Monitorización de la temperatura de un CPD mediante una red de sensores inalámbricos

Estudiante: **Isaac Peña Torres**

Ingeniería Técnica de Informática de Sistemas

Consultor: **Jordi Bécares Ferrés**

Fecha Entrega: enero 2011

Resumen

Este proyecto se enmarca dentro del área de sistemas empotrados y redes de sensores inalámbricos. Se basa en el sistema operativo TinyOS y en una red de sensores inalámbricos para el desarrollo de una aplicación encargada de monitorizar la temperatura ambiental de un espacio físico.

Aunque existen multitud de entornos y situaciones para los que la aplicación desarrollada podría ser de utilidad, en este proyecto nos hemos centrado en el caso concreto de controlar la temperatura de un Centro de Proceso de Datos (CPD), lo que permitirá asegurar la continuidad de los servicios ofrecidos por los sistemas de información de una organización.

La aplicación desarrollada se encarga de leer regularmente los sensores en los nodos inalámbricos y analizar los valores recogidos. En caso de detectar que esos valores se encuentren fuera de unos rangos definidos nos informará de ello mediante el envío de un mensaje, que recogeremos a través de un nodo inalámbrico conectado a un PC.

Además se ha provisto de otras funcionalidades para la interacción remota con los nodos mediante una aplicación cliente que ejecutaremos en un terminal.

Mediante el sistema desarrollado se pretende que la organización sea más consciente del estado de sus máquinas y anticiparse a una situación de pérdida de información, parada de servicios o parada por sobrecalentamiento de servidores.

Índice de contenido

1. Introducción

1.1. Justificación

Se denomina Centro de Proceso de Datos (CPD) a la ubicación física donde se concentran los recursos necesarios para el procesamiento de la información de una organización. Son creados principalmente por organizaciones de gran tamaño que poseen información crítica que quieren tener controlada, aunque también pueden ser creados por organizaciones de tamaño medio.

Normalmente se trata de un edificio o sala de gran tamaño rigurosamente acondicionada para evitar el acceso por parte de personal malintencionado o no autorizado, e impedir que se produzcan situaciones de riesgo que puedan interrumpir la continuidad de los servicios ofrecidos por la organización. Existen multitud de elementos que ayudan a alcanzar estos objetivos, como por ejemplo: la redundancia de lineas de corriente y conexiones de datos, control de la temperatura mediante sistemas de aire acondicionado, sistemas de vigilancia, suelo o techo técnico, medidas de extinción de incendios adecuadas al material eléctrico, puertas y armarios ignífugos, alarmas, acceso físico por identificación, etc.

Las organizaciones realizan un gran desembolso económico en el diseño, creación y mantenimiento de sus Centros de Proceso de Datos con el objetivo de ofrecer a sus usuarios un servicio continuo de calidad y tener controlados sus datos.

La idea de este proyecto parte de esta necesidad por parte de las organizaciones. Se añade otro mecanismo más, alineado con los ya citados, para conseguir evitar una situación que pueda poner en peligro la pérdida de información de la organización o la imposibilidad del acceso a ella.

Como ya se ha comentado, los Centros de Proceso de Datos poseen máquinas de aire acondicionado que mantienen la sala o edificio a una temperatura óptima para el funcionamiento de los servidores o material electrónico. La temperatura óptima recomendada para evitar averías por sobrecalentamiento está entre los 21 y 23 º C, exactamente es de 22.3 º C.

La aplicación desarrollada monitorizará periódicamente la temperatura de la sala en diferentes puntos estratégicos mediante nodos inalámbricos provistos de sensores, e informará en caso de detectar una temperatura fuera del rango establecido, o en el caso de que las baterías necesitan ser reemplazadas.

1.2. Objetivos

Se identifican varios tipos de objetivos a cumplir durante el desarrollo de este proyecto referentes a distintos ámbitos:

- Cumplimientos de objetivos técnicos que cubran las necesidades planteadas y resulte de utilidad al cliente en el entorno analizado.
- Objetivos de cumplimiento en tiempo, calidad del proyecto y plazos de entrega fijados a lo largo del desarrollo del proyecto.
- Objetivos propios de la plataforma utilizada, en este caso concreto con respecto a TinyOS y sensores inalámbricos.

1.2.1. Objetivos técnicos

Para que la herramienta sea realmente útil es necesario no solo que monitorice la temperatura de la sala donde se ubicarán los sistemas de información, también será necesario que nos informe si se detecta una situación de riesgo, es decir, que se detecte unos valores por los sensores fuera de los rangos establecidos.

También sería práctico el poder comunicarnos con las motas remotamente y realizar modificaciones en los valores umbrales definidos, así como realizar pruebas de envíos de mensajes y comprobar su funcionamiento.

Para evitar tener una falsa sensación de seguridad, aumentar la confianza en el producto y ,como se explicará más adelante, asegurarnos del correcto funcionamiento de los sensores y componentes electrónicos, sería necesario estar al tanto del nivel de baterías de los sensores. De esta manera nos aseguramos que el sistema esté operativo en todo momento, tener datos de su consumo, y anticiparnos a un cambio de baterías masivo en los nodos que componen la red.

Para ello nos marcamos los siguientes hitos:

- Monitorización de la temperatura ambiental y nivel de baterías.
- Envío de alertas al detectar temperaturas o nivel de baterías fuera de rango.
- Desarrollo de aplicación cliente que permita la comunicación con las motas remotas.
- Modificación remota de los valores umbrales.
- Envío de información bajo petición de forma remota: valores recogidos por los sensores y valores asignados a los umbrales.

1.2.2. Objetivos sobre la planificación y calidad del proyecto

Se da una gran importancia a cumplir con los plazos de entrega y con el compromiso establecido en un principio en la planificación, así como en la calidad del trabajo desarrollado y material entregado.

1.2.3. Objetivos propios de la plataforma utilizada

Como ya se ha comentado, la aplicación se construye sobre el sistema operativo de código abierto TinyOS. Este sistema posee unas características y restricciones que hay que tener en cuenta a la hora de utilizarlo como base de un proyecto. Por una parte proporciona un enfoque basado en componentes que es necesario conocer, este aporta una gran flexibilidad que hace que sea muy sencillo incorporar nuevos cambios de forma rápida. Pero también hay que tener en cuenta que está diseñado para dispositivos con pocos recursos, con poca memoria y baja capacidad de procesamiento, además de estar enfocado al bajo consumo energético.

En base a esto se proponen los siguientes objetivos:

- Optimización de ciclos de procesamiento. Evitar realizar cálculos que consuman muchos ciclos de CPU en las motas en la medida de lo posible.
- Optimización de memoria. Evitar definir variables innecesarias e intentar definir los tipos de mensajes a enviar ajustados a las necesidades.
- Desactivar los sensores mientras no sea necesario su consulta.
- Optimización de consumo de energía en las motas. Optimizar el valor de la frecuencia de disparo de contadores en la consulta de sensores.

Estos puntos hay que tenerlos en cuenta a los largo de todo el desarrollo del proyecto para evitar derrochar recursos innecesariamente. Con un buen diseño e implementación de las funcionalidades en el código que se alojará en los sensores ganaremos en autonomía y a la larga optimizaremos los recursos disponibles.

En un proyecto en el que hubiera varios cientos de sensores distribuidos en un área amplia, un aumento de la vida de las baterías puede significar un ahorro de coste considerable en recursos humanos y económicos que la organización tendría que invertir en la sustitución de las baterías.

1.3. Enfoque y método seguido

El método seguido a lo largo del desarrollo del proyecto no ha sido constante en todas las fases. En un principio se toma un enfoque de cumplimiento de objetivos y desarrollo del proyecto de tipo incremental. Al hacer la propuesta de proyecto se hace una diferencia explícita entre objetivos básicos y de obligado cumplimiento, y objetivos secundarios.

En primer lugar se desarrolla una aplicación que proporciona las funcionalidades básicas y, a partir de esta, se implementan progresivamente más funcionalidades, se va mejorando el diseño de la aplicación, optimizando el código para el aprovechamientos de los recursos y corrigiendo los errores detectados.

Durante el transcurso del proyecto de ha ido redefiniendo y sopesando cada objetivo y las alternativas disponibles, controlando su complejidad y riesgo, procurando siempre el cumplimiento de los plazos fijados hasta llegar a la solución final.

El concepto inicial del proyecto, análisis de necesidades, diseño de la arquitectura y plan de trabajo se definen utilizando un enfoque de cascada, seguida por la iteración de prototipos que finaliza con el producto final.

Para llevar un mayor control de los cambios realizados durante el desarrollo de la aplicación se ha usado un sistema de control de versiones Git.

Se ha procurado durante todo el desarrollo que la aplicación fuera usable en todas las fases, procurando solucionar lo antes posible los fallos surgidos de compilación y/o ejecución. Con esto se ha pretendido evitar la acumulación y complicación de errores de diferentes orígenes, hecho que normalmente desencadena un aumento en la inversión de tiempo para su resolución.

1.4. Planificación del proyecto

Se definen las siguientes tareas:

Tarea 1. Propuesta del Trabajo Fin de Carrera

Duración: 5 días (22/09/10 – 28/09/10)

Descripción: se genera un documento donde se propone la temática del proyecto, una propuesta de título que tomará, se realiza una descripción del proyecto a realizar a grandes rasgos, y se definen los objetivos y la arquitectura.

Tarea 2. Preparación del entorno de trabajo

Duración: 5 días (22/09/10 – 28/09/10)

Descripción: se instalan los paquetes del sistema operativo TinyOS, se instala el plugin de Eclipse Yeti para programar con Nesc, se compilan aplicaciones de prueba, se instala meshbean y se cargan aplicaciones en las motas para probar su funcionamiento.

Tarea 3. Lectura de documentación

Duración: 78,5 días (21/09/10 – 07/01/11)

Descripción: lectura de tutoriales de la página web del proyecto TinyOS y consulta de documentación para la resolución de dudas para conseguir los objetivos propuestos y proponer un buen diseño de la aplicación. Se considera que está tarea estará activa a lo largo de todo el proyecto pues tendremos que ir recogiendo información en función de la tarea que estemos llevando a cabo hasta finalizar el proyecto.

Tarea 4. Plan de trabajo del Trabajo de Fin de Carrera

Duración: 8 días (28/09/10 – 07/10/10)

Descripción: se genera el plan de trabajo del proyecto donde se incluye una introducción, un diagrama de bloques, descripción de objetivos y tareas, cronograma, diagrama de Gant y recursos que se utilizarán.

Tarea 5. Diseño de la aplicación

Duración: 1 día (08/10/10 – 08/10/10)

Descripción: se realiza un diseño de la arquitectura de la aplicación a desarrollar.

Tarea 6. Generación de código en la mota remota

Duración: 66 días (11/10/10 – 10/01/11)

Descripción: esta tarea se refiere al desarrollo de la aplicación que irá en las motas remotas. Estará dividida en subtareas referentes a las diferentes funcionalidades a implementar y testeos a realizar en el sistema desarrollado.

Tarea 6.1 Recogida de temperatura

Duración: **5 días (11/10/10 - 15/10/10)**

Descripción: se recoge la temperatura ambiental mediante los sensores embebidos en la mota.

Tarea 6.2 Detección y configuración de valores críticos

Duración: 3 días (18/10/10 – 20/10/10)

Descripción: una vez tenemos los valores de temperatura tenemos que detectar cuando supera una temperatura crítica. El valor máximo de temperatura se definirá en el código de la mota remota.

Tarea 6.3 Generación de mensajes

Duración: 3 días (21/10/10 – 25/10/10)

Descripción: se define el mensaje que se enviará a la mota base avisando de la detección de un valor de temperatura crítico.

Tarea 6.4. Envío de mensaje a mota base

Duración: 9 días (26/10/10 – 05/11/10)

Descripción: se implementa la funcionalidad de envío de mensajes a la mota base. Mientras no tenemos todavía desarrollada la aplicación de consola que escuchará y mostrará por pantalla los mensajes recibidos hacemos uso de la aplicación ya desarrollada Listen para comprobar el envío de mensajes.

