	
26	**03. Entrega del codigo (2/2) - PAC3**	**24 días**	24/04/12	**22/05/12**	
27	03.01 Interface estación base	2 días	24/04/12	25/04/12	
28	03.02 Modo debug de aplicación	2 días	26/04/12	27/04/12	
29	03.03 Modelado BBDD	1 día	28/04/12	29/04/12	
30	03.04 Enlace estación base / BBDD	2 días	30/04/12	01/05/12	27;29
31	03.05 Aplicación usuario – mostrar alarma, temperatura, hora	2 días	02/05/12	03/05/12	15;30
32	03.06 Notificación/Cancelación alarma	1 día	04/05/12	04/05/12	16;30
33	03.07 Cambio de muestreo TEMP_ALARM	1 día	05/05/12	06/05/12	21;30
34	03.08 Ayuda contextual de la aplicación	1 día	07/05/12	07/05/12	
35	03.09 Manual de instalación	1 día	08/05/12	08/05/12	
36	03.10 Script instalación aplicación usuario	1 día	09/05/12	09/05/12	
37	*03.11 OPT- Protocolo notificación/cancelación alarma a la red*	1 día	10/05/12	10/05/12	30;32
38	*03.12 OPT- Notificaciones sonoras*	2 días	11/05/12	13/05/12	6;17
39	*03.13 OPT- Medición de la potencia prueba de cobertura (dB) .*	1 día	14/05/12	14/05/12	16
40	*03.14 OPT- Protocolo de cambio dinámico o manual del canal RF*	2 días	15/05/12	16/05/12	39
41	*03.15 OPT- Interface web del sistema*	2 días	17/05/12	18/05/12	30
42	03.16 Revisión Documentación PAC3	2 días	19/05/12	21/05/12	
43	**03.17 Entrega PAC3**	0 días	22/05/12	22/05/12	
44	**04. Memoria y Presentación TFC**	**57 días?**	**10/04/12**	**14/06/12**	
45	04.01 Elaboración Memoria	55 días?	10/04/12	12/06/12	
46	04.02 Elaboración Presentación	7 días?	07/06/12	14/06/12	
47	**04.03 Entrega TFC**	0 días	14/06/12	14/06/12	

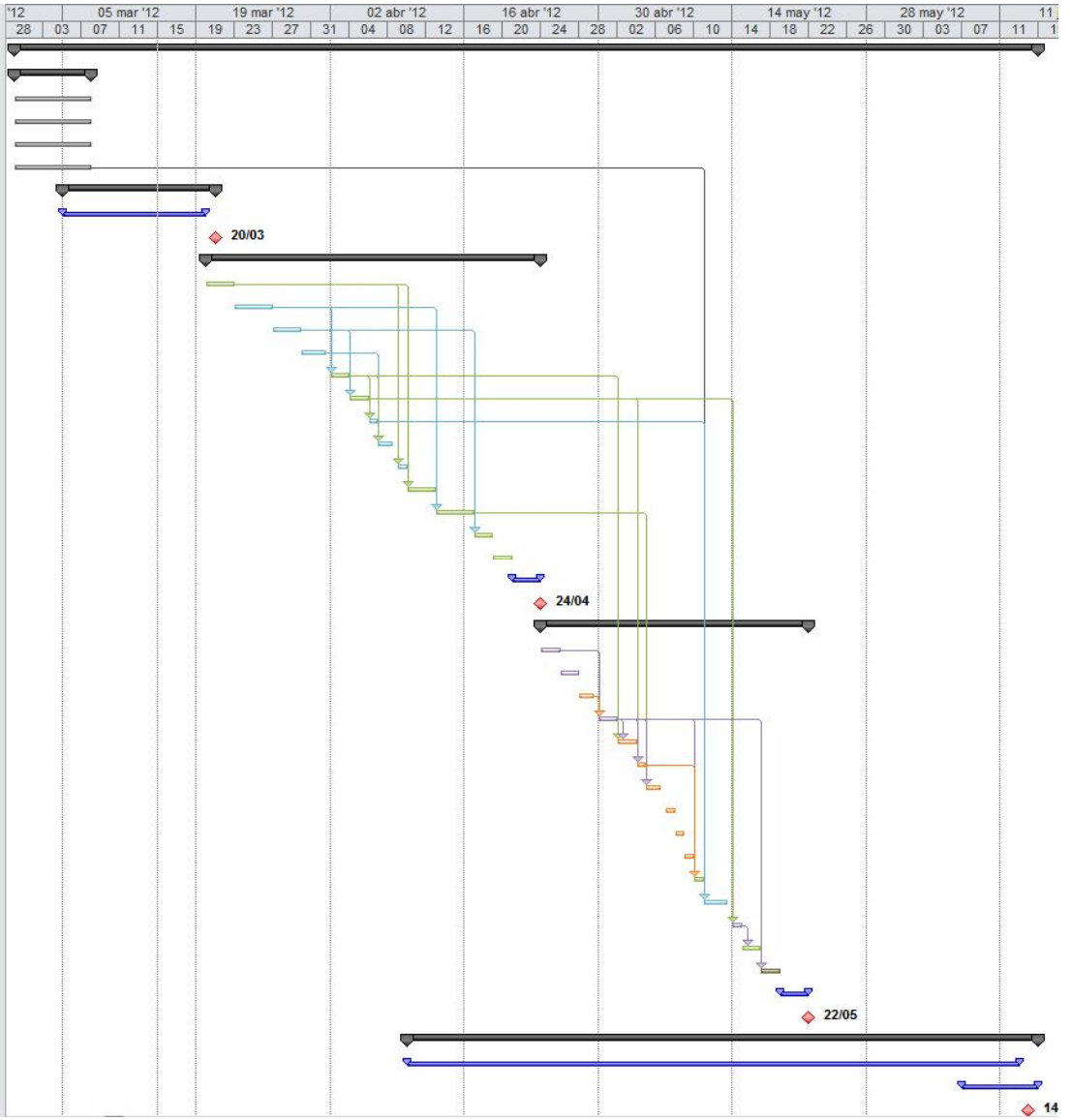

Ilustración 3 – Planificación. Diagrama Gantt

1.6. Recursos empleados

En este apartado enumeramos los recursos hardware y software empleados durante el proyecto.

a) Recursos hardware

- **Ordenador portátil Dell Latitude D620**

  Equipado con un procesador Core Duo con arquitectura de 32 bits y 4 GB de RAM, con sistema operativo dual Windows XP SP2 y distribución linux OSIAN; se destina en exclusiva a la realización del proyecto.

Ilustración 4 – Ordenador portátil modelo Dell Latitude D620

- **Dos Motes**

  Modelo ***COU 1_2 24 A2***.

  Incluyen el siguiente hardware:

  - Microcontrolador **Atmel 1281** del fabricante Atmel.
  - Sensor de luz: **PDV-P9003-1** del fabricante Advanced Photonix.
  - Sensor de temperatura: Termistor lineal activo de bajo consumo de la familia MCP970xA del fabricante Microchip.
  - Sensor de efecto Hall: modelo **BU52011HFV** del fabricante ROHM Semiconductor.
  - Transceptor de radio: modelo **AT86RF230** que opera en la banda de 2,4GHz, del fabricante Atmel.

Ilustración 5 – Mote COU 1_2 24 A2

  Uno de los motes se usará como coordinador de la red conectado por el puerto USB al PC de control y el otro se usará de sensor inalámbrico

- **Hardware adicional misceláneo**

  - Cables USB
  - Concentrador de puertos USB
  - Baterías recargables AA de 1,2v
  - Cargador de baterías

b) Recursos software

- Distribución linux OSIAN IPV6 basada en Ubuntu 10.04.
- TinyOS 2.0.1
- Java arquitectura 32 bits versión 1.7.0-b147 para linux.
- meshprog 0.1.2
- redirector de puertos remserial (Ver anexos).
- Editor VI
- Lazarus 0.9.30.4 arquitecturas 32 y 64 bits para Windows que incluye FPC v2.6.0
- Microsoft Office versiones 2003, 2007 y 2010.
- Microsoft Project 2007.
- Microsoft Visio 2007.
- MagicDraw UML

1.7. Productos obtenidos

Tras el desarrollo del proyecto hemos como resultado unos productos software y se han usado otros desarrollados por terceros:

a) Aplicación mote para el sensor inalámbrico

Aplicación **NodeStationAppC** desarrollada mediante TinyOS.

Realiza las medidas de batería, temperatura, controla la pulsación del botón y el sensor de efecto Hall para la activación del mote. Efectúa la transmisión inalámbrica de las medidas obtenidas del ADC y también de las alarmas detectadas.

b) Aplicación mote para el coordinador de red

Aplicación **BaseStationAppC** desarrollada mediante TinyOS.

Realiza todas las funciones enumeradas para el apartado del sensor inalámbrico, pero el envío lo realiza por medio del puerto serie. Adicionalmente actúa de puente entre la aplicación PC y la red de sensores proporcionando un interfaz de comandos y eventos creados al efecto basados en cadenas de texto ASCII.

c) Aplicación de control para PC

Aplicación para PC **MoteCtrl**, desarrollada mediante el entorno de desarrollo multiplataforma Lazarus.

Controla, configura, y gestiona toda la red y alarmas. Recibe del interfaz del sofware BaseStationAppC las notificaciones de alarmas de la red, la unión de nodos a la red de sensores, las notificaciones de reinicio del nodo controlador de red, etc., y muestra en el interfaz gráfico el estado de la red.

Nota: Independientemente de que se hayan proporcionado ejecutables para Windows (versiones 32 y 64 bits), Lazarus es multiplataforma; permite generar ejecutables para Linux, Windows y MacOSX entre otros. Mi intento de generar una aplicación para Linux no funcionó porque el componente que realiza las comunicaciones TCP/IP hacía un uso muy elevado del procesador.

1.8. Descripción de otros capítulos de la memoria.

En los siguientes apartados describiremos los diferentes aspectos del proyecto.

Apartado 2. ANTECEDENTES

Explicaremos cual es la tecnología en vigor para el campo de aplicación del proyecto, qué se puede encontrar en el mercado en la actualidad y el sector al que está dirigido este proyecto.

Apartado 3. DESCRIPCIÓN FUNCIONAL

Describiremos el diseño de nuestro proyecto y como lo hemos realizado.

Apartado 4. DESCRIPCIÓN DETALLADA

Explicaremos detalladamente el funcionamiento de los diferentes sistemas y los detalles de los módulos

Apartado 5. VIABILIDAD TÉCNICA

Se analizará y evaluará el producto obtenido tras la realización del proyecto. Que queda pendiente para que este sea viable.