Tarea 6.5. Detección en el código de mota remota

Duración: 2 días (08/11/10 – 09/11/10)

Descripción: se localizan los errores surgidos a partir de las funcionalidades implementadas.

Tarea 6.6. Corrección de errores en mota remota

Duración: 26 días (10/11/10 – 15/12/10)

Descripción: se corrigen los errores detectados.

Tareas 6.7. Optimización de código en mota remota e implementación de nuevas funcionalidades.

Duración: 18 días (16/12/10 – 10/01/11)

Descripción: esta tarea se refiere a la optimización del diseño de la aplicación organizando el código en diferentes componentes, implementación de nuevas funcionalidades e implementación de cambios necesarios para cumplir los objetivos indicados anteriormente.

Tarea 7 Generación del código en el sistema base

Duración: 74 días (11/10/10 – 20/01/11)

Descripción: en esta tarea prepararemos la aplicación que irá en la mota base y la aplicación de consola que se comunicará con las motas.

Tarea 7.1 Código principal

Duración: 8 días (11/10/10 - 20/10/10)

Descripción: carga de la aplicación BaseStation en la mota base conectada a un pc. La

aplicación se encuentra en la carpeta apps del sistema operativo TinyOS.

Tarea 7.2 Recepción de mensaje de mota remota

Duración: 8 días (21/10/10 – 01/11/10)

Descripción: comprobar que el reenvío de mensajes funciona correctamente.

Tarea 7.3 Generación de código de aplicación en linea de comandos

Duración: 6 días (02/11/10 – 09/11/10)

Descripción: se implementas las siguientes funcionalidades.

1. Escucha de mensajes: implementación de funcionalidad de escucha de mensajes y mostrado por consola de mensajes recibidos.
2. Modificación de valores remotamente: implementación de modificación de rango de valores de temperatura remotamente.
3. Solicitud de envío de mensajes: implementación de solicitud de envío de mensajes a una mota.

Tarea 7.4 Detección de errores

Duración: 8 días (10/11/10 - 19/11/10)

Descripción: se localizan los errores producidos en la aplicación de consola y aplicación de nodo base.

Tarea 7.5 Corrección de errores

Duración: 26 días (22/11/10 - 27/12/10)

Descripción: se corrigen los errores detectados.

Tarea 7.6. Optimización de código en sistema base y ampliación de funcionalidades

Duración: 18 días (28/12/10 - 20/01/11)

Descripción: una vez tenemos una aplicación usable que proporciona las funcionalidades básicas nos dedicamos a optimizar el diseño y a implementar de forma incremental nuevas funcionalidades. Se añade la monitorización de las baterías en las motas remotas, se libera a las motas de la realización de los cálculos de conversión entre counts y grados, y counts y voltios, se defines tres tipos de mensajes (InfoMsg, BatMsg y TempMsg), se redefine el diseño de la aplicación dividiéndola en componentes con funcionalidades concretas, etc.

Tarea 8. Generación de memoria

Duración: *25 días (10/12/10 – 13/01/11)*

Descripción: se genera la memoria del Trabajo Fin de Carrera. Se trata del presente documento.

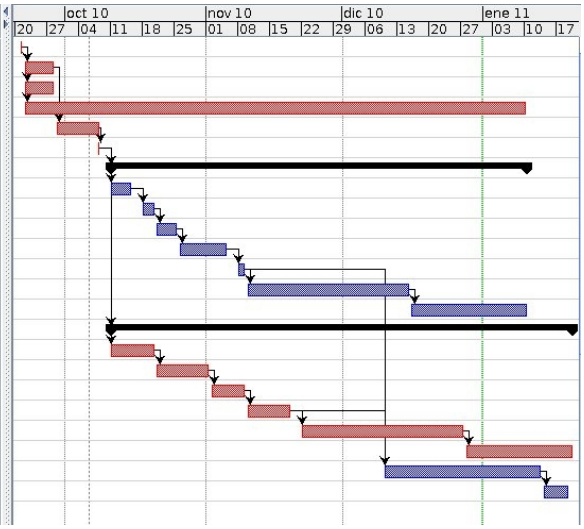
Tarea 9. Generación de presentación

Duración: 4 días (14/01/11 - 19/01/11)

Descripción: se genera la presentación del Trabajo Fin de Carrera. Recoge los conceptos más importantes de trabajo realizado.

A continuación se muestra una imagen del cronograma de tareas y el diagrama de Gant, donde se puede ver la estimación de tiempo dedicada a cada tarea del proyecto, la fecha de inicio y finalización y la relación existente entre tareas:

	Nombre	Duración	Inicio	Terminado	Pre...
1	Comienzo TFC	1 day?	21/09/10 8:00	21/09/10 17:00	
2	Propuesta TFC	5 days?	22/09/10 8:00	28/09/10 17:00	1
3	Preparación de entorno de trabajo	5 days?	22/09/10 7:00	28/09/10 17:00	1
4	Lectura de documentación	78,5 days?	21/09/10 8:00	7/01/11 13:00	1
5	Plan de trabajo TFC	8 days?	28/09/10 8:00	7/10/10 17:00	2
6	Diseño de aplicación	1 day?	8/10/10 8:00	8/10/10 17:00	5
7	**Generación código mota remota**	**66 days?**	**11/10/10 8:00**	**10/01/11 17:00**	**6**
8	Recogida de temperatura	5 days?	11/10/10 8:00	15/10/10 17:00	6
9	Detección y conf. de valores críticos	3 days?	18/10/10 8:00	20/10/10 17:00	8
10	Generación de mensaje	3 days?	21/10/10 8:00	25/10/10 17:00	9
11	Envío de mensaje a mota base	9 days?	26/10/10 8:00	5/11/10 17:00	10
12	Detección errores código mota remota	2 days?	8/11/10 8:00	9/11/10 17:00	11
13	Corrección de errores mota remota	26 days?	10/11/10 8:00	15/12/10 17:00	12
14	Optimización de código mota remota	18 days?	16/12/10 8:00	10/01/11 17:00	13
15	**Generación código mota estación base**	**74 days?**	**11/10/10 8:00**	**20/01/11 17:00**	**6**
16	Código principal	8 days?	11/10/10 8:00	20/10/10 17:00	6
17	Recepción de mensaje de mota remota	8 days?	21/10/10 8:00	1/11/10 17:00	16
18	Aplicación linea de comandos	6 days?	2/11/10 8:00	9/11/10 17:00	17
19	Detección de errores mota base	8 days?	10/11/10 8:00	19/11/10 17:00	18
20	Corrección de errores mota base	26 days?	22/11/10 8:00	27/12/10 17:00	19
21	Optimización de código mota base	18 days?	28/12/10 8:00	20/01/11 17:00	20
22	Generación de memoria	25 days?	10/12/10 8:00	13/01/11 17:00	12;19
23	Generación de presentación	4 days?	14/01/11 8:00	19/01/11 17:00	22



Durante el desarrollo del proyecto ha resultado más práctico abordar las tareas de detección y corrección de errores de manera iterativa. Conforme se detectaba un error seguidamente se corregía hasta resolver todos los errores antes de seguir implementando u optimizando el código.

1.5. Productos obtenidos

Al finalizar el proyecto se han obtenido los siguientes productos:

- **Aplicación TempReporter:** aplicación elaborada con el lenguaje de programación NesC que se cargará en las motas remotas. Está encargada de la lectura de los sensores a una frecuencia definida, enviar mensajes por la red hacia la mota base, recepción de mensajes de control y ejecución de la acción solicitada desde la aplicación RemoteCon.

- **Aplicación RemoteCon:** aplicación desarrollada en java que se ejecutará en un terminal y que nos permitirá recepcionar los mensajes y realizar algunas acciones sobre las motas de manera remota.
- **Memoria del proyecto:** documento que describe el trabajo realizado en el Trabajo Fin de Carrera. Se trata del presente documento y recoge los conocimientos aprendidos durante el desarrollo del proyecto.
- **Presentación:** documento que sintetiza la información sobre el trabajo que se ha realizado en el Trabajo Fin de Carrera.

1.6. Descripción de capítulos posteriores

En los siguientes capítulos de este documento se describirán las tecnologías utilizadas durante el desarrollo del proyecto como son: el sistema operativo en el que se basa la aplicación TempReporter, **TinyOS**, el lenguaje de programación utilizado para la aplicación ubicada en las motas remotas, **Nesc**, los componentes **hardware** utilizados, el protocolo de comunicaciones usado, **ZigBee**, una descripción de la aplicación y justificación sobre las decisiones de diseño tomadas a la hora de desarrollar los diferentes componentes de la aplicación.

Más adelante se hace una reflexión y se muestran algunas conclusiones sacadas sobre el trabajo realizado y el ámbito del proyecto.

Se finaliza el documento con un glosario que contiene los términos específicos de este ámbito, usados a lo largo del documento, y una referencia bibliográfica del material y los recursos utilizados.

2. Antecedentes

En este apartado se hablará de toda la base teórica en la que se sustenta el proyecto. Se hablará de las redes de sensores inalámbricos, del sistema operativo TinyOS, el lenguaje de programación NesC, el hardware que usaremos y el protocolo de comunicaciones para realizar la comunicación entre motas, ZigBee.

2.1. Estado del arte

Las redes de sensores tienen su origen en iniciativas militares. Actualmente son un campo de investigación en auge y muy activo que está evolucionando bastante en los últimos años.

Una red de sensores está compuesta por una red de ordenadores muy pequeños (llamados nodos o motas), con capacidades muy limitadas y de bajo coste económico, los cuales están equipados con sensores capaces de recoger información del medio, procesarla y enviarla.

Se comunican mediante redes inalámbricas en modo ad hoc sin infraestructura física definida ni administración central. Los nodos suelen tener un bajo gasto energético lo que aumenta en gran medida su autonomía.

Los nodos de una red colaboran y se coordinan para llevar a cabo una tarea común. Éstos son fácilmente desplegables y pueden ser al mismo tiempo emisores, receptores, encaminadores entre nodos y recoger información de sus sensores.

Las redes de sensores tienen muchas aplicaciones en distintos ámbitos como: entornos industriales, domótica, entornos militares o detección ambiental.

Como predecesores de las redes de sensores se consideran la Sound Surveillance System (SOSUS), red de boyas sumergidas instaladas en EEUU durante la Guerra Fría para detectar submarinos usando detectores de sonido.

La investigación de las redes de sensores comenzó en 1980 con el proyecto Distributed Sensor Networks (DSN) de la agencia militar de investigación de EEUU (Defense Advanced Research Projects Agency) (DARPA).

Una de las características que deben tener los nodos de la red es el bajo consumo energético, para conseguirlo se cuidará el consumo de energía en el hardware, el sistema operativo y el lenguaje de programación utilizado para desarrollar las aplicaciones.

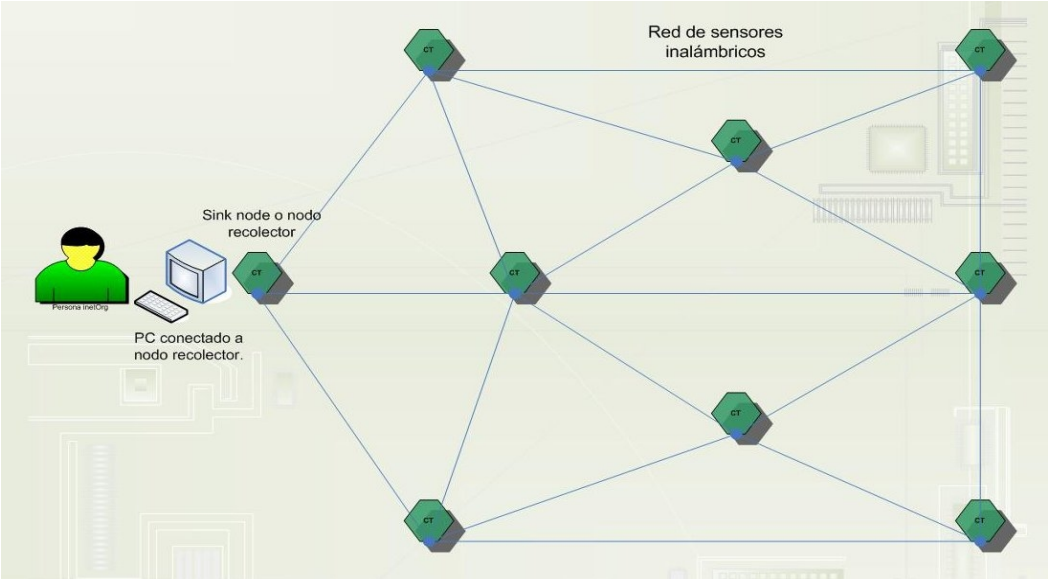
2.1.1. Red de sensores

Como ya se ha comentado, una red de sensores (Wireless Sensor Networks) está compuesta por un conjunto de nodos interconectados mediante redes inalámbricas y que son capaces de recoger información mediante sensores que tienen integrados.