Apartado 6. VALORACIÓN ECONÓMICA

Se realizará una valoración de los costes de producción.

Apartado 7. CONCLUSIONES

En este apartado, valoraremos la consecución de objetivos fijados al inicio del proyecto, si queda parte por realizar y, en ese caso, porque no se ha cumplido con los objetivos.

Apartado 8. GLOSARIO

Contiene el glosario de términos usados en el proyecto.

Apartado 9. BIBLIOGRAFÍA

Fuentes de información consultadas para la realización del proyecto.

Apartado 10. ANEXOS

Contiene el manual de usuario, las instrucciones de compilación del software y por último las licencias identificadas de las se han hecho uso durante la realización del proyecto.

2. Antecedentes

2.1. Estado del arte

En el área de los sistemas empotrados/embebidos podemos encontrar multitud de opciones.

Estas opciones son muy amplias. La selección de los diferentes parámetros de cada uno de los componentes, hace del diseño de una arquitectura basada en microcontrolador, un 'estado del arte' debido al 'diseño a medida' que se realiza.

Empezaremos enumerando los principales parámetros para la elección del microcontrolador, selección de tipo de radio inalámbrica, pila de comunicaciones usada sobre la radio física, entorno operativo, etc.

Continuaremos describiendo los WSN (*wireless sensor network*, conocidos en español con el nombre de *motes)*.

• Microcontrolador

Los parámetros básicos para seleccionar una microcontrolador (**C**) son:

- Familia del microcontrolador (PIC, Arm, Xscale, AVR, MCS-31/51, MSP430)
- Rango de temperaturas de funcionamiento
- Frecuencia de reloj (KHz-GHz)
- Tensión de alimentación.
- Tipo de arquitectura del microprocesador (CISC/RISC).
- Tipo de arquitectura de acceso a memoria (Von Neumann/Harvard).
- Cantidad de memoria RAM/ROM/EPROM/EEPROM/FLASH
- Anchura en bits de la unidad aritmético-lógica: 8, 16 o 32 bits.
- Numero de pines del encapsulado.
- Número de entradas/salida digitales y analógicas
- Si contiene conversores analógico-digitales o digital-analógico, controladores especializados PWM, comunicaciones serie RS-232, RS-485, SPI, I2C, Ethernet, USB, comunicaciones radio 802.11, 802.15.4…
- Ratio MIPS/W

Con toda seguridad una vez fijados los requerimientos del proyecto podremos elegir un micro-controlador que se ajuste a las necesidades.

• Radio inalámbrica

Las posibilidades de conexión de radios inalámbricas a micro-controladores han ido evolucionando con el tiempo. En el inicio, se usaban módulos analógicos en la banda de 433Mhz, convirtiéndose con el tiempo en módulos digitales parecidos a los registros de desplazamiento, dónde el control se realizaba de forma similar a un puerto serie.

Estos módulos fueron cambiando y para aumentar las ofertas de frecuencia, así como los tipos de conexiones: bandas de 868Mhz, 915Mhz o de 2,4Ghz; conexiones de alto nivel mediante SPI, bus paralelo, etc.

Otro de los parámetros que afectan a este tipo de dispositivos es el tipo de modulación que soportan, así aunque los estándares 802.11 y 802.15.4 pueden operar ambos en la frecuencia de 2,4Ghz, no son interoperables entre sí.

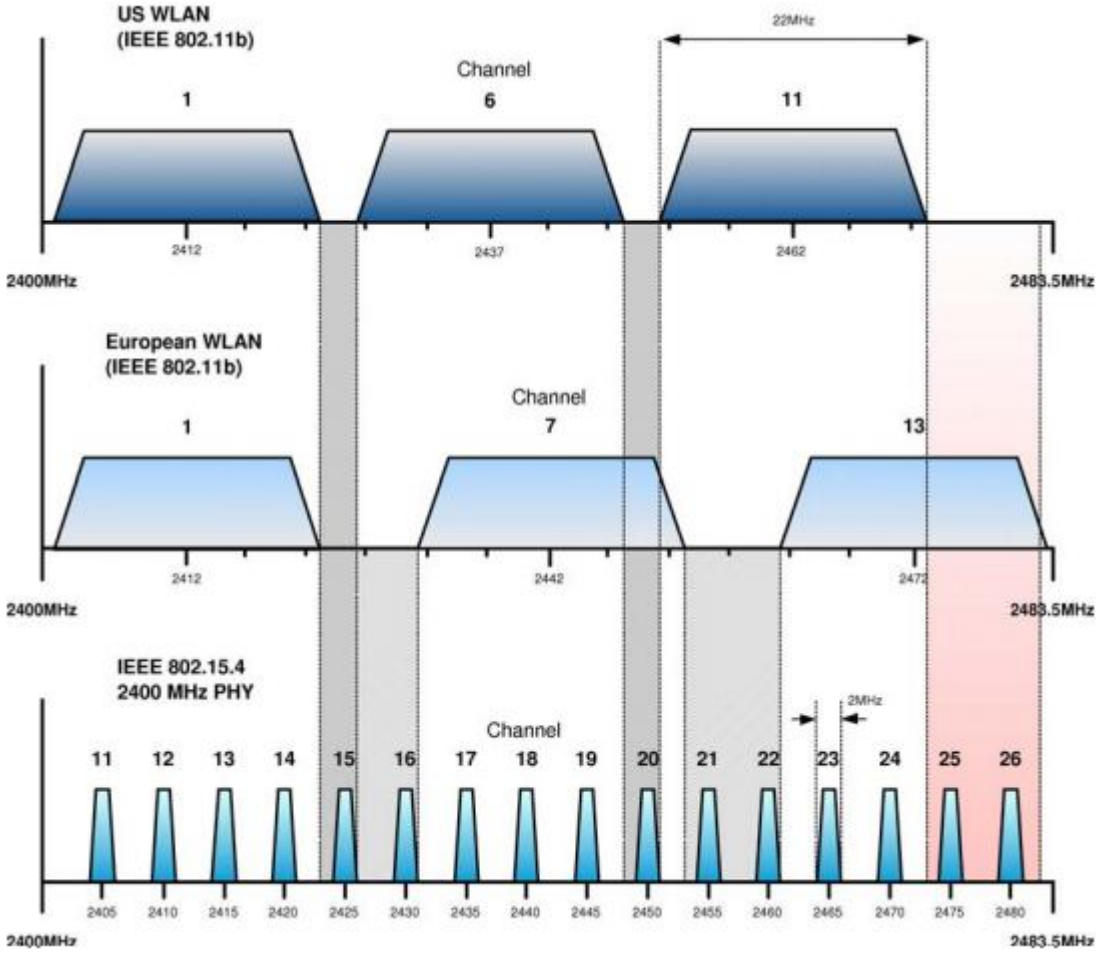

Ilustración 6 – Asignación de canales en la banda de 2,4GHz según IEEE 802.11b y 802.15.4

Además de la interoperabilidad entre las diferentes normas, debemos tener en cuenta que la banda de 2,4GHz es una banda regulada para emisión no comercial, también denominadas bandas ISM (Industrial, científica y médica). Para emitir en ese rango de frecuencias no se

requiere la solicitud de licencia y cualquier aplicación puede usarla. A día de hoy, las redes inalámbricas del hogar, dispositivos bluetooth, teléfonos fijos inalámbricos con tecnología DECT, dispositivos que usan el estándar 802.15.4 y cualquier otro dispositivo de consumo es libre de emitir en esa banda; el resultado de todo esto es que muchas veces nos encontramos esa banda totalmente saturada, con ratios de error muy altos y relación señal/ruido de mala calidad.

Los módulos de radio más habituales para conexiones inalámbricas en micro-controladores son:

- Chipcon CC1000
- Chipcon CC1020
- Chipcon CC2420
- Xemics XE1205
- Atmel AT86RF212 y AT86RF23x

La clasificación general de redes atendiendo a los requerimientos de ancho de banda y alcance se expone en el siguiente gráfico:

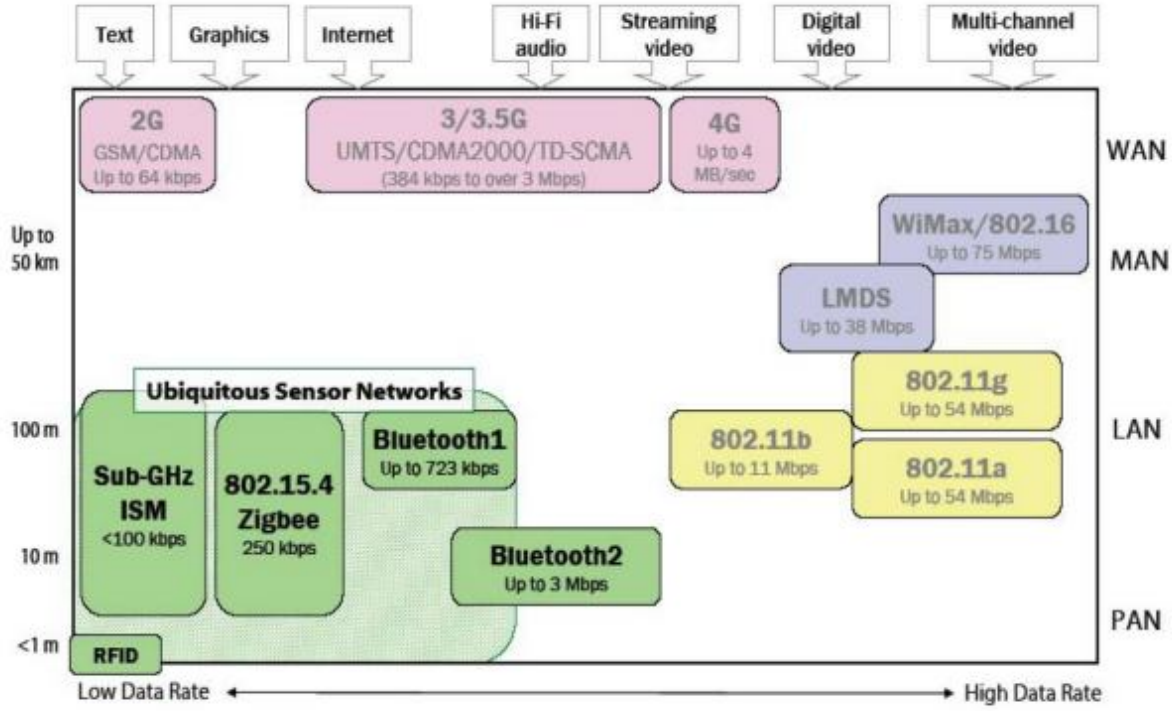

Ilustración 7 – Clasificación de redes inalámbricas: PAN, LAN, MAN y WAN

• Pila de comunicaciones

Al hablar de la pila de comunicaciones, nos referimos al protocolo en referencia a los niveles OSI.