Las redes de sensores están compuestas por los siguientes elementos:

- **Sensores:** recogen datos y los convierten en señales eléctricas. Normalmente están incrustados en las motas pero se pueden usar sensores externos conectados a puertos de expansión en la motas.
- **Nodos o motas:** recogen la información de los sensores, la procesan y la envían a la red inalámbrica por la radio.
- **Nodo recolector o sink node:** nodo que recoge los datos de las red de sensores y los reenvía a la estación base, o recoge datos de las estación base y los envía a los nodos.
- **Estación base:** es un ordenador que tiene conectado el nodo recolector y a través de este recoge los datos o los envía a la red de sensores.
- **La red:** es la red resultante formada por todos los nodos.

Las redes de sensores presentan una estructura como la que se puede ver en la siguiente imagen:



Estas redes de sensores poseen multitud de usos prácticos:

- **Eficiencia energética:** uno de los posibles usos de las redes de sensores es para controlar el uso eficaz de la electricidad, como se hace en Japón o España.
- **Entornos de alta seguridad:** para lugares con altos niveles de seguridad se utilizan las redes de sensores en la detección de ataques terroristas, es el caso de centrales nucleares, aeropuertos, edificios del gobiernos. Permiten obtener información que otros dispositivos tradicionales como cámaras de vídeo son incapaces de detectar.
- **Sensores ambientales:** las redes de sensores son de gran utilidad para recoger información ambiental de muchos tipos diferentes en áreas extensas: temperatura, humedad, luminosidad, actividad sísmica, etc.
- **Sensores industriales:** el reducido tamaño de los sensores es un punto a favor para poder recolectar información donde se requiera dentro de las fábricas.
- **Automoción:** son de gran utilidad para la regulación del tráfico e información de accidentes y atascos como complemento a las cámaras.
- **Medicina:** mejorará la calidad de vida de los pacientes que necesiten tener controladas sus constantes vitales.
- **Domótica:** otro ámbito en el que sus características como el tamaño, velocidad de despliegue y bajo coste lo hacen ideal para automatizar tareas en el hogar a un precio asequible.

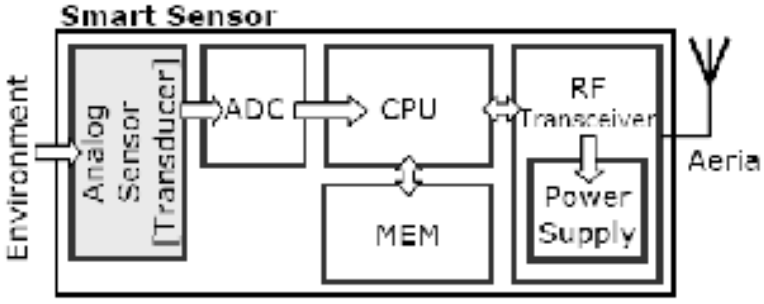
Las características propias de las redes de sensores son las siguientes:

- **Topología dinámica:** la topología siempre es cambiante y los sensores tienen que adaptarse para poder comunicar nuevos datos recogidos.

- **Variabilidad del canal:** el canal radio es un canal muy variable en el que fenómenos como la atenuación, desvanecimientos lentos e interferencias pueden producir errores en los datos.
- **No se utiliza infraestructura de red:** las redes de sensores no necesitan de una infraestructura para poder operar. Los nodos pueden ser emisores, receptores o enrutadores de información. Aunque sí es necesario que haya un nodo recolector conectado aun pc (también llamado sink node) que recolecta la información y por el cual se recoge la información generada normalmente en tiempo discreto. La información generalmente es adquirida por un ordenador conectado a este nodo y es sobre este ordenador en el cual recae la posibilidad de transmitir los datos por tecnologías inalámbricas o cableadas según sea el caso.
- **Tolerancia a errores:** nodo sensor debe ser capaz de seguir funcionando incluso si se producen errores en el sistema propio.
- **Comunicaciones multisalto o broadcast:** en aplicaciones en redes de sensores es común el uso de protocolos que permitan comunicaciones multisalto (multi-hop), como AODV, DSDV, EWMA u otras, aunque también se suele utilizar en envío de mensajes por broadcast.
- **Consumo energético:** ya que cuentan con un suministro de energía limitado, normalmente un par de pilas de tipo R6 (AA), aunque también podría ser a través de placas solares u otros métodos, es necesario cuidar el consumo de energía a todos los niveles para aumentar su autonomía: bajo consumo del procesador y transceptor de radio y un software que sea restrictivo en este tema.
- **Limitaciones hardware:** es necesario tener un hardware muy sencillo para reducir el consumo energético, lo que reduce la capacidad de proceso.
- **Costes de producción:** dado que para la recolección de datos fiables es necesario desplegar un gran número de sensores, lo nodos son económicos de hacer si se producen en grandes cantidades.

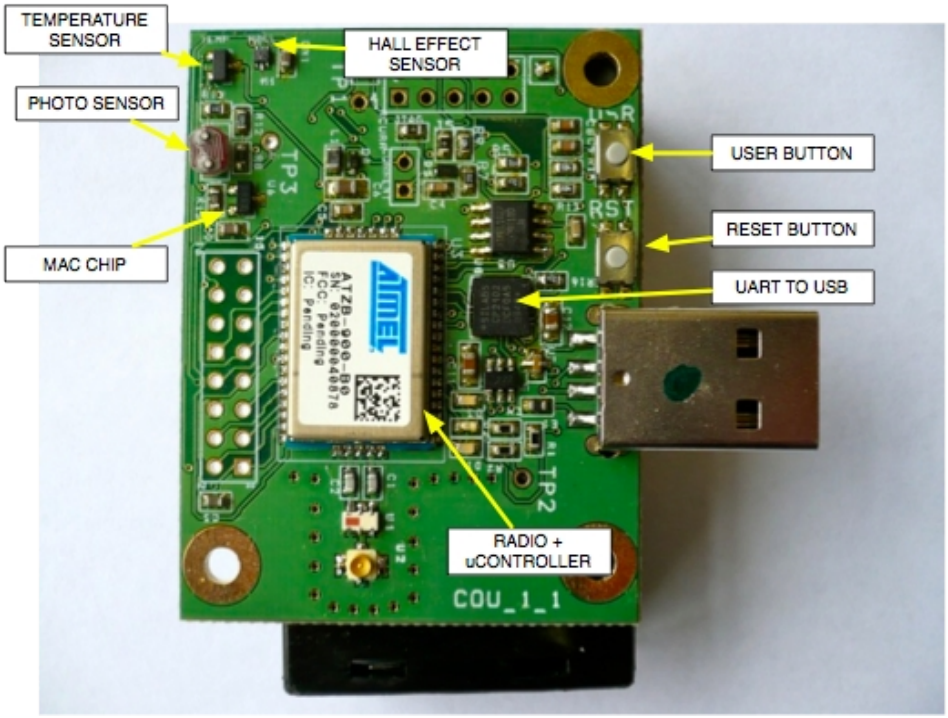
2.1.2. Hardware utilizado

Un nodo o mota es un circuito electrónico que posee un microcontrolador, un controlador radio con una antena, algunos sensores y una fuente de energía para funcionar. Una estructura genérica de una mota sería la siguiente:



Las motas que usaremos en este proyecto poseen un microcontrolador atmega1281, una radio at86rf212, sensores capaces de recoger la luminosidad, la temperatura, picos magnéticos, y la temperatura y el voltaje de la fuente de energía. El núcleo de la mota, formado por la radio y el microcontrolador, es proporcionado por el módulo Atmel ATZB-900-B0. Se muestra una imagen de la mota que se usará:

En la imagen siguiente se muestran los componentes hardware que la forman:



A continuación se describirán de forma más detallada las partes que componen los nodos o motas.

2.1.2.1. Microcontrolador

Un microcontrolador es un circuito integrado o chip que incluye en su interior las tres unidades funcionales de una computadora: unidad central de procesamiento, memoria y unidades de entrada/salida.

Sus principales características son el bajo costo económico y consumo de energía. Un microcontrolador se diferencia de una CPU normal en que con pocos chips externos de apoyo lo convertimos en una computadora en funcionamiento. Un microcontrolador normalmente incluirá un generador de reloj, memoria RAM y memoria no volátil, y para hacerlo funcionar solo necesitaremos algunos programas de control y un cristal de sincronización. En la mayoría de los casos poseen dispositivos de entrada/salida, como convertidores de analógico a digital (ADC), temporizadores, UARTs y buses de interfaz serie especializados, como I2C y CAN.

Un microcontrolador normalmente está encapsulado en un circuito integrado, con su procesador, buses, memoria, periféricos y puertos de E/S. Fuera del encapsulado se encuentran otros circuitos para completar los periféricos internos y dispositivos que pueden conectarse a los pines de E/S. También se conectarán a los pines del encapsulado la alimentación, masa, circuito complementario del oscilador y otros circuitos necesarios para que el microcontrolador pueda trabajar.

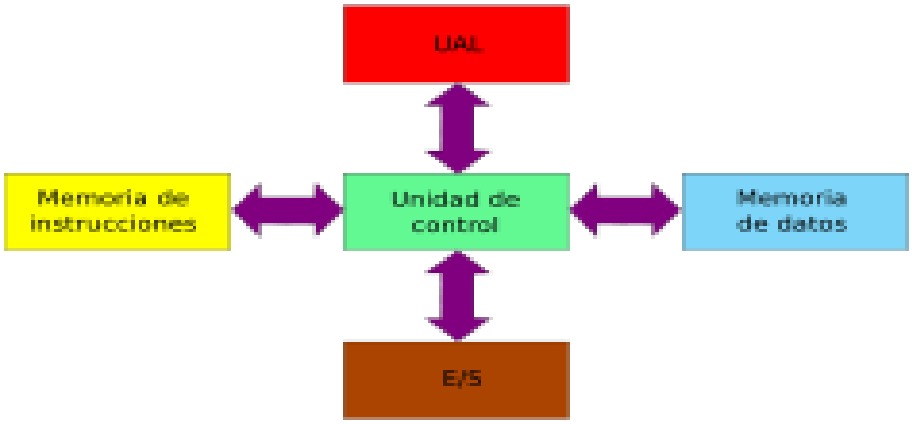
El microcontrolador que usaremos del fabricante ATMEL, modelo atmega1281, y es un microcontrolador CMOS de 8 bits de bajo consumo. Forma parte de una familia de microcontroladores RISC de Atmel llamada AVR de arquitectura Harvard.

La arquitectura Harvard se refiere a la arquitectura de computadores que utilizan dispositivos de almacenamiento físicamente separados para las instrucciones y para los datos. Cada tipo de memoria tiene un bus de datos, uno de direcciones y uno de control.