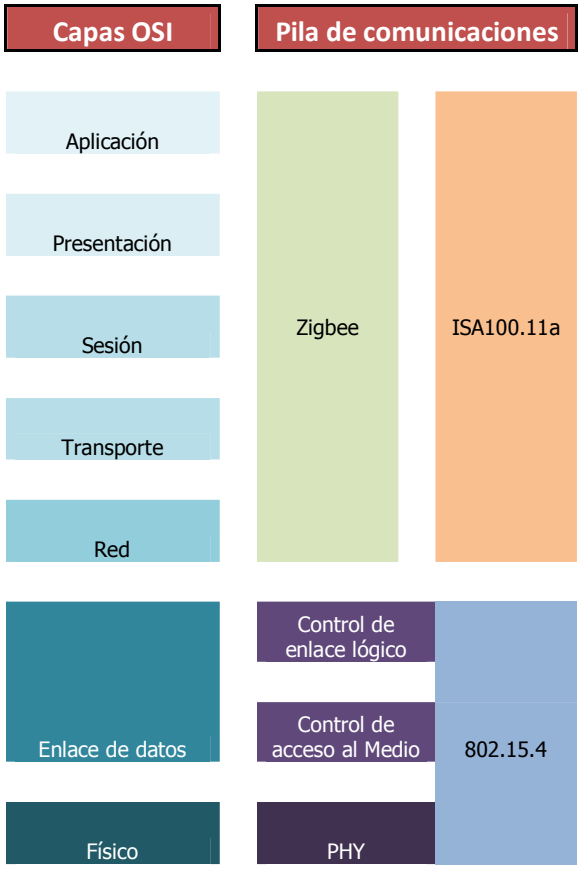

Ilustración 8 – Capas OSI y correspondencia con pilas de comunicaciones

Según las comunicaciones proporcionadas por los entornos de desarrollo, o directamente mediante las librerías de funciones provistas por los fabricantes, el control de las transferencias de datos se realizará por el programa o por las librerías y requerirán más o menos esfuerzo de programación.

• Entornos operativos o de desarrollo

Los principales entornos de desarrollo disponibles para la programación de los micro-controladores son:

- Los proporcionados por el fabricante del micro-controlador, generalmente se trata de ensambladores, intérpretes de lenguaje BASIC, compiladores de C con extensiones no estándar y librerías estáticas enlazables en tiempo de compilación al código de usuario.

- TinyOS
  Entorno y sistema operativo de código abierto para redes de sensores inalámbricos. Proporciona soporte a gran cantidad de plataformas, muy heterogéneas, basado en un lenguaje C ampliado llamado nesC. Soporte nativo para comunicaciones 802.15.4.

- Contiki OS

  Similares capacidades a TinyOS pero con soporte nativo para pila de comunicaciones IPv4/IPv6

• WSN o motes

La abreviatura WSN puede aplicarse a dos términos: *Wireless Sensor Network* o *Wireless Sensor Node*. En este caso nos referimos al segundo término, traducido al español serían Nodos Sensores Inalámbricos, también conocidos como ***motes***.

Un mote es la combinación de un micro-controlador con una radio inalámbrica añadiendo hardware variado que le permitirá tomar medidas de diferentes parámetros ambientales dependiendo de los sensores físicos que el mote incorpore en su diseño.

La denominación mote es genérica y no especifica ni el microcontrolador, ni la radio usada ni tampoco los sensores que incorpora. Normalmente se tratan de desarrollos hardware realizados para una función específica o bien prototipos que permitirán hacer un desarrollo para luego aplicarlo a líneas de producción. Normalmente han sido desarrollados bien por los fabricantes hardware del micro-controlador (caso del mote Raven), particulares o universidades. Se detallan a continuación las características de los motes más conocidos obtenidos en su mayoría de la página de soporte TinyOS.

Sensor Node Name	Microcontroller	Tranceiver	Program+ Data Memory	External Memory	Programming
COU24	ATmega 1281	Atmel AT86RF230 802.15.4/ZigBee compliant radio	8K RAM	128K FLASH ROM, 4K EEPROM	nesC
Arago SystemsWiSMote Dev	MSP430F5437	CC2520	RAM : 16 Kbytes Flash : 256 Kbytes	up to 8 Mbits	C
Arago SystemsWiSMote 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	
Eqs [1]	ARM Cortex M3	CC2520, Mitsumi's class 2 Bluetooth module		2 Gbit	
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 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash	
Mica	ATmega 1034 MHz 8-bit	RFM TR1000 radio 50 kbit/s	128+4K RAM	512K Flash	nesC Programming

	CPU				
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 (seemc1322x.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 IEEE 802.15.4	4k RAM	128k Flash	
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
Zolertia Z1	TI MSP430	CC2420	8K RAM	92K Flash	C, nesC

Ilustración 9 – Listado de motes conocidos y soportados pot TinyOS

2.2. Estudio de mercado

Existen productos similares, y según normativa Europea, vienen regulados según la norma UNE/EN54-25 "***Fire detection and fire alarm systems. Components using radio links***", publicada en Marzo de 2008. Esta norma, es específica para dispositivos de alarma de incendios inalámbricos y no regula la banda de frecuencia, de hecho, se puede aplicar la regulación que establece para Europa la ETSI. Los productos de este tipo deben obtener la marca CE y el sello correspondiente claramente identificado en cada uno de los dispositivos:

Ilustración 10 – Sello del cumplimiento UNE/EN54-25

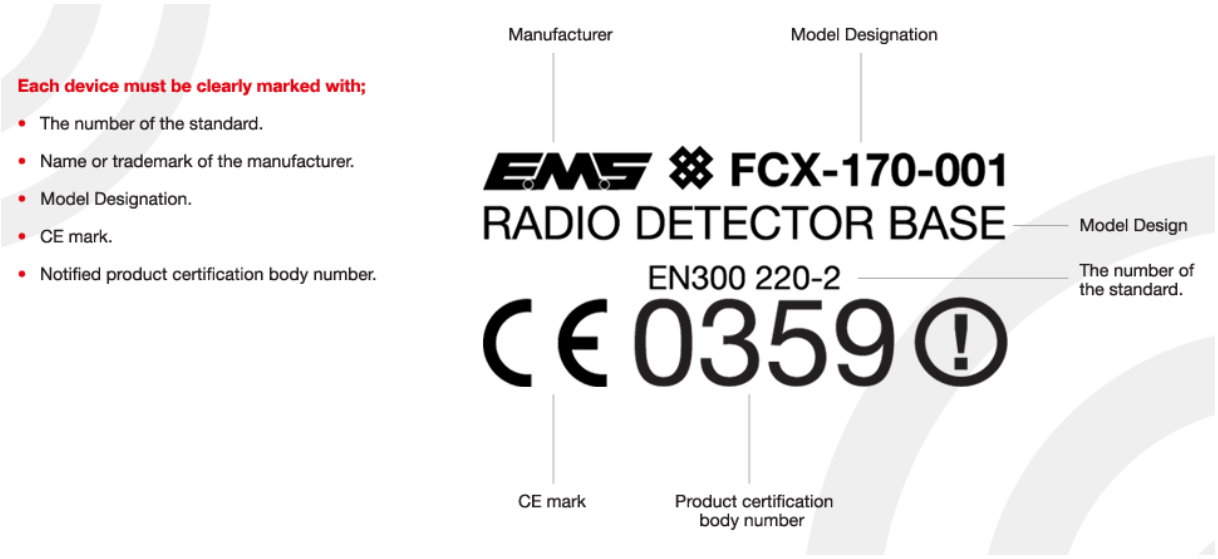

Ilustración 11 – Detalle de la certificación según norma CE para UNE/EN54

Descripción	Enlace
Normativa Europea UNE/EN-54	http://en.wikipedia.org/wiki/EN_54
Artículo sobre UNE/EN-54	http://www.notifier.es/desdocumentacion/notifier/noticias/articulos/Normativa.pdf
Información divulgativa UNE/EN-54 25	http://www.en54-25.com/en54-part-25/standards.aspx
Documentación AENOR	http://www.aenor.es/aenor/normas/normas/fichanorma.asp?tipo=N&codigo=N0046608&PDF=Si
Vendedores de productos	http://www.sourceen54.eu/ Productos Bosh Norma EN54

Ilustración 12 – Tabla de enlaces para productos similares al desarrollado

3. Descripción funcional

El sistema de alarma de incendios puede ser dividido en subsistemas independientes, dónde cada capa ejecuta una función y el usuario interactúa mediante la aplicación que se ejecuta en el PC con las configuraciones de la red, eventos y datos recibidos.

Estas capas de aplicación hacen uso de los diferentes componentes, métodos y funciones que se han desarrollado, y de hecho, comparten componentes.

Desde el punto de vista de los datos transferidos los clasificaremos en tres subsistemas de alto nivel:

3.1. Visión global

a) Subsistema de alarmas

Es el subsistema encargado de enviar, reenviar y confirmar las alarmas recibidas, tanto de temperatura, manuales como de batería baja.

Activa la confirmación de recepción de alarmas de extremo a extremo, asegurando la recepción de alarmas al usuario, que son reenviadas desde el nodo origen mientras no se haya recibido la confirmación de esa alarma concreta.

Dependiendo de cada uno de los subestados internos de este subsistema se actualizan las notificaciones visuales y sonoras.