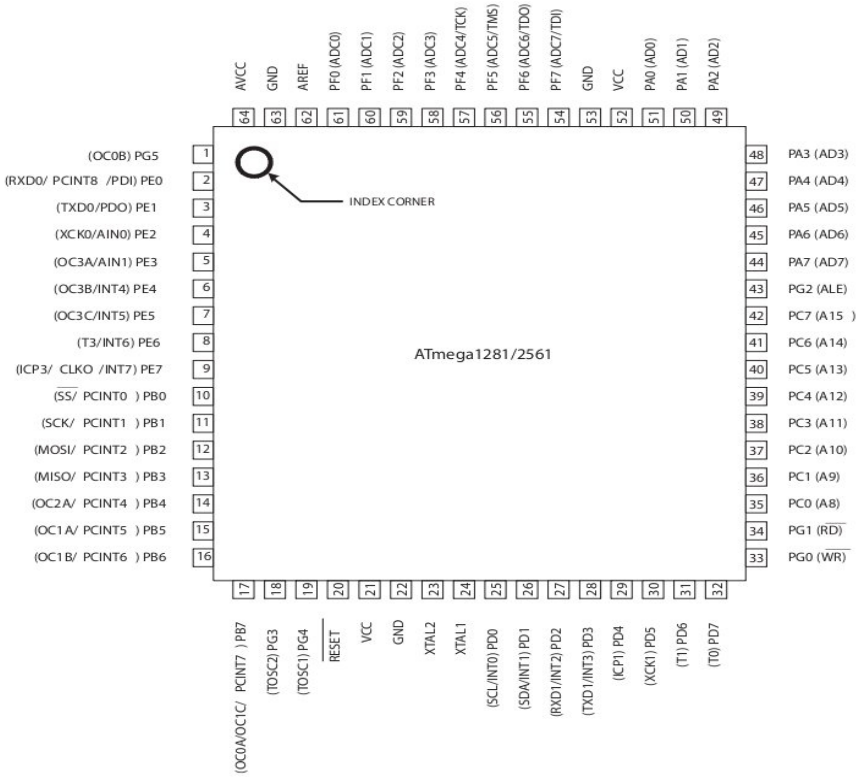
La ventaja fundamental de esta arquitectura es que permite adecuar el tamaño de los buses a las características de cada tipo de memoria; el procesador puede acceder a cada una de ellas de forma simultanea, lo que se traduce en un aumento significativo de la velocidad de procesamiento, los sistemas con esta arquitectura pueden ser dos veces más rápidos que sistemas con la arquitectura Von Neumann. Es utilizada por supercomputadores, microcontroladores y sistemas integrados en general.

La desventaja es que consume muchas lineas de E/S del procesador, por lo que solo se utiliza en supercomputadores o en microcontroladores donde la memoria de datos y programas comparten el mismo encapsulado que el procesador donde este inconveniente deja de ser un problema serio.

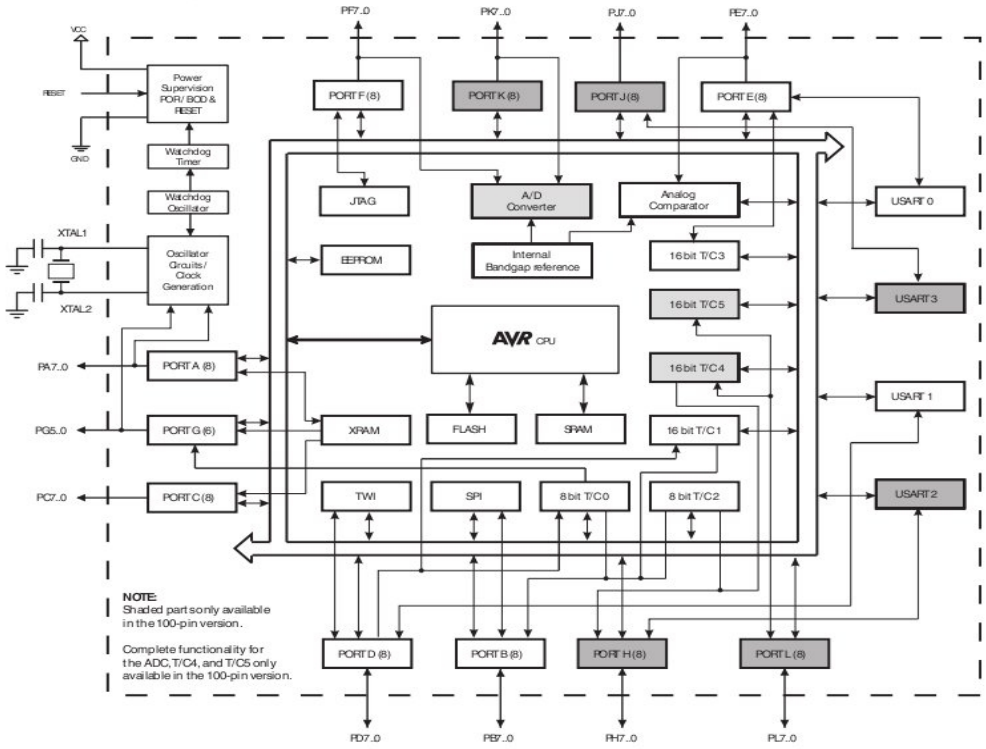
Se muestra una imagen del esquema de una arquitectura Harvard:



A continuación vemos el diagrama de pines del atmega1281:



Se muestra el diagrama de bloques que componen el microcontrolador atmega1281:



El Atmega1281 proporciona las siguientes características:

- Velocidad de procesamiento: 0 – 8 Mhz 2.7V – 5.5V, 0 – 16 Mhz 4.5 – 5.5V
- Bajo consumo de energía:
  - Modo activo: 1 Mhz, 1.8V: 500uA
  - Modo bajo consumo: 0.1 uA a 1.8V
- Flash: 128 KB
- EEPROM: 4 KB
- RAM: 8 KB
- I/O de propósito general: 54
- Canales ADC: 8

El microcontrolador posee un conversor analógico-digital (ADC) que permite convertir una entrada analógica de voltaje en un valor binario. Se utiliza para muestrear la señal analógica, continua en el tiempo, obteniendo una señal digital a la salida del mismo.

Esto nos resultará útil para obtener la temperatura ambiental del voltaje que proporciona el sensor de temperatura. Obtendremos counts de temperatura que posteriormente tendremos que aplicar una fórmula de conversión para obtener la temperatura.

Para una información más detallada sobre las características del microcontrolador Atmega1281 se puede consultar su hoja de especificaciones indicada en el apartado de bibliografía de este documento.

2.1.2.2. Radio at86rf212

El AT86RF212 es un transceptor de 700/800/900 MHz de bajo consumo y bajo voltaje especialmente diseñado para el estandar IEEE 802.15.4, ZigBee, 6LoWPAN, y aplicaciones de alta tasa ISM. Para las bandas por debajo de 1 Ghz soporta bajas tasas de transferencia, de 20 a 40 kbit/s, del estandar 802.15.4-2003/2006 y proporciona opcionales tasas de transferencia de datos, 100 y 200 kbits/s.

Proporciona en un solo chip una interfaz de radio entre la antena y el microcontrolador. Comprende parte de la radio analógica, la modulación y demodulación digital, incluyendo el tiempo y la sincronización de frecuencia, así como almacenamiento temporal de datos. Un solo byte buffer de 128 bytes almacena datos recibidos o a transmitir. El número de componentes externos necesarios es minimizado.

Sus características principales son las siguientes:

Fully Integrated 700/800/900 MHz-Band Transceiver
- Chinese WPAN Band from 779 to 787 MHz
- European SRD Band from 863 to 870 MHz
- North American ISM Band from 902 to 928 MHz

Direct Sequence Spread Spectrum with Different Modulation Schemes and Data Rates
- BPSK with 20 and 40 kbit/s, compliant to IEEE 802.15.4-2003/2006
- O-QPSK with 100 and 250 kbit/s, compliant to IEEE 802.15.4-2006
- O-QPSK with 250 kbit/s, compliant to IEEE 802.15.4c-2009
- O-QPSK with 200, 400, 500, and 1000 kbit/s PSDU Data Rate

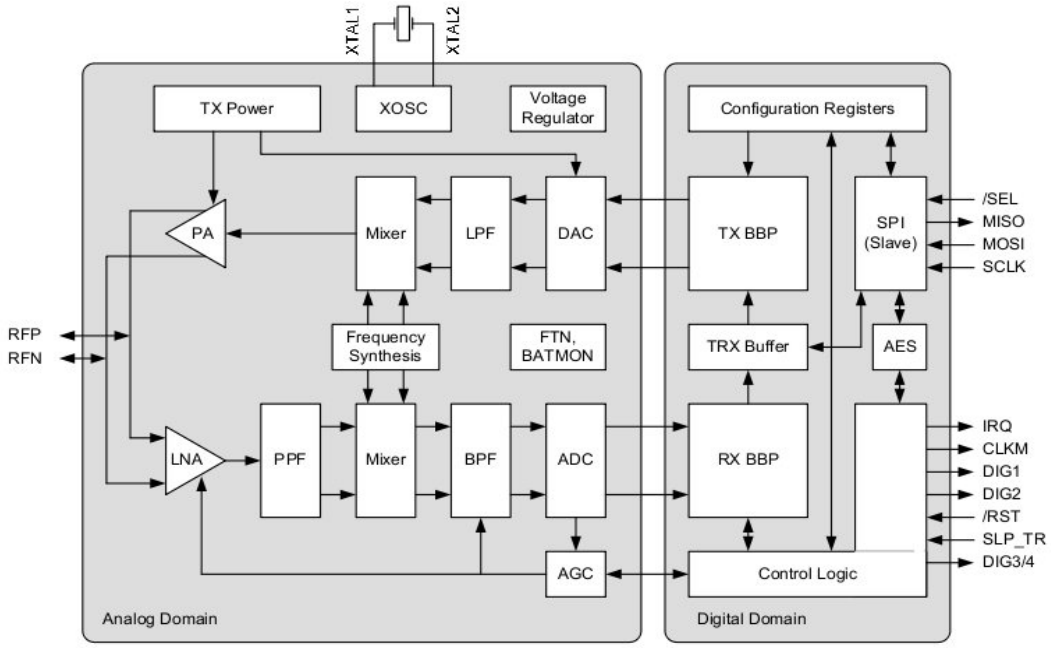
Receiver Sensitivity up to -110 dBm
- Programmable TX Output Power up to +10 dBm

Low Power Supply Voltage from 1.8 V to 3.6 V
- Internal Voltage Regulators and Battery Monitor

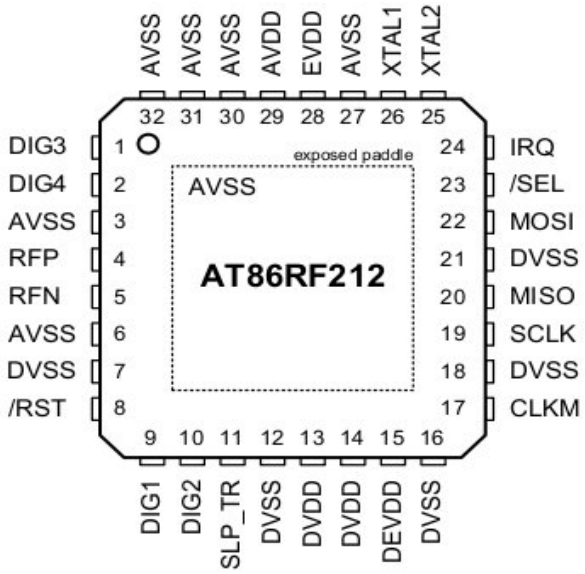
Low Current Consumption
- SLEEP = 0.2 µA
- TRX_OFF = 0.4 mA
- RX_ON = 9.2 mA
- BUSY_TX = 17 mA at P TX = 5 dBm

Radio Transceiver Features
- Adjustable Receiver Sensitivity
- Integrated TX/RX Switch, LNA, and PLL Loop Filter
- Fast Settling PLL Supporting Frequency Hopping
- Automatic VCO and Filter Calibration
- Integrated 16 MHz Crystal Oscillator
- 128 byte FIFO for Transmit/Receive

Se muestra a continuación un diagrama de bloques del transceptor T86RF212:



En la siguiente imagen vemos la correspondencia de pines del AT86RF212:



Una de las características de este chip es que posee un monitor interno de las baterías. Para obtener el voltaje es necesario aplicar una fórmula al valor recogido, se muestra a continuación:

Vbat = 1333/333 * Vleido

donde Vleido = 2.56/1024*countsADC

Esto se puede aproximar como Vbat = countsADC/100

Si se desea una información más detallada se puede consultar la hoja de especificaciones indicada en la sección de bibliografía de este documento.

2.1.2.3. ZigBit ATZB-900-B0

ZigBit 900 contiene un microcontrolador Atmel Atmega1281V y un transceptor AT86RF212 RF. El módulo posee 128 Kbytes de memoria flash y 8 Kbytes de RAM.

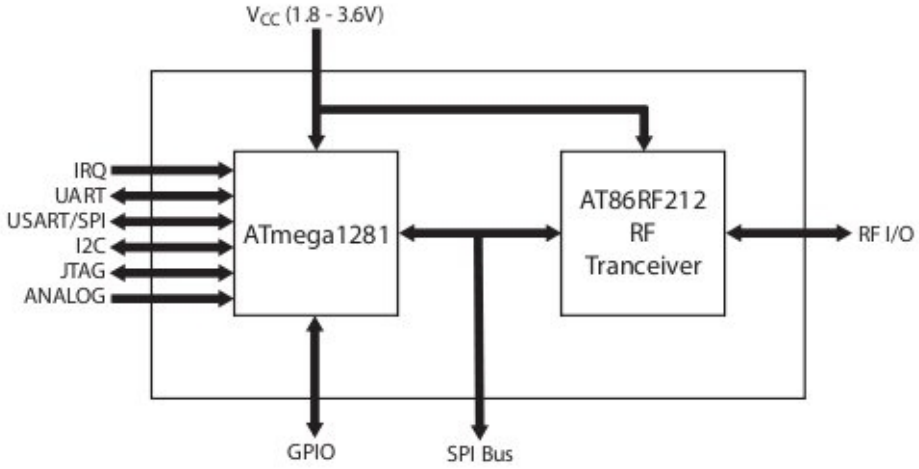
Es compatible con el estandar IEEE 802.15.4/ZigBee y tiene las siguientes características:

- Ultra compact size (18.8 x 13.5 mm)
- High RX sensitivity (-110 dBm)
- Outperforming link budget (120 dB)
- Up to 11 dBm output power
- Very low power consumption (< 6 µA in Sleep mode)
- Ample memory resources (128K bytes of flash memory, 8K bytes RAM, 4K bytes EEPROM)
- Wide range of interfaces (both analog and digital):
  - 9 spare GPIO, 2 spare IRQ lines
  - 4 ADC lines + 1 line for supply voltage control (up to 9 lines with JTAG disabled)
  - UART with CTS/RTS control
  - USART
  - I2C
  - SPI
  - 1-Wire
  - Up to 30 lines configurable as GPIO
- Capability to write own MAC address into the EEPROM
- Optional antenna reference designs
- IEEE 802.15.4 compliant transceiver
- 868 / 915 MHz band
- 784 MHz Chinese band
- BitCloud embedded software, including serial bootloader and AT command set

Ofrece los siguientes beneficios:

- Más de 6 kilómetros de alcance al aire libre con linea de visión directa
- Ocupa poco espacio físico
- Larga duración de baterías
- Capacidad de redes en malla
- Fácil de usar y bajo coste del kit de evaluación
- Una sola fuente de soporte para hardware y software

Se muestra un esquema de los componentes:



2.1.2.4. Sensores

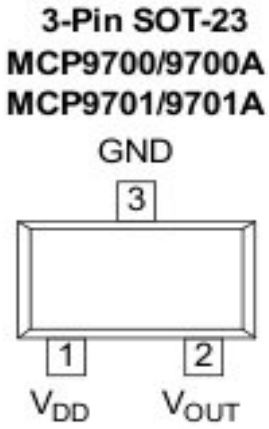
Los sensores que tenemos disponibles están integrados en las motas COU900 y nos permitirán recoger información del entorno. Son los siguientes:

- **Sensor de temperatura MCP9700A**

El MCP9700A es un sensor analógico de temperatura que convierte la temperatura en un voltaje analógico con una exactitud de +-2 ºC de 0 ºC a +70ºC y un consumo de 6µ A. Puede medir un rango de temperatura desde -40 ºC hasta +150 ºC.

No requiere un circuito adicional. El voltaje de salida (Vout) puede ser conectado directamente a la entrada del conversor analógico digital (ADC).

Proporciona una solución de bajo coste para aplicaciones que requieran medidas de temperatura de relativos cambios.



El voltaje de salida será desde 100mV a 1.75V. Para calcular los grados centígrados a los que equivale este voltaje es necesario aplicar la siguiente fórmula:

$$temp\ (ºC) = (VoltajeLeido\ (mV) - 500\ (mV))/10\ mV$$

Donde VoltajeLeido es igual a:

$$VoltajeLeido = countsADC * Vref / 2^{10}$$

Vref = 2.56V.

Para una información más detallada puede consultarse la hoja de especificaciones del sensor indicada en el apartado bibliografía en este documento.

- **Sensor de luminosidad PDV-P9003-1**

Se trata de un sensor de bajo coste integrado en la mota que posee fotocélulas fotoconductivas sensibles a la luz de 400 a 700 nm.

La hoja de especificaciones viene indicada en el apartado bibliografía de este documento.

- **Sensor de magnetismo BU52001gul-e**

Este sensor detecta si hay un material magnético cerca. Solo tiene dos estados posibles, la detección de un metal o la no detección de un metal. Podría usarse para contar los dientes de una rueda dentada o para la detección de la apertura de una puerta.

Se puede consultar la hoja de especificaciones en el apartado bibliografía.

2.1.2.5. Fuente de alimentación

La mota cuenta como fuente de alimentación con dos pilas de tipo R6(AA) de 1.5 voltios. Dado que se trata de una fuente de alimentación limitada es importante procurar optimizar el consumo de energía para extender lo máximo posible su autonomía.

También se cuenta con 3 leds integrados en el circuito de la mota que podremos usar para distinguir entre diferentes tipos de actividad en los nodos.

2.1.3. TinyOS

TinyOS es un sistema operativo open source diseñado para redes de sensores inalámbricos escrito en el lenguaje de programación NesC, del cual hablaremos más adelante. De entre sus características, destaca su diseño orientado a eventos para redes de nodos de sensores que tienen unos recursos muy limitados. Está desarrollado por un consorcio liderado por la Universidad de California en Berkley en cooperación con Intel

Research.

Está diseñado para realizar operaciones de bajo consumo energético en dispositivos muy limitados, como por ejemplo microcontroladores con uno pocos kilobytes de RAM y espacio para código. Posee sistemas y mecanismos bastante severos para el ahorro de energía. Proporciona un conjunto de servicios y abstracciones útiles para comunicaciones, almacenamiento, contadores o sensores.

Se ha liberado bajo licencia BSD la cual no obliga a distribuir el código fuente, al contrario que la licencia GPL. Tiene una arquitectura basada en componentes que permite el desarrollo rápido y minimiza el código requerido.

Soporta numerosas plataformas de nodos de sensores, entre las que se encuentra la que utilizaremos en este proyecto.

TinyOS gestiona básicamente dos tipos de procesos:

- **Los eventos:** son procesos ligeros que responden a interrupciones de hardware, realizan pocas cantidades de cálculos. Tienen prioridad de ejecución y pueden interrumpir las tareas en ejecución. Se ejecutan hasta que se completan.
- **Las tareas:** no son críticas temporalmente y pueden realizar más cálculos que los eventos. No se pueden interrumpir entre ellas y el planificador las ejecuta por orden de llegada, cuando termina una comienza la siguiente. Para asegurar la baja latencia de ejecución, las tareas deben de ser cortas, largas operaciones deben ser divididas en varias tareas.

El núcleo de TinyOS es completamente no bloqueante, las operaciones que toman un tiempo largo son divididas en fases. En la primera fase comienza la operación y retorna inmediatamente en lugar de esperar hasta completarse. Los comandos, comands, son normalmente peticiones para ejecutar una operación. Si la operación está dividida en fase retorna inmediatamente y la finalización será señalada mediante un evento en una segunda fase. Las operaciones no divididas en fase no tienen eventos de finalización.

Ejemplos de división de fase pueden ser al enviar mensajes (send/sendDone), comenzar contadores (start/fired), consultar sensores (read/readDone) y almacenar datos (write/writeDone).

Este diseño permite la ejecución rápida de eventos sin necesidad de tener que esperar a terminar una tarea.

2.1.4. NesC

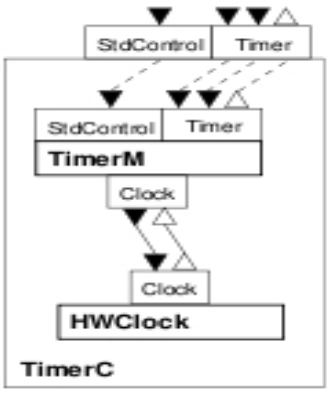
NesC es una extensión del lenguaje C diseñada para incorporar los conceptos de estructura y modelo de ejecución de TinyOS. Produce código eficiente para ser usado en microcontroladores de redes de sensores. Combina un modelo orientado a objetos, implementación de interfaces, con un modelo de eventos.

En las librerías de TinyOS podemos encontrar multitud de módulos, configuraciones e interfaces ya implementadas para desarrollar aplicaciones.

Los conceptos básicos de NesC son los siguientes:

- **Separación entre construcción y composición.** Los programas se construyen a partir de componentes, los cuales son ensamblados o conectados (wired) para formar programas. Los componentes definen dos ámbitos, uno para la especificación que contienen el nombre de sus interfaces y uno para su implementación.
- **Especificación de componentes.** Las interfaces pueden ser proporcionadas o usadas por un componente. Pueden proporcionar interfaces implementando una funcionalidad para sus usuarios, las interfaces usadas representan las funcionalidades que el componente necesita para hacer su trabajo y son implementadas por otros componentes.
- **Las interfaces son bidireccionales.** Especifican un conjunto de funciones a ser implementadas por el componente que proporcione la interface (commands) y eventos a ser implementadas por el usuario de la interface (events). Una interface es una interacción entre componentes en ambos sentidos. Cuando se ejecuta un comando (call send(...)) en un componente se retorna imediatamente y es necesario implementar el evento que se lanza cuando el comando a finalizado (event sendDone(){...}).
- **Los componentes están estáticamente linkados entre ellos a través de sus interfaces.** Esto incrementa la eficiencia en tiempo de ejecución, hace el diseño más robusto y permite un mejor análisis estático de programas.
- **NesC posee herramientas para la optimización** de generación de códigos.

La imagen siguiente muestra la estructura de un componente:



Se describen los ficheros que pueden formar parte de un componente:

- **Configuración e implementación:** conectan las interfaces utilizadas con las implementadas por los componentes.
- **Módulos:** se indican las interfaces que se usarán y se proporcionarán y se programan las acciones que realizará el componente.
- **Librerías:** archivos de cabecera con extensión .h. Contiene subrutinas, variables u otros identificadores.
- **Interfaces:** se definen los comandos que el componente que proporcione esta

interfaz tendrá que implementar y los eventos que el componente que use esta interfaz tendrá que implementar.

- **Makefile:** en este archivo se definen las reglas de compilación.

Los archivos básicos son el de configuración y el módulo. Un ejemplo de un componente sencillo podría ser el siguiente:

```
configuration BlinkAppC {
}

implementation {
  components MainC, BlinkC, LedsC;
  components new TimerMilliC() as Timer0;

  BlinkC -> MainC.Boot;

  BlinkC.Timer0 -> Timer0;
  BlinkC.Leds -> LedsC;
}
```

En el archivo de configuración conecta la interface del contador y led usado con la implementación de la interface por TimerMillic() y LedsC respectivamente.

```
#include "Timer.h"

module BlinkC @safe()
{
  uses interface Timer<TMilli> as Timer0;
  uses interface Leds;
  uses interface Boot;
}
implementation
{
  event void Boot.booted()
  {
    call Timer0.startPeriodic( 250 );
  }

  event void Timer0.fired()
  {
    dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
    call Leds.led0Toggle();
  }
}
```

En el archivo de módulo se implementan los eventos recibidos de las interfaces usadas, cuando se lanza un contador y al arranque, y se realiza una llamada a una función para cambiar de estado el led cero. Si se proporcionara alguna interfaz habría que implementar los comandos en este archivo. Este componente simplemente enciende un led cada vez que se lanza el contador.