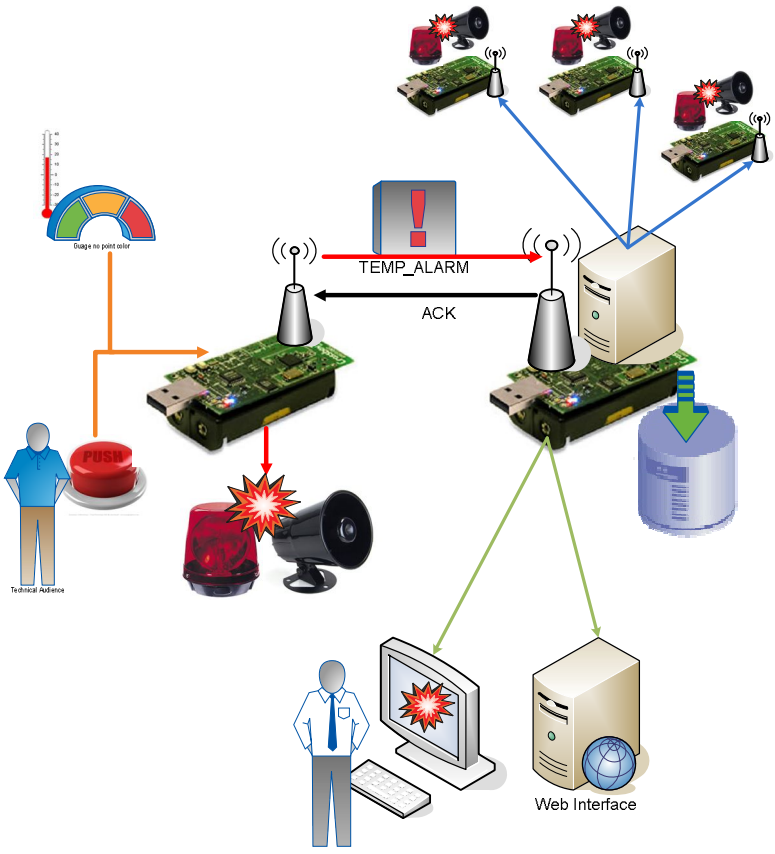

Ilustración 13 – Subsistema de alarmas

b) Subsistema de monitorización de nivel de baterías

Es el subsistema encargado de gestionar la recepción de datos relativos a la carga de las baterías si no constituyen alarma.

Este sistema no garantiza la entrega de los datos. Tampoco se gestionan confirmaciones de recepción.

El envío se realiza periódicamente; si no se reciben datos de un nodo concreto durante un tiempo dado, la aplicación de gestión procede a la eliminación del nodo del interfaz de usuario.

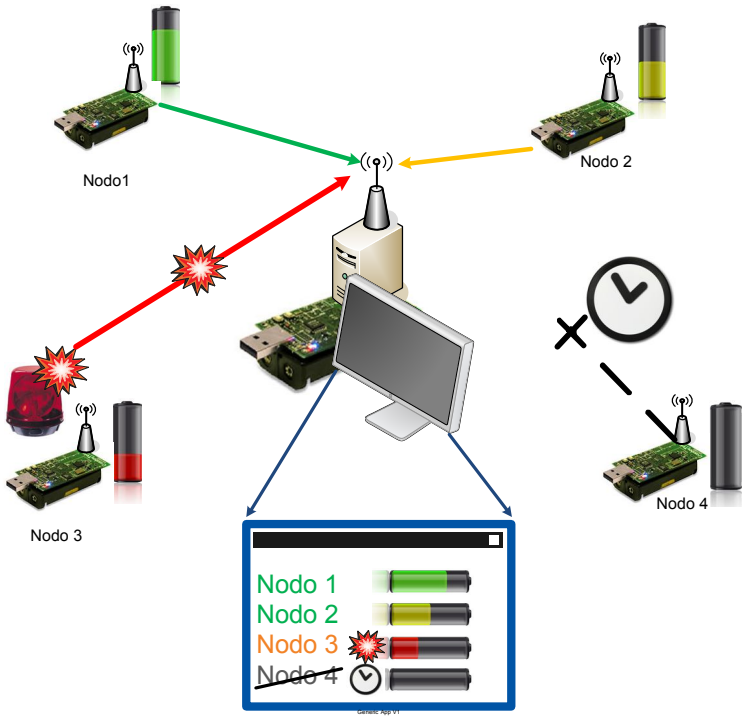

Ilustración 14 – Subsistema de monitorización de baterías

c) Subsistema de monitorización de cobertura

Subsistema encargado de procesar la calidad de la señal con que se reciben el resto datos.

Aunque se ha separado a nivel funcional, es un sistema que forma parte de los dos anteriores; cada uno de los mensajes de alarma o de datos recibidos genera meta-información en la estación coordinadora que indica la calidad de señal.

Esta meta-información se emplea para la toma de decisión de cambio de canal usado para la red de sensores.

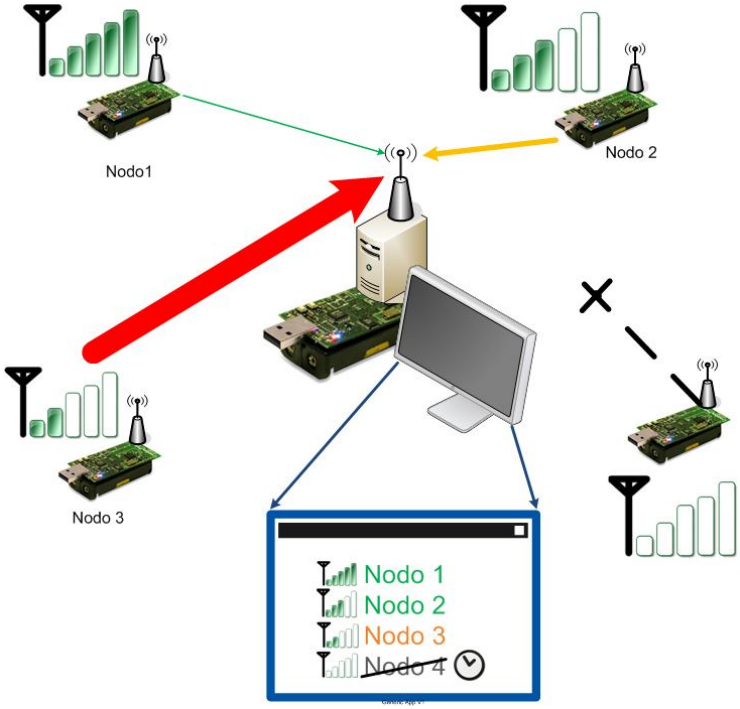

Ilustración 15 – Subsistema de monitorización de cobertura

d) Comunicaciones de red

Desde el punto de vista de las comunicaciones de red, incluimos un diagrama de interacciones entre los diferentes sistemas relacionados:

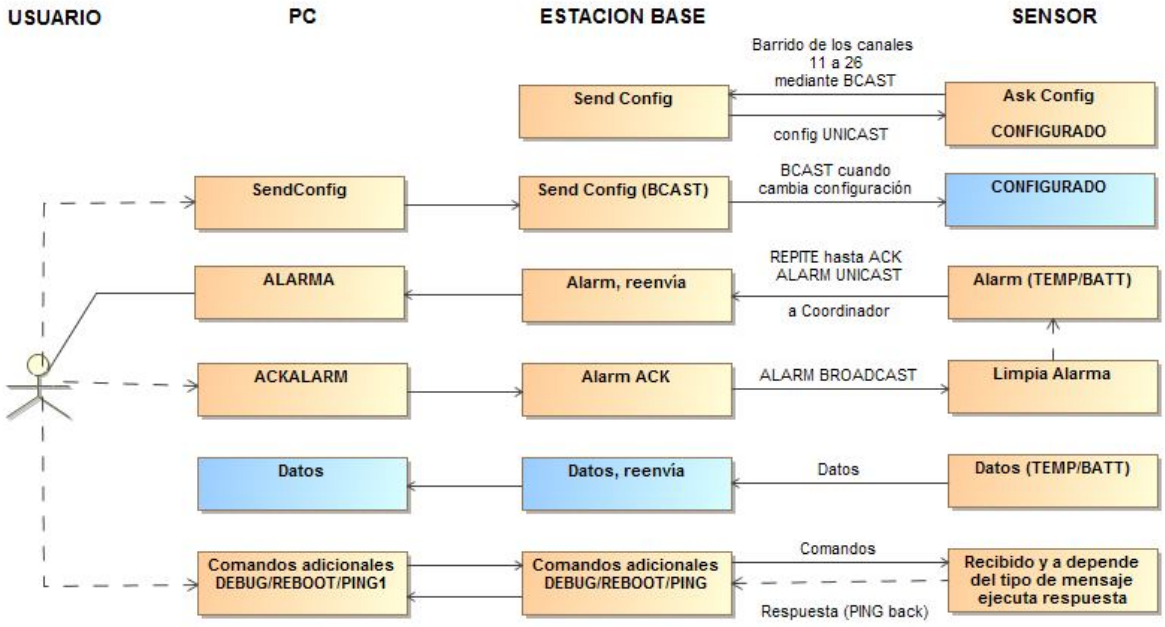

Ilustración 16 – Comunicaciones de red

Las estructuras de datos que se intercambian en las comunicaciones de red son las que a continuación:

CSB_CONFIG **AM_CSB_CONFIG**

Tipo de dato	nx_uint16_t	nx_uint16_t	nx_uint16_t	nx_uint16_t	nx_uint16_t	nx_uint16_t
longitud	16 bits	16 bits	16 bits	16 bits	16 bits	16 bits
nombre	channel	nodeid	temp_alarm_s	temp_alarm_threshold	batt_alarm_s	batt_alarm_threshold

SCX_ASK_CONFIG **AM_SCX_ASK_CONFIG**

Tipo de dato	nx_uint16_t	nx_uint16_t
longitud	16 bits	16 bits
nombre	dst_nodeid	orig_nodeid

SCX_ALARM **AM_SCX_ALARM**

Tipo de dato	nx_bool	nx_uint16_t	nx_uint8_t	nx_uint16_t	nx_uint16_t
longitud	8 bits	16 bits	8 bits	16 bits	16 bits
nombre	is_alarm	nodeid	alarm_kind	*param*	counter
			ALRM_TEMP	*temp_level*	
			ALRM_BATT	*batt_level*	
			DATA_DEBUG	*temp_debug_s*	
			*	*dummy*	

Ilustración 17 – Estructuras de datos intercambiadas en los mensajes de red

3.2. PC

a) Diagrama de bloques de la aplicación

Una de las decisiones tomadas explicaba que el desarrollo sería modular para poder reutilizar el software; así, cada uno de los componentes creados, se ha diseñado para ser independiente de cualquier otro y evitar interacciones indeseadas