2.1.5. Protocolo de comunicación ZigBee

ZigBee es una especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica de bajo coste y consumo. Está basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal (WPAN). Está pensado para aplicaciones que requieran una baja tasa de transferencia de datos y un mínimo consumo de energía.

ZigBee se concibió en 1998 cuando se vio claramente que WIFI y Bluetooth no serían soluciones válidas para todos los contextos. El ámbito en el que se piensa que tendrá más éxito es en la domótica debido a sus características principales, que son;

- Posee un bajo consumo energético
- Su topología de red en malla, lo que significa que se establecen conexiones entre nodos permitiendo la ampliación de la información en todo el área de despliegue.
- La posibilidad de fabricar nodos con pocos componentes electrónicos.

Utiliza la banda ISM para usos industriales, científicos y médicos; 868 MHz en Europa, 915 en EEUU y 2,4 Ghz en todo el mundo.

La necesidad de hardware para construir un nodo ZigBee es mucho menor que para Bluetooth o Wi-Fi, en torno a un 10% si se trata de un nodo ZigBee complejo y de un 2% si es un nodo sencillo. El código fuente necesario sería de un 50% del tamaño de Bluetooth.

El bajo consumo energético es otra característica importante de este protocolo, llegando hasta los 5 años de autonomía antes de proceder a un cambio de baterías en un nodo.

Permite transferencias de datos hasta 250 kbps y un rango de alcance de 10 m a 75 m.

En una red ZigBee puede haber tres tipos de dispositivos según el papel que desempeñe:

- **Coordinador ZigBee** (ZigBee Coordinator, ZC). Este es el tipo más completo y solo debe de haber uno por red. Es el encargado de controlar los caminos que se deben seguir los dispositivos para conectarse entre ellos.
- **Router ZigBee** (ZigBee Router ZR). Este tipo es el encargado de interconectar dispositivos separados en la topología de la red. También permite la ejecución de código de usuario.
- **Dispositivo final** (ZigBee End Device, ZED). Este tipo puede comunicarse con un nodo router o coordinador pero con otro nodo. Esto hace que pueda estar dormido la mayor parte del tiempo ahorrando energía y que necesite menos memoria, lo que hará que normalmente sea más barato.

Otra clasificación se puede realizar en base a funcionalidades y tenemos:

- **Dispositivo de funcionalidad completa (FFD) o nodo activo:** puede recibir mensajes en formato 802.15.4 y funcionar como Coordinador o Router ZigBee, o en dispositivos de red que actúen de interface con los usuarios.

- **Dispositivo de funcionalidad reducida (RFD) o nodo pasivo:** este tipo de nodos posee funcionalidades limitadas con el objetivo de conseguir un bajo coste y simplicidad.

Las topologías posibles en ZigBee son las siguientes:

- Topología en estrella: el coordinador se sitúa en el centro.
- Topología en árbol: el coordinador será la raíz del árbol.
- Topología en malla: al menos uno de los nodos tendrá más de dos conexiones.

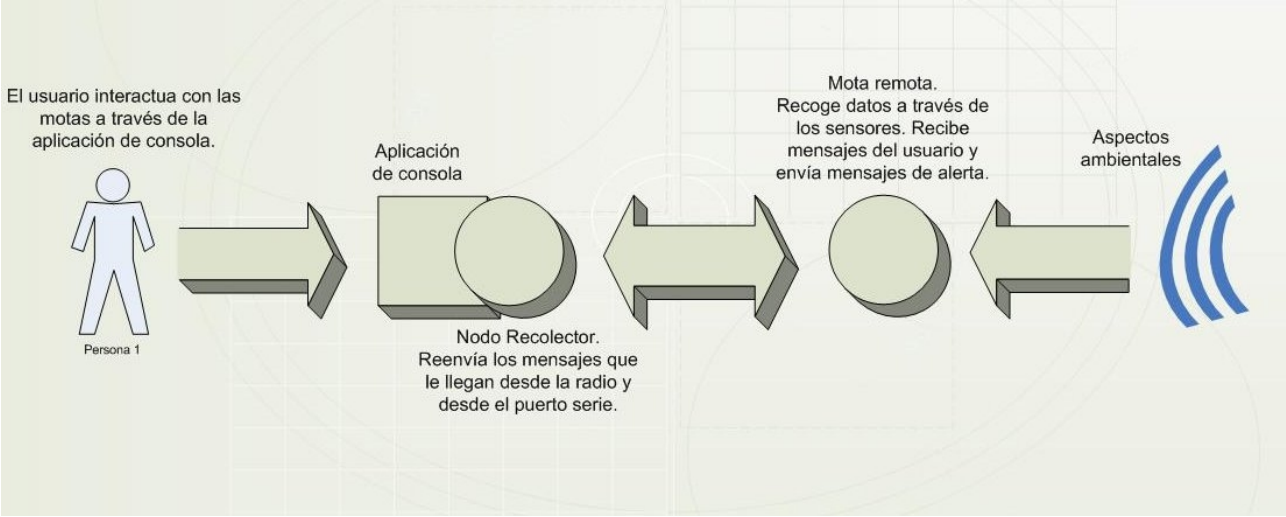
La topología en malla establece comunicaciones ad-hoc entre nodos. Esta topología permite que el coordinador restablezca nuevos caminos entre nodos si algún nodo de un camino fallara.

3. Desarrollo

En este apartado se describirá la aplicación desarrollada. Se comenzará describiendo la aplicación a nivel funcional, los diferentes bloques de los que está compuesta y cual es la función de cada uno. Posteriormente se entrará más en detalle sobre su diseño, y se terminará mencionando algunos desarrollos que podrían realizarse para mejorar la aplicación partiendo del producto obtenido.

3.1. Descripción funcional

A grandes rasgos, los bloques que componen el producto desarrollado se pueden ver en el diagrama siguiente:



3.1.1. Aplicación de consola RemoteCon

El usuario utiliza una aplicación desarrollada que se ejecuta en un terminal para interactuar con las motas, a la que se le ha llamado RemoteCon. Ésta, por defecto, se conecta al puerto serial@/dev/ttyUSB0 para establecer comunicación con el nodo recolector o sink node y que este nos reenvíe a la red de sensores. Se puede indicar otro puerto mediante las opciones disponibles. Si quisiéramos conectarnos desde otra máquina se podría utilizar la aplicación SerialForwarder. Ésta se conectaría por usb con el nodo recolector, y nos conectaríamos remotamente desde otra máquina ejecutando la aplicación RemoteCon y conectándola al SerialForwarder por la IP y puerto que escucha.

RemoteCon permite mostrar por pantalla los mensajes de alerta enviados por las motas cuando recogen valores de los sensores fuera de los rangos establecidos. También permite solicitar a una mota que envíe un mensaje, este podrá ser uno se los siguientes:

- BatMsg: mensaje que contiene los campos identificador de mota desde el que se envía, tensión de las baterías de la mota y número de mensaje. La tensión está expresada en voltios. También muestra la fecha y hora con precisión de milisegundos del momento en que ha llegado el mensaje.

```
13/01/2011 20:41:16:859
        Mensaje recibido de tipo BatMsg:
                MoteId: 1
                Nivel de baterías: 2.54 voltios.
                Número de paquete: 172.
```

- TempMsg: mensaje que contiene los campos identificador de la mota, temperatura recogida por el sensor y número de mensaje. La temperatura se expresa en grados centígrados.

```
13/01/2011 20:42:04:256
        Mensaje recibido de tipo TempMsg:
                MoteId: 1
                Temperatura: 25.25 ºC.
                Número de paquete: 269.
```

- InfoMsg: contiene los campos identificador de mota, valor de temperatura máxima y mínima, valor de batería mínima y número de mensaje. Este tipo de mensaje también contiene dos campos más, opción de mensaje y opción de valor, se utilizan para solicitar a una mota remota una acción a realizar pero no se mostrarán por pantalla al solicitar un mensaje de tipo InfoMsg pues su valor no es relevante.

```
13/01/2011 20:41:57:865
        Mensaje recibido de tipo InfoMsg:
                MoteId: 1
                MAXTEMP: 24.0 ºC.
                MINTEMP: 20.0 ºC.
                MINBAT: 2.0 voltios.
                Número de paquete: 3.
```

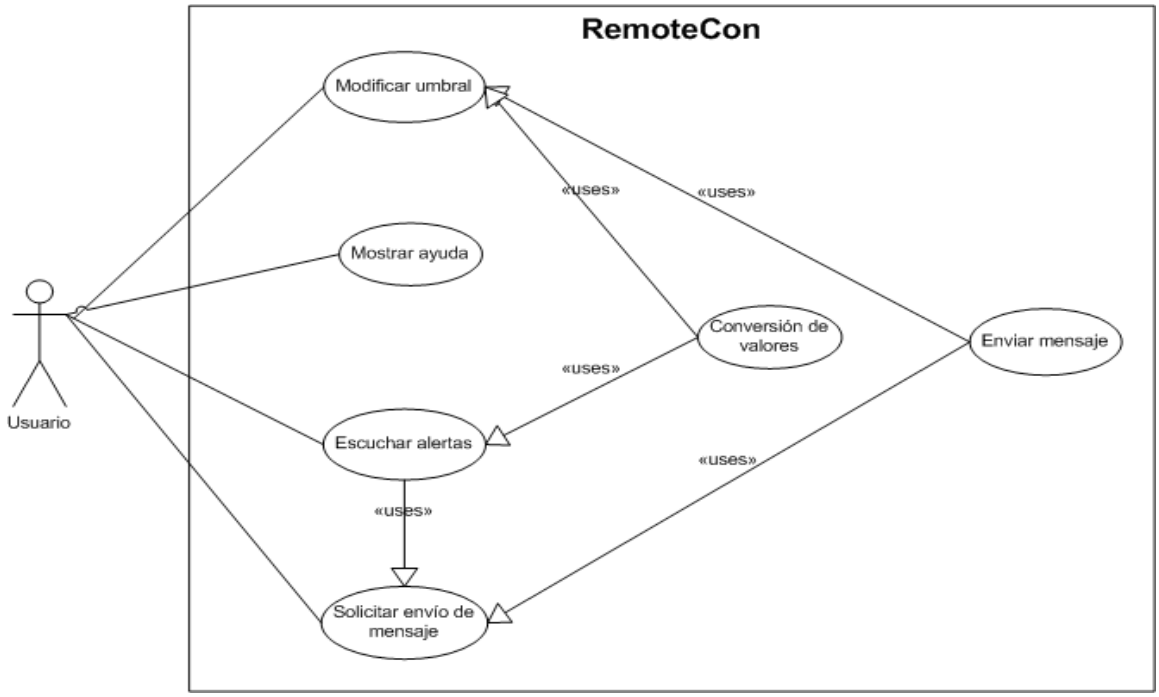
Al recibir un mensaje de los tipo indicado podemos saber de cual se trata por la cabecera, de manera que ejecutamos un método distinto para mostrar el mensaje recibido en función del tipo que se trate.

Otra funcionalidad que ofrece esta aplicación es la posibilidad de modificar los valores umbrales establecidos:

- Tensión mínima de baterías indicado en voltios.
- Temperatura mínima, indicada en grados centígrados.
- Temperatura máxima, indicada en grados centígrados.

Las motas enviarán los mensajes de alerta en función de si el valor recogido por los sensores está dentro de estos rangos.