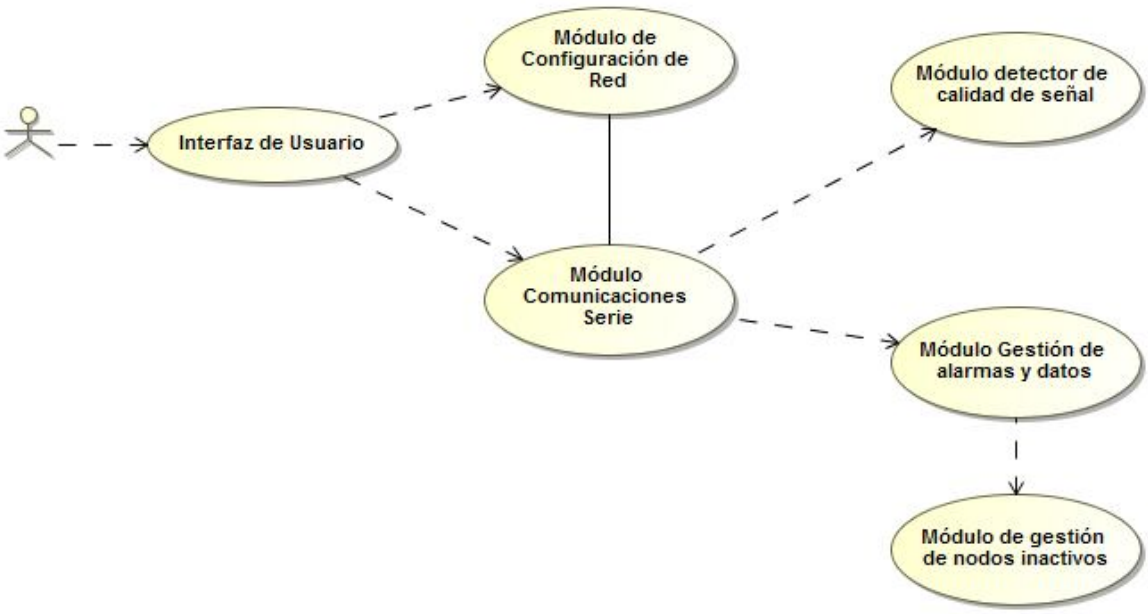

Ilustración 18 – Diagrama de bloques de la aplicación PC

3.3. Motes

Una de las decisiones tomadas explicaba que el desarrollo sería modular para poder reutilizar el software; así, cada uno de los componentes creados, se ha diseñado para ser independiente de cualquier otro y evitar interacciones indeseadas.

Los programas finales que se ejecutan en los motes hacen uso común de cada uno de estos componentes software habilitando sólo las funciones necesarias para su correcto funcionamiento.

• BaseStationAppC

El programa controlador de red hace uso de los componentes que se especifican en el siguiente diagrama:

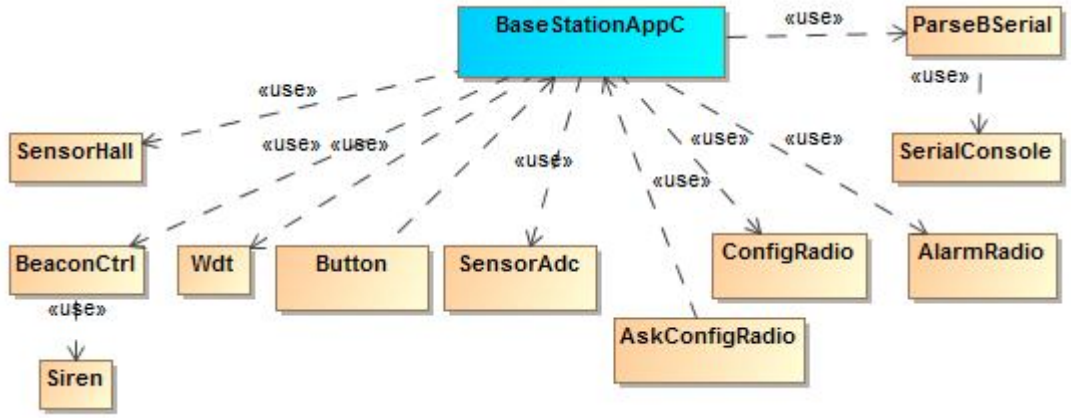

Ilustración 19 – BaseStationAppC: uso de componentes

• SensorAppC

El nodo sensor utiliza los siguientes componentes:

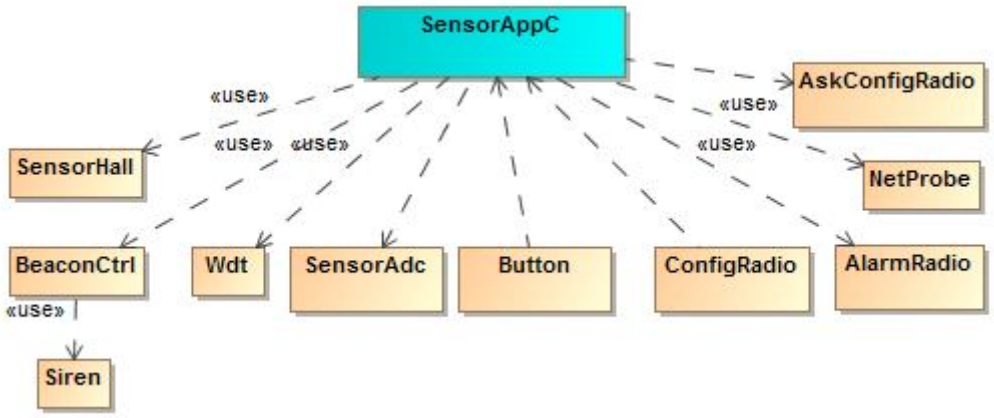

Ilustración 20 – SensorAppC: uso de componentes

- **Componentes**

A continuación se enumeran los diferentes componentes:

- ***AlarmRadio***

Componente diseñado para el envío de alarmas y datos de medida entre los motes sensores y el controlador de red.

- ***AskConfigRadio***

Componente que gestiona la petición de configuración de los motes sensores hacia el mote controlador de la red tras su inicialización.

- ***Button***

Componente que gestiona la pulsación del botón de usuario del mote.

- ***BeaconCtrl***

Módulo encargado de la señalización visual y sonora de la baliza según el estado interno del mote.

- ***ConfigRadio***

Componente que gestiona el envío de la configuración desde el nodo controlador hacia los sensores.

- ***ParseBSerial***

Sólo usado en nodo Controlador. Este componente realiza la conversión entre mensajes de red inalámbricos y serie.

- ***Siren***

Módulo opcional. Sólo se debería utilizar cuando se disponga de hardware adicional conectado al bus de expansión del mote.

- ***NetProbe***

Sólo usado en nodo Sensor. Componente encargado de realizar la autoconfiguración de red del mote sensor. Realiza el barrido de canales, busca en cada uno de ellos al nodo controlador y cuando lo localiza solicita la configuración activa en la red de sensores.

- ***SensorAdc***

Módulo que realiza la lectura de los valores de los sensores, notifica cuando la lectura se ha realizado y proporciona el valor leído.

- ***SensorHall***

Módulo que realiza la gestión de activar y desactivar el sistema de alarmas.

- ***SerialConsole***

Sólo usado en nodo Controlador. Gestión de mensajes recibidos por el puerto serie del mote. Transforma cadenas de texto ASCII a acciones específicas del mote controlador.

- ***Wdt***

Componente cuya función es habilitar el reinicio automático del mote cuando se bloquea y deja de funcionar correctamente.

4. Descripción detallada

4.1. Mote: BaseStationAppC

BaseStationAppC hace uso de los componentes enumerados en el apartado anterior, y funciona de manera resumida como se describe en el siguiente diagrama:

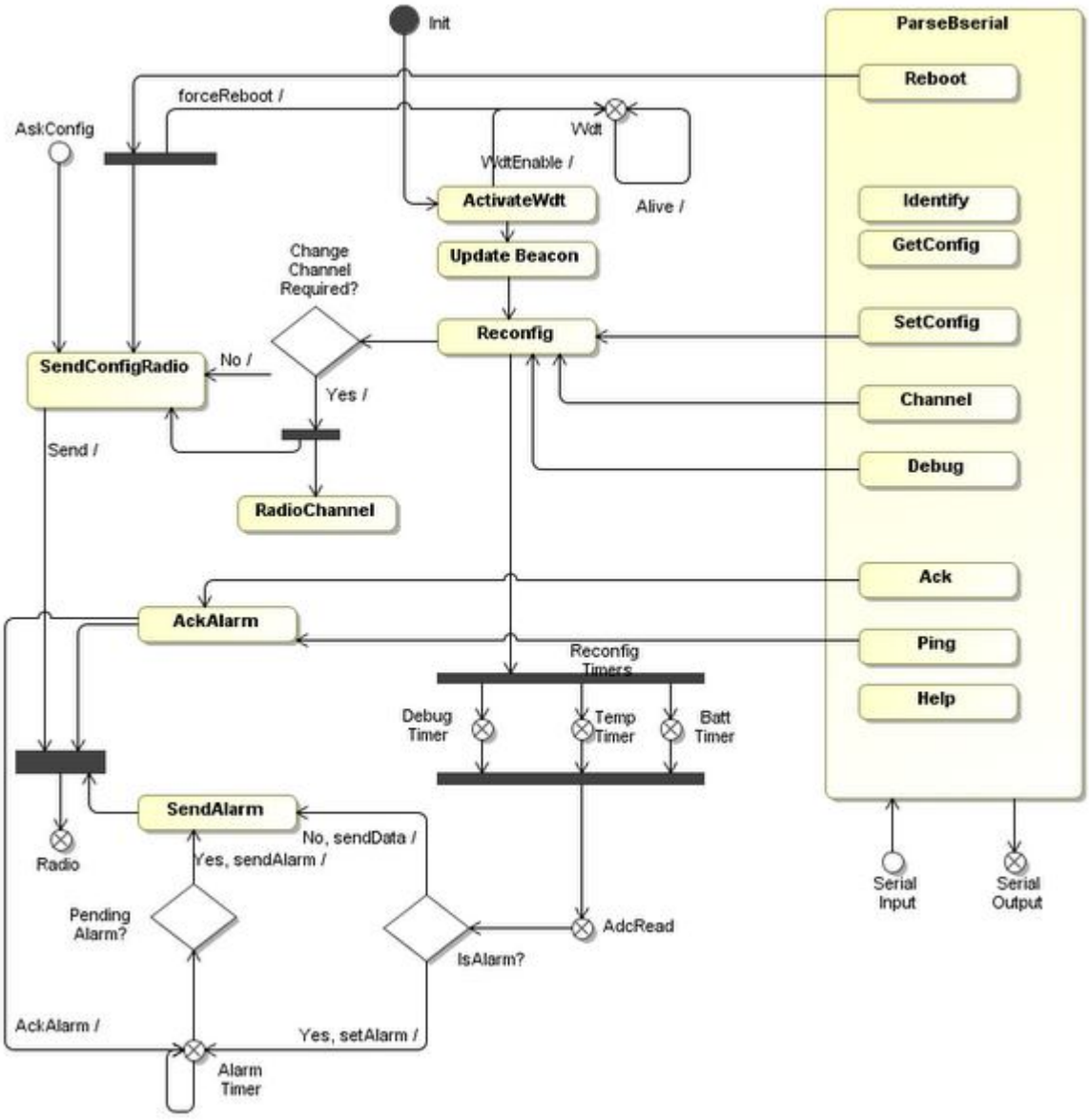

Ilustración 21 – BaseStationAppC: funcionamiento

4.2. Mote: SensorAppC

SensorAppC hace uso de los componentes enumerados en el apartado anterior, y su diagrama de funcionamiento es el que se detalla a continuación:

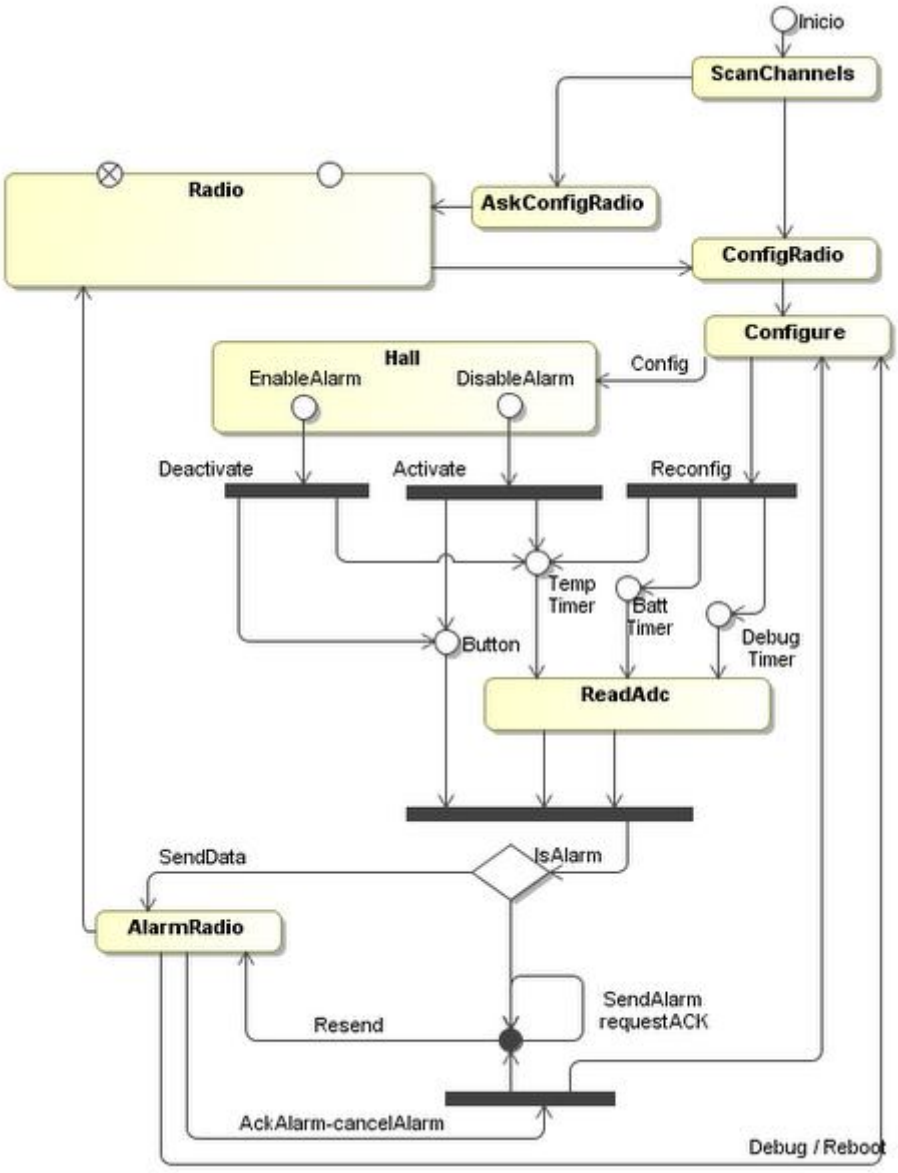

Ilustración 22 – SensorAppC: funcionamiento

4.3. Mote: Componentes

En este apartado detallaremos los diferentes módulos, comandos que proporcionan y eventos generados. Las aplicaciones BaseStationAppC y SensorAppC hacen uso de estos componentes llamando a los comandos y métodos públicos del componente o capturando los eventos.

AlarmRadio

interface AlarmRadio

Commands
command SCX_ALARM *getData()
command int16_t getRssi()
command void sendData(uint16_t dstNode, SCX_ALARM data)

Events
event void DataReceived()
event void DataSent()
event void PktAcked()

AskConfigRadio

interface AskConfigRadio

Commands
command SCX_ASK_CONFIG *getData()
command void sendData(uint16_t dstNode, SCX_ASK_CONFIG data)

Events
event void DataReceived()
event void DataSent()

Button

interface Button

Events
event void fired()

BeaconCtrl

interface BeaconCtrl

Commands
command void askControl()
command void get_alarm_status()

command void get_battery_status()
command void get_network_status()
command void notify_booted()
command void releaseControl()
command void set_alarm_status()
command void set_battery_status()
command void set_status(beacon_state b_state)
command void still_alive()

ConfigRadio

interface ConfigRadio

Commands

command CSB_CONFIG *getData()
command void sendData(uint16_t dstNode, CSB_CONFIG data)

Events

event void DataReceived()
event void DataSent()

ParseBSerial

interface ParseBSerial

Commands

command SCX_ALARM *getACK(char *data)
command uint8_t getBoolean(char *data)
command uint8_t getByte(char *data)
command uint8_t getBytedecimal(char *data)
command CSB_CONFIG *getConfig(char *data)
command SCX_ALARM *getDebug(char *data)
command uint16_t getWord(char *data)

Siren

interface Siren

Commands

command void off()
command void on()
command void toggle()

NetProbe

interface NetProbe

Commands
command bool ConfigReceived()
command uint8_t getChannel()
command CSB_CONFIG *getConfig()

Events
event void channelChanged()
event void Scanfinished()

SensorAdc

interface SensorAdc

Commands
command error_t askRead(adc_read_t read_what, read_type_t kind)
command uint16_t getTempLastValue()
command uint16_t getValue()

Events
event void BattReaded()
event void DBattReaded()
event void DPhotoReaded()
event void DTempReaded()
event void PhotoReaded()
event void TempReaded()

SensorHall

interface SensorHall

Events
event void Notify()<hall_state_t>;

SerialConsole

interface SerialConsole

Events
event void Notify<console_command_t>;

Wdt

interface Wdt

Commands
command void enable()
command void force_sensor_delayed_reboot(uint16_t ms)
command void force_sensor_reboot()

5. Viabilidad técnica

El producto tal y como se ha entregado no se debería comercializar, aunque corrigiendo las carencias que se expondrán a continuación, sería viable.

- Cumple los requisitos necesarios para realizar la implantación en cuanto a cantidad de nodos soportados
- No cumple un requisito no necesario para que el producto sea viable técnicamente: evitar colisiones de sensores con el mismo identificador en la red. Respecto a este punto, convendría proporcionar un modo de autoconfiguración de identificador de nodo, y también sería recomendable poder configurar con metadatos añadidos al nodo la ubicación del nodo en la instalación, por ejemplo mediante coordenadas de columnas o por piso, oficina, etc.
- No cumple el requisito de cobertura de red: La topología diseñada se corresponde a una red en estrella donde la extensión de la instalación viene limitada por el alcance de la señal del nodo coordinador. Este punto puede ser solucionado instalando varios nodos coordinadores en diferentes ubicaciones; este punto fue decisivo para decidir no usar java y utilizar un redirector serie programado en C; usando un router wifi Ethernet que disponga de puerto USB sería posible compilar el programa que hace de puente entre el mote coordinador y el puerto TCP. Generalmente los dispositivos routers wifi no soportan la instalación de java pero si permiten compilar e instalar programas adicionales (ver dd-wrt).
- Es un producto que no está terminado. Falta por añadir funcionalidad final a la aplicación de control ejecutada en el PC.

El punto más importante para poder comercializar el producto sería obtener la certificación CE según la norma UNE/EN-54 25; en España, esa certificación podría ser extendida por AENOR.

6. Valoración económica

No puedo realizar una valoración final del proyecto comercial.

Puedo estimar el coste de cada uno de los componentes actuales, horas dedicadas por persona y rol desarrollado y un precio estimado unitario, pero para llevar el proyecto a una fase de producción deberíamos tener en otros factores:

Los motes con los que se ha desarrollado el proyecto son prototipos. En fase productiva se eliminaría el puerto serie/USB, habría que proporcionar posibilidad de conectar leds externos, proporcionar una caja adecuada para la instalación del producto que además cumpla normativas de estanqueidad de las instalaciones, etc.

Todo ello requeriría un nuevo proyecto de diseño de producción en el que se tengan en cuenta estos aspecto así como los costes de lanzamiento, publicidad y de comercialización.

Una valoración del coste estimado asociado al proceso de desarrollo sería:

- Un ordenador PC completo: entre 400€ y 1.000€
- Cada uno de los motes: entre 40€ y 100€
- Unidades de diseño/programación(h): 400h
- Precio unitario programación: 40€/h
- Precio unitario análisis y diseño: 60€/h

En los dos últimos elementos especifico precio y no costes.

7. Conclusiones

7.1. Conclusiones

Este proyecto me ha permitido retomar el contacto con el diseño de aplicaciones con micro-controladores.

Me sorprendió gratamente que aquellos micro-controladores de la familia MCS-51 con los que comencé se han convertido en elementos mucho más potentes, con herramientas y hardware que permiten extender su uso y en los que no hay que tener en cuenta cada ciclo usado de reloj.

Esta nueva toma de contacto ha exigido bastante esfuerzo. TinyOS facilita muchísimo las tareas de desarrollo y con un buen diseño convierte el código en una herramienta realmente modular, dónde cada capa hace uso de las inferiores. Como inconveniente he percibido que es difícil de aprender, la curva de aprendizaje es muy alta al principio ya que introduce una nueva metodología de desarrollo orientada a comandos, señales y evento, debido tener bastante en cuenta la concurrencia e interacciones que puedan aparecer entre los diferentes componentes.

He tenido que tomar la decisión de renunciar a ciertos objetivos como el control de cobertura de señal (está implementada sólo en los motes pero no en la aplicación) y la creación de un interface web por mi continuada percepción de *falta de tiempo* e incumplimiento de los plazos auto establecidos.