Se muestra un diagrama de casos de uso donde se pueden ver las funcionalidades descritas de la aplicación RemoteCon:



Se muestra la ayuda de la aplicación:

```
  ____                      _       ____
 |  _ \ ___ _ __ ___   ___ | |_ ___ / ___|___  _ __
 | |_) / _ \ '_ ` _ \ / _ \| __/ _ \ |   / _ \| '_ \
 |  _ <  __/ | | | | | (_) | ||  __/ |__| (_) | | | |
 |_| \_\___|_| |_| |_|\___/ \__\___|\____\___/|_| |_|

usage: java RemoteCon [-comm device] [-h] [-S maxTemp] [-s minTemp] [-b minBat] [-I] [-B] [-T] [-i moteId]

RemoteCon version 2.3

Comunicación remota con las motas a través del puerto que se le indique. Por defecto se
asigna el puerto serial@/dev/ttyUSB0:19200. Si se desea otro puerto es necesario asignarlo
explícitamente con el parámetro -comm <source>.
```

```
La aplicación ofrece tres funcionalidades principales:

  1. Recepción de mensajes de alerta de las motas:

    Ejemplo: java RemoteCon -comm serial@/dev/ttyUSB0:19200

  2. Modificación de valores umbrales en motas:

    Ejemplo: java RemoteCon -S 30 -s 20 -b 2.1 -i 1

  3. Solicitud de envío de mensaje de los siguientes tipos a una mota:

    - InfoMsg: valores umbrales actualmente asignados
    - BatMsg: valor de temperatura ambiente recogido por el sensor
    - TempMsg: estado actual de baterías

    Ejemplo: java RemoteCon -I -i 1

    Las opciones de tipo 2 y 3 se pueden indicar en la misma linea de ordenes. Es
    necesario indicar el identificador de mota para cualquiera de las dos opciones.

    Ejemplo: java RemoteCon -I -i 1 -S 30 -s 20 -b 2.1

  -h, --help              : Muestra la ayuda
  -S, --setMaxTemp <TEMP> : Asigna el valor de temperatura máxima
  -s, --setMinTemp <TEMP> : Asigna el valor de temperatura mínima
  -b, --setMinBat <BAT>   : Asigna el valor de tensión mínima de las baterías
  -I, --requestInfoMsg    : Solicita un mensaje de tipo InfoMsg a una mota
  -B, --requestBatMsg     : Solicita un mensaje de tipo BatMsg a una mota
  -T, --requestTempMsg    : Solicita un mensaje de tipo TempMsg a una mota
  -i, --moteId <MOTEID>   : Se indica a que mota nos referimos
  -comm <source>          : Se indica el dispositivo con el que establecemos comunicación

MOTEID es un valor entero positivo. TEMP y BAT son valores en coma flotante con 2 y 1
cifras decimales respectivamente. TEMP tiene un margen de error por redondeo de 0.25 ºC.
Se recomienda utilizar múltiplos de este número para las asignaciones de valores de temperatura.

La aplicación puede emitir mensajes de información y de error durante su ejecución, serán
lineas que comiencen por "INFO:" o "ERROR:" repectivamente.
```

3.1.2. Nodo recolector o sink node

Para conectar con la red de sensores se utiliza una mota conectada por usb a la máquina. Esta mota tiene instalada la aplicación BaseStation que reenvía los paquetes que le llegan por la radio por el puerto serie, y los que le llegan por serie los reenvía por usb.

De esa manera cuando recibe un mensaje de alerta de una mota por la radio, lo reenvía por el usb, y si tenemos la aplicación RemoteCon escuchando mostrará por pantalla el mensaje recibido.

De la misma manera si enviamos una solicitud de envío de mensaje o cambio de valores umbrales, enviaremos el mensaje de tipo InfoMsg al puerto usb, el nodo recolector lo reenviará por la radio y le llegará a la mota destino.

La aplicación BaseStation iluminará el led ambar cada vez que reenvia un mensaje y el led rojo si se produce un error.

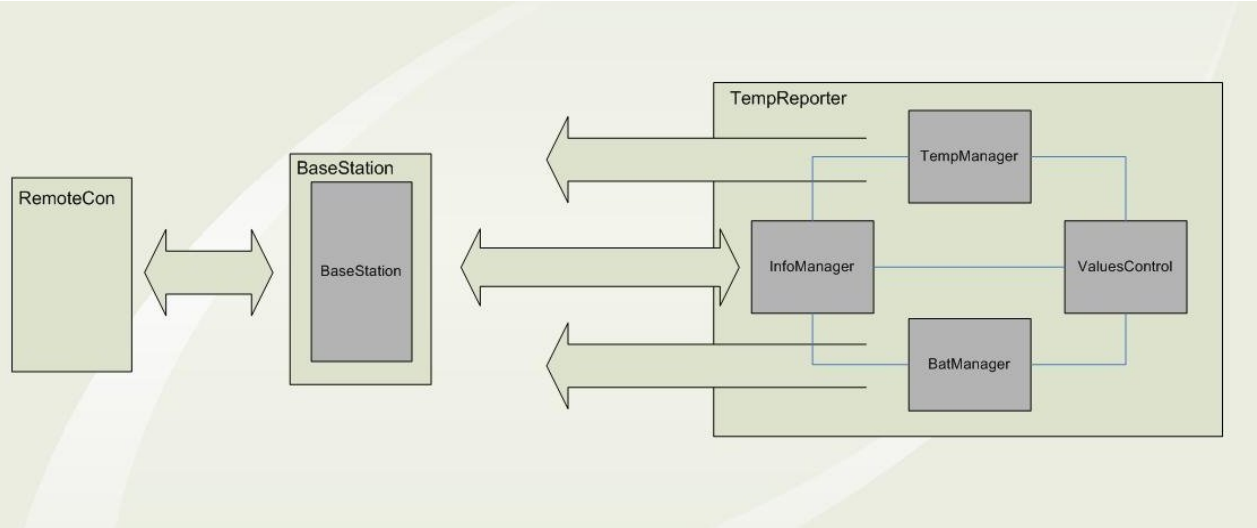
3.1.3. TempReporter

La aplicación TempReporter irá cargada en las motas que forman la red de sensores del área a monitorizar. Está compuesta por varios componentes, los cuales se conectan entre ellos mediante la implementación y uso de interfaces, permitiendo usar la implementación de un comando en varios componentes o tener varias implementaciones distintas de un mismo comando. Enviará mensajes por broadcast a la red informando de alertas o de los valores de los umbrales establecidos.

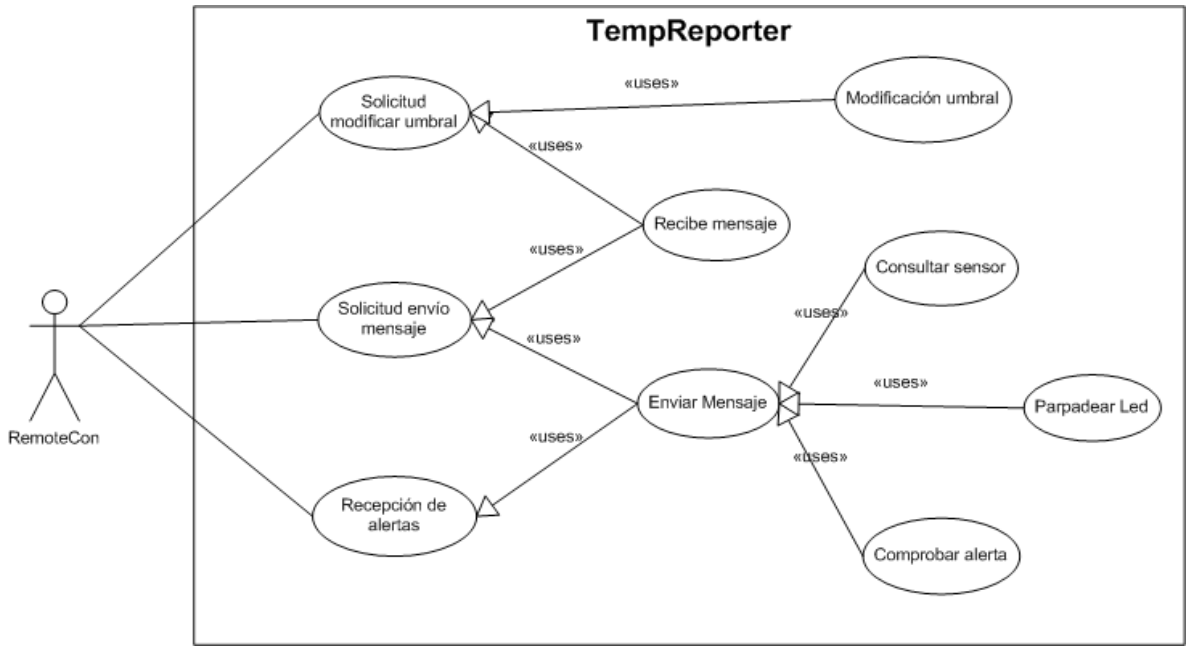
Se describen los componentes que la componen:

- **InfoManager:** recibe los mensajes de solicitud de envío de mensajes o modificación de valores umbrales desde la aplicación RemoteCon. Puede solicitar el envío de un mensaje a los componentes BatManager y TempManager. También puede envíar un mensaje de tipo InfoMsg si se le solicita. Cada vez que envía un mensaje de tipo InfoMsg hace parpadear el led verde.
- **BatManager:** consulta el sensor de nivel de baterías y envía un mensaje en caso de alerta o recibir una solicitud de envío por parte de InfoManager. Cada vez que envía un mensaje hace parpadear el led ambar.
- **TempManager:** consulta el sensor de temperatura y envía un mensaje en caso de alerta o al recibir una solicitud de envío por parte de InfoManager. Cada vez que envía un mensaje hace parpadear el led rojo.
- **ValuesControl:** este componentes permite mediante comandos accesores obtener o modificar los valores umbrales de temperatura y voltaje de baterías definidos. También ofrece un comando para calcular si un valor de temperatura o nivel de baterías es una alerta o no.

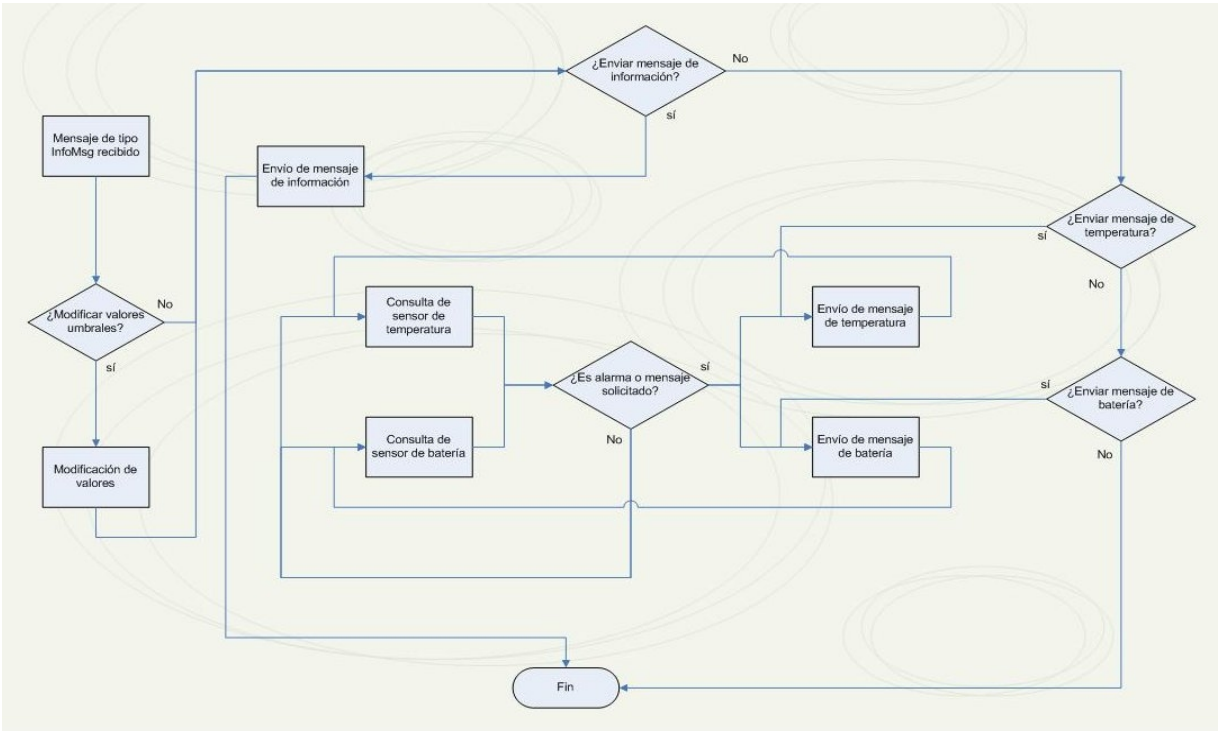
En la imagen siguiente se puede ver el sistema a grandes rasgos y la conexión entre componentes.