Si los objetivos incompletos o no iniciados se completasen, podría convertirse en un producto viable comercialmente, destinado al menos al nicho de detección de incendios en la fase de construcción de edificios.

7.2. Propuesta de mejora

Las carencias principales detectadas han sido:

- Habilitar la autoconfiguración de los nodos para evitar colisiones de mensajes en caso de diferentes nodos operando con el mismo TOS_NODE_ID.
- Ampliación de la cobertura del nodo controlador para cubrir instalaciones más extensas, bien usando múltiples nodos controladores o utilizando una arquitectura de red tipo ***mesh*** en lugar de una arquitectura en estrella.

7.3. Autoevaluación

Uno de los problemas continuos que he tenido a lo largo del proyecto ha sido la percepción de *falta de tiempo*. Habiendo realizado una planificación (en mi opinión) coherente con los plazos establecidos siempre han surgido problemas que me hacían incumplir los plazos.

Ha habido cinco 'incidentes' inesperados de este tipo:

- Interacción no esperada entre el sensor efecto Hall y el botón de disparo de alarmas: La activación de uno de ellos causaba la ejecución de las rutinas de ambas (primera fase de entrega de código).
- Corrupción del campo "nodo origen" en el paso de información de las funciones de la red: Hasta que no se rectificó el problema las comunicaciones por red no eran posibles (primera fase de entrega de código).
- Error de programación en el módulo ParseBSerial que provocaba el bloqueo del mote y posterior recuperación por el watchdog interno mediante un reinicio no esperado. Error de programación debido al uso de una estructura intermedia de tamaño insuficiente.
- Estimación incorrecta del tiempo necesario para desarrollar la documentación –memoria y presentación-.
- Coincidencia con una reasignación de nuevas tareas en mi trabajo a mediados del cuatrimestre que han exigido mayor dedicación temporal en detrimento de la dedicación esperada para el proyecto.

8. Glosario

802.11	Estándar *IEEE* que define el uso de la capa física y enlace de datos según la arquitectura OSI en una red inalámbrica WLAN
802.15.4	Estándar *IEEE* que define el uso de la capa física y acceso al medio según la arquitectura OSI en una red inalámbrica WPAN
ADC	Abreviatura de *Conversor Analógico Digital* en inglés.
Advanced Photonix	Fabricante de componentes electrónicos. www.advancedphotonix.com
Arm	Arquitectura de procesadores de 32 bits desarrollada por ARM Holdings
ASCII	Acrónimo de **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
ATmega1281	Micro-controlador de 8 bit del fabricante Atmel
Atmel	Fabricante de componentes electrónicos. www.atmel.com
AVR	Arquitectura de procesadores de 8 bits comercializada por Atmel
Basic	Acrónimo de **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode; lenguaje de programación diseñado en 1964.
C	Lenguaje de programación creado en 1972 por Dennis M. Ritchie
CISC/RISC	Arquitecturas de procesadores. Acrónimos de ***C**omplex **I**nstruction **S**et **C**omputer* y ***R**educed **I**nstruction **S**et **C**omputer* respectivamente.
Contiki OS	Sistema operativo de red diseñado para *Internet of Things*.o identificación unívoca de objetos, en este caso, dispositivos.
COU_1_2 24	Mote desarrollado por la *Universitat Oberta de Catalunya* bajo el programa APLICA2010
CP2102	Conversor USB / RS-232 en un único chip; fabricado por Silicon Labs
dd-wrt.org	Alternativa OpenSource a firmwares de enrutadores inalámbricos. www.dd-wrt.com
eCos	Sistema operativo de tiempo real open source para micro-controladores. ecos.sourceware.org
EEPROM, E2PROM	Acrónimo de ***E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory*
EPROM	Acrónimo de ***E**rasable **P**rogrammable **R**ead-**O**nly **M**emory*
Ethernet	Denominación del estándar IEEE 802.3
FLASH	Tipo de memoria derivada de la memoria EEPROM que permite la lectura o escritura por bloques.
GUI	*Acrónimo de **G**raphical **U**ser **I**nterface.* Interfaz gráfico de usuario.
I2C, I²C	Acrónimo de ***I**nter-**I**ntegrated Circuit*. Bus de comunicaciones serie.
IEEE	Acrónimo de ***I**nstitute of **E**lectrical and **E**lectronics **E**ngineers*. Instituto de Ingenieros eléctricos y electrónicos, dedicados a la creación de estándares.
IPv4/IPv6	Acrónimos de ***I**nternet **P**rotocol **v**ersion **4*** e ***I**nternet **P**rotocol **v**ersion **6***.Revisiones 4 y 6 respectivamente del protocolo de internet (IP).

ISA 100.11a	Estándar de tecnología de red inalámbrica abierta desarrollado por la *International Society of Automation.*
Lazarus	Entorno de desarrollo RAD basado en FreePascal
MCP9700	Termistor linear activo de bajo consumo del fabricante Microchip
MCS-31/51	Familias de micro-controladores de la casa Intel
Meshprog	Programador de motes para entorno linux
Microchip	Fabricante de componentes electrónicos y microcontroladores. www.microchip.com
Micro-controlador	También denominado **C**. Combinación de varios componentes electrónicos para formar un 'ordenador en un solo chip'.
nesC	Extensión del lenguaje C usado en TinyOS
OSI	Acrónimo de ***O**pen **S**ystem **I**nterconnection*
PIC	Familia de microcontroladores RISC del fabricante Microchip
PWM	Acrónimo de **P**ulse-**W**idth **M**odulation
RAM	Acrónimo de ***R**andom **A**ccess **M**emory*. Memoria de acceso aleatorio.
ROM	Acrónimo de ***R**ead **O**nly **M**emory*. Memoria de sólo lectura.
RS-232	Estandar de comunicaciones serie.
RS-485	Estandar de comunicaciones serie diferencial.
Sensor Hall	Sensor que detecta y mide campos magnéticos
Sistemes embebidos, empotrados	Sistemas diseñados con un propósito específico, en contraposición a sistema de computación general.
SPI	Acrónimo de ***S**erial **P**eripheral **I**nterface*.
TinyOS	Sistema operativo de red enfocado a programación de WSNs o motes.
UART	Acrónimo de ***U**niversal **A**synchronous **R**eceiver-**T**ransmitter*
UML	Acrónimo de ***U**nified **M**odeling **L**anguage*. Lenguaje Unificado de Modelado
USB	Acrónimo de ***U**niversal **S**erial **B**us*.Bus Universal Serie
Von Neumann/Harvard, arquitecturas	Arquitecturas de procesadores que usa la misma memoria para almacenar código y datos (primer caso) y diferentes memorias de código y datos (segundo caso). En el segundo se puede optimizar el ancho de palabra de la memoria de código.
Watchdog	Mecanismo de seguridad en dispositivos electrónicos que provoca un reset del sistema cuando detecta un bloqueo del sistema.
Wireless	Inálambrico
WSN	Wireless Sensor Network / Wireless Sensor Node
Xscale	Familias de micro-controladores de la casa Intel
Zigbee	Especificación de protocolos de comunicación inalámbrica en entornos WPAN.

9. Bibliografía

[1] **UOC** (2010). "*Arp@:Embedded Systems Lab@Home"* [fecha de consulta: 2011-09-19]. <http://cv.uoc.edu/app/mediawiki14/>

[2] **Varios** (2010). "*TinyOS Documentation Wiki"* [fecha de consulta: 2011-09-21]. <http://docs.tinyos.net/tinywiki/index.php/Main_Page>

[3] **LEVIS, Philip y GAY, David** (2009). "*TinyOS Programming"*. Cambridge University Press. 282p. ISBN: 978-0521896061

[4] **FALUDI, Robert** (2010). "*Building Wireless Sensor Networks"*. O'Reilly Media. 322p. ISBN: 978-0596807733

[5] **NXP Laboratories UK Ltd** (2008). "*Low Complexity Antenna Diversity For IEEE 802.15.4 2.4 GHz PHY*". [fecha de consulta: 2011-10-04]. <http://www.jennic.com/files/support_files/JN-AN-1079 Coexistence of IEEE 802.15.4 In The 2.4GHz Band-1v0.pdf>

[6] **Zolertia Z1** (2011). "*Z1 Platform*". [fecha de consulta: 2011-10-25]. http://www.zolertia.com/ti

[7] **Contiki OS** (2012). "*The Contiki OS*". [fecha de consulta: 2012-06-20]. http://www.contiki-os.org/

[8] **Wikipedia** (2012). "*Glosario*". [fecha de consulta: 2012-06-23]. http://es.wikipedia.org

10. Anexos

10.1. Manual de usuario

La aplicación `./mote/BaseStationAppC` proporciona un intérprete integrado de comandos para interactuar con la red de sensores.

```
+HELP

        AYUDA:

        * Identify (print node id): I<CR>
        * GetConfig (print net config): G<CR>
        * SetConfig (change net config): S (0x)TSTSTTTTBSBSBTBT<CR>
                TSTS: Hex Temp Sec Alarm ; TTTT: Hex Temp TH Alarm
                BSBS: Hex Batt Sec Alarm ; BTBT: Hex Batt TH Alarm
        * Debug (toggle debug): D (0x)DSDS<CR>
                DSDS: Hex Debug Secs
        * ChangeChannel (cambiar canal - decimal 11..26) nn: N CH<CR>
        * AckAlarm (ack alarma): A [B|T|M]ALRM<CR>
                B = Batt, T=Temp, M=Manual; ALRM: hex alarm serial
        * Reboot (reboot Node): R NNNN<CR>
                NNNN: Hex node id
        * Ping: (ping network): P<CR>
        * HELP: Esta pantalla de ayuda: [H|?]<CR>
```

Ilustración 23 – Comandos incorporados en el intérprete de la aplicación BaseStationAppC

La aplicación del PC simplemente captura y filtra los comandos y mensajes enviados por el mote coordinador.

Cuando ejecutamos el programa del PC, accedemos a la pantalla principal:

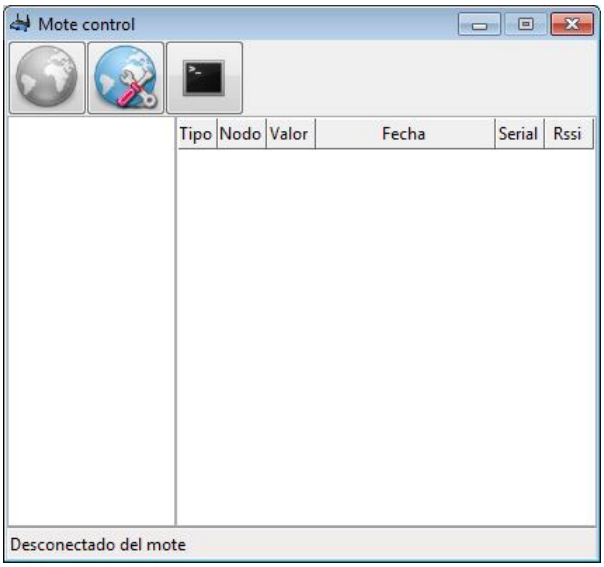

Ilustración 24 – Pantalla principal de la aplicación de control

Podemos acceder a los siguientes botones con la funcionalidad que se describe:

Botón	Explicación
	Botón que permite conectar y desconectar del mote coordinador
	Botón que permite configurar los parámetros operativos de la red de sensores
	Botón que despliega la visualización de mensajes enviados y recibidos al mote coordinador

Ilustración 25 – Botones de la ventana principal aplicación PC

Si pulsamos en el segundo botón de la tabla anterior  , nos aparecerá el siguiente diálogo de configuración de la red:

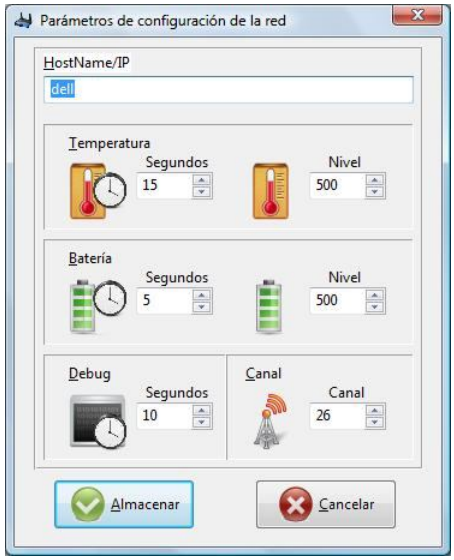

Ilustración 26 – Diálogo de configuración de parámetros de red

Al pulsar en el botón  se accede a una sección oculta de la aplicación:

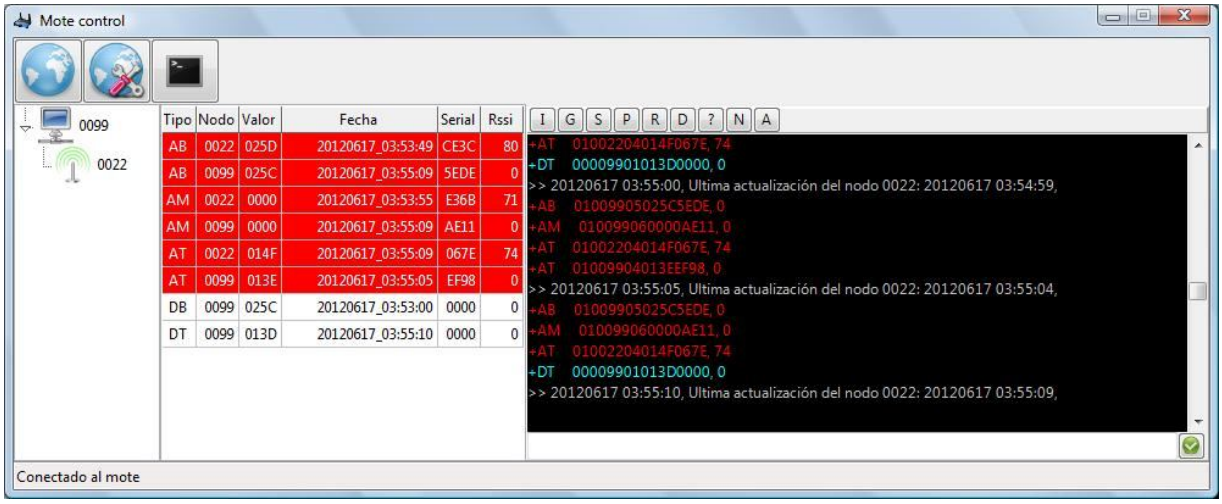

Ilustración 27 – Pantalla principal con ventana de trazas activada

En la sección derecha podremos ver el intercambio de mensajes entre la aplicación y el mote coordinador de red.

La confirmación de alarmas se realiza **haciendo doble clic** sobre tabla de alarmas. Si hay alarmas, nos mostrará un diálogo que permite seleccionar la alarma a confirmar (como en la ilustración siguiente), en caso de que no haya alarmas, mostrará un mensaje '***No hay alarmas para confirmar***.'

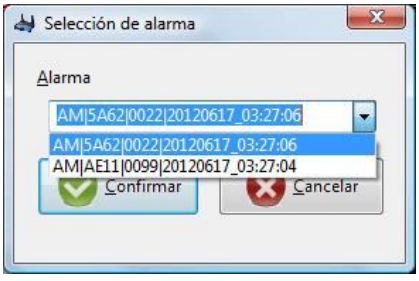

Ilustración 28 – Diálogo de confirmación de alarmas

Por último, queda por describir los botones que aparecen al mostrar la pantalla de debug:

Ilustración 29 – Diálogo de confirmación de alarmas

Cada uno de ellos tiene una correspondencia directa con los comandos del intérprete de la aplicación `./mote/BaseStationAppC` mostrados en "Ilustración 23 – Comandos incorporados en el intérprete de la aplicación BaseStationAppC":

Botón	Comando
I	**Identify** (print node id): I<CR>
G	**GetConfig** (print net config): G<CR>
S	**SetConfig** (change net config): S (0x)TSTSTTTTBSBSBTBT<CR> TSTS: Hex Temp Sec Alarm ; TTTT: Hex Temp TH Alarm BSBS: Hex Batt Sec Alarm ; BTBT: Hex Batt TH Alarm
P	**Ping**: (ping network): P<CR>
R	**Reboot** (reboot Node): R NNNN<CR> NNNN: Hex node id
D	**Debug** (toggle debug): D (0x)DSDS<CR> DSDS: Hex Debug Secs
?	**HELP**: Esta pantalla de ayuda: [H|?]<CR>
N	**ChangeChannel** (cambiar canal - decimal 11..26) nn: N CH<CR>
A	**AckAlarm** (ack alarma): A [B|T|M]ALRM<CR> B = Batt, T=Temp, M=Manual; ALRM: hex alarm serial

Ilustración 30 – Botones de la ventana de debug

10.2. Ejecución y compilación

• Ejecución y compilación de aplicación Mote

Las diferentes aplicaciones se proporcionan compiladas.

En caso requerir la recompilación de los diferentes módulos del mote, habría que situarse en el directorio `./mote/BaseStationAppC` o `./mote/NodeStationAppC` y ejecutar el comando `make cou24 install.NODEID` siendo NODEID la identificación numérica interna del mote en TinyOS.

Las dos aplicaciones deben usar NODEIDs **diferentes**, y una vez compilados podrían instalarse usando el comando ***meshprog***, sustituyendo los valores correspondientes al puerto USB al que está conectado el mote y el NODEID elegido en la compilación (ver "Ilustración 31- Ejemplo de compilación e instalación de código en mote COU24")

```
COU24$              make              cou24              install.34
COU24$ meshprog -t /dev/ttyUSB0 -f ./build/cou24/main.srec.out-34
```

Ilustración 31- Ejemplo de compilación e instalación de código en mote COU24

Cada uno de los directorios anteriores tiene la siguiente funcionalidad:

`./mote/NodeStationAppC`: Funcionalidad de mote sensor remoto.

`./mote/BaseStationAppC`: Funcionalidad de mote local conectable a USB, coordinador de la red de sensores y arbitraje de la red mediante el enlace al software desarrollado al efecto.

• Ejecución y compilación del redirector Serie-TCP

Cambiar al directorio `./PC/linux/Serial2TCP/remserial-1.4` y ejecutar el comando `make`

Para ejecutar el redirector debe usarse un comando similar a "Ilustración 32 – Ejemplo de ejecución del redirector RS-232/TCP" siendo el parámetro `19.200` la velocidad del puerto serie (estándar para la plataforma COU24) y ttyUSB0 el dispositivo USB que tiene instalado el software `BaseStationAppC` del punto anterior

```
COU24$ ./remserial -d -p 2000 -s "19200 raw icanon cs8 sane"
/dev/ttyUSB0 &
```

Ilustración 32 – Ejemplo de ejecución del redirector RS-232/TCP

• Ejecución y compilación de aplicación PC

La aplicación PC se ha desarrollado mediante el entorno de programación `Lazarus` basado en FreePascal. Requiere de la instalación de Lazarus y de los módulos `lNet`, `UniqueInstance` y `CmdLine` (ver sección 11.5 – Software y licencias).

Una vez instalados, abrir el archivo de proyecto `lazarus ./PC/windows/source/MoteCtrl.lpi` y usar el comando Ejecutar-> Construir (o teclas Ctrl+F9) dentro de Lazarus. Tras la compilación encontraremos el fichero `MoteCtrl.exe` en el mismo directorio del proyecto.

Ejecutar como cualquier otro programa.

10.3. Software y licencias

Plataforma desarrollo Mote

Linux (Osian): Ya no está disponible para descarga.
Licencia: >> BSD License vía http://en.wikipedia.org/wiki/OSIAN

TinyOS: http://www.tinyos.net/
Licencia: >> New BSD license:
http://www.opensource.org/licenses/bsd-license.php

Plataforma desarrollo Aplicación de control

Lazarus: http://www.lazarus.freepascal.org/ | http://sourceforge.net/projects/lazarus/files
Entorno de programación multiplataforma clónico a delphi basado en FreePascal

```
Licencia:    htp://wiki.lazarus.freepascal.org/Lazarus_Faq#Licensing
             >> LCL is licensed under the LGPL with an exception, which
             >> allows you to link to it statically without releasing
             >> the source of your application.
```

Componentes para Lazarus

http://wiki.freepascal.org/lNet → **Conexión TCP**
Licencia: >> lNet units (units in lib and lazaruspackage
>> directories) are licensed under a modified LGPL
>> license. See file lnet/LICENSE and lnet/LICENSE.ADDON.

http://wiki.freepascal.org/UniqueInstance → **Control de una única ejecución**
Licencia: >> Modified LGPL

http://wiki.freepascal.org/CmdLine → **Emulación de consola (Debug)**
Licencia: >> LGPL

Otro Software

http://lpccomp.bc.ca/remserial → **Redirección de puerto serie a TCP**
Licencia: >> GNU GPL