Se muestra un diagrama de casos de uso con las funcionalidades descritas de la aplicación TempReporter:



A continuación se puede ver un diagrama de flujo de la aplicación TempReporter:



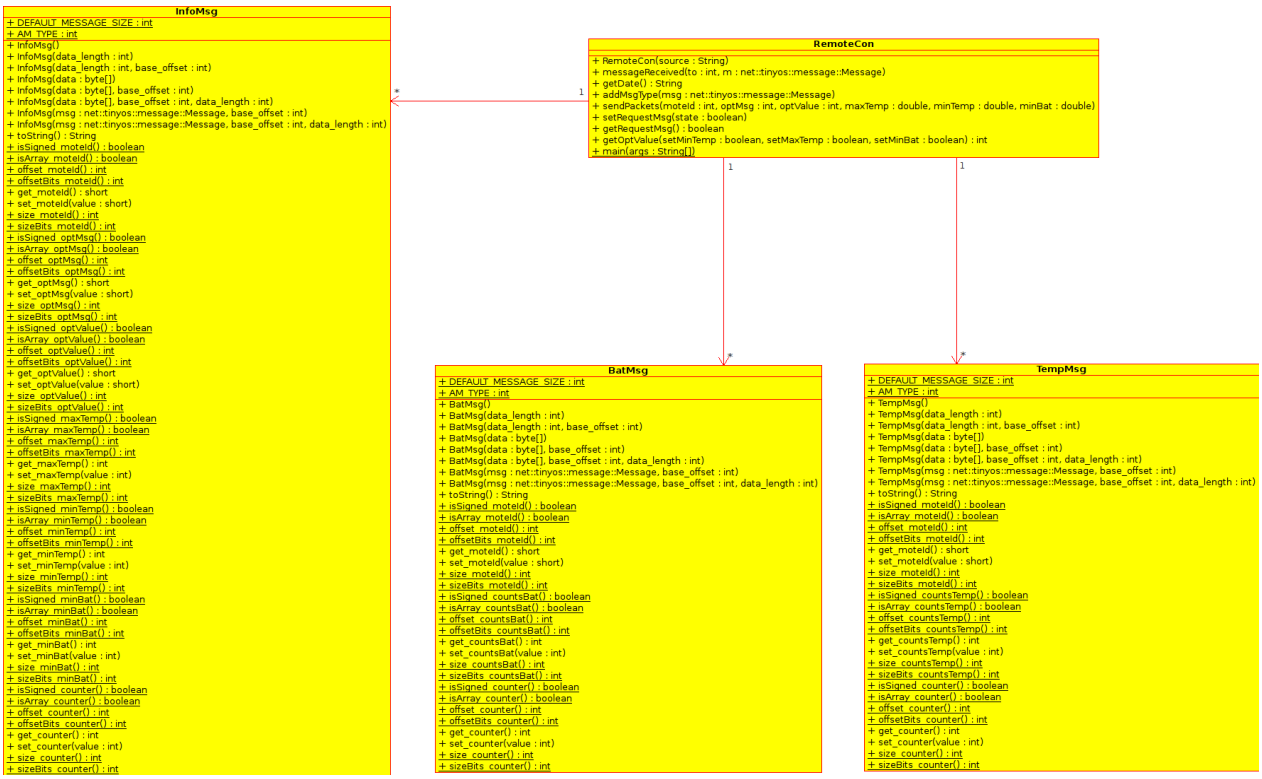
En el siguiente apartado se describirá más detalladamente la interconexión entre estos componentes y el diseño de la aplicación.

3.2. Descripción detallada

3.2.1. Aplicación RemoteCon

Para acceder a los campos de los mensajes recibidos en la aplicación de consola RemoteCon y para facilitar la generación de mensajes a enviar se ha hecho uso de las clases generadas por la aplicación “mig” para cada tipo de mensaje definido en la aplicación TempReporter. La generación de las clases por “mig” se realiza de manera automática indicándolo en el archivo Makefile.

Se ha establecido una relación de asociación entre la clase RemoteCon y las clases generadas para cada tipo de mensaje, de manera que se crean objetos de estos tipos para manejar los mensajes. El diagrama de clases es el siguiente:



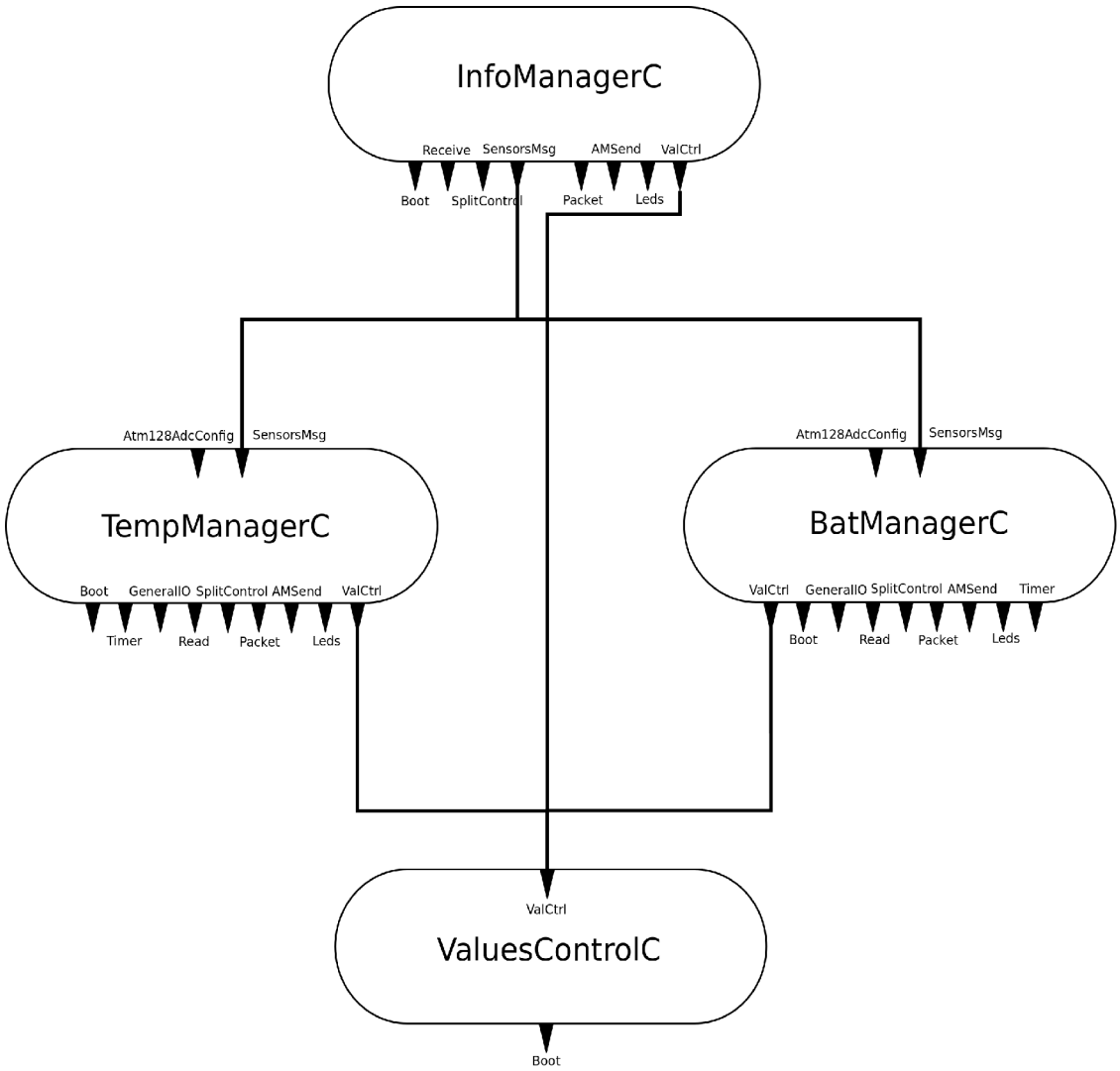
Se ha intentado evitar cálculos innecesarios en las motas para aprovechar la vida de las baterías. Es por este motivo que las conversiones de los valores recogidos por los sensores a grados centígrados o a voltios no se realizan en las motas sino en la aplicación cliente RemoteCon. En todo momento se trabaja con counts en las motas. Al asignar un valor umbral desde RemoteCon, primero se convierte el valor en grados o voltios a counts y se asigna el valor obtenido al mensaje. Cuando el mensaje llega al componente InfoManager asigna directamente los counts a los valores umbrales.

Como consecuencia de estas conversiones, se ha detectado un error de precisión. Tras realizar los cálculos de conversión, si se obtenía valores decimales en los counts, al enviarlos a la mota son descartados pues en las motas trabajamos con valores enteros.

Esto hace que si tenemos un valor de 320.9 el 0.9 se descarta. Para reducir el error de conversión se hace uso del método rint de la clase Math, este método redondea el valor a entero en la conversión de grados a counts. De esta manera, y siguiendo el ejemplo anterior, a la mota se le enviaría el valor 321 y el error se reduce de 0.9 a 0.1. Si antes contábamos con un error máximo de 0.9 counts, ahora el error máximo es de 0.5.

3.2.2. Aplicación TempReporter

Como se ha comentado en el apartado anterior, la aplicación TempReporter está compuesta por cuatro componentes. Estos componentes, y las interfaces que usan e implementan cada uno, se pueden ver en el gráfico siguiente.



Las interfaces que usa un componente se representan en el gráfico mediante triángulos fuera del rectángulo redondeado y estarán implementadas en otros componentes. Las interfaces implementadas por los componentes se representas por triángulos dentro de

los rectángulos.

Los componentes BatManager y TempManager se encargan de consultar los sensores de batería y temperatura respectivamente. Esto lo hacen a una frecuencia asignada a un temporizador en el código de la aplicación. Esta frecuencia puede ser cambiada, para así adaptarla a las necesidades de la situación. Hay que tener en cuenta que el valor asignado a este contador incidirá directamente en la autonomía de las baterías por lo que se recomienda realizar un estudio para intentar asignar el máximo valor posible con la idea de aprovechar al máximo la vida de éstas, sin que se vea comprometido el objetivo principal del sistema, reportarnos una situación crítica con suficiente antelación como para poder reaccionar a la situación.

Para optimizar el consumo de las baterías los sensores se desactivan tras su consulta cada vez que se dispara el contador. Mientras están desactivado no consumen energía. Si el tiempo asignado al contador es pequeño el ahorro de energía es mínimo, pero si se aumenta el valor de este puede suponer un ahorro considerable.

El componente BatManager juega un papel importante en el buen funcionamiento de la aplicación pues nos monitoriza el nivel de la batería, si éste es demasiado bajo podemos recoger valores no correctos de los sensores. Para que el conversor de Analógico a Digital (ADC) funcione correctamente necesita una tensión superior a la referencia (2.56V). El sensor de temperatura también necesita un voltaje mínimo de 2.3V para operar correctamente. Por esta razón se recomienda asignar un valor de 2.6V al umbral mínimo de baterías.

El componente ValuesControl implementa la interface ValCtrl. Esta interfaz define unos comandos para acceder a los valores umbrales y comprobar si el valor indicado se trata de una alarma. Este componente se ha creado para implementar una funcionalidad común a varios componentes, BatManager y TempManager comprueban si el valor recogido del sensor se trata de una alarma antes de enviar un mensaje. De esta manera si en un futuro necesitamos modificar o ampliar aspectos relacionados con los valores umbrales o alertas solo tendremos que modificar un componente.

Se ha creado otra interface llamada SensorMsg. Esta interfaz es implementada por BatManager y TempManager para enviar un mensaje en ese momento si necesidad de que se produzca una alerta. Es usada por el componente InfoManager cuando un usuario solicita el envío de un mensaje de tipo BatMsg o TempMsg. Resulta de utilidad para que cada componente la implemente de forma diferente si es necesario, ya que la interfaz define solo lo que debe hacer y no el como.

Sobre los mensajes enviados, se han definido de tres tipos diferentes, se describen los campos que contiene cada uno:

- BasMsg: mensajes para informar del estado de las baterías.
  - moteId: identificador de la mota.
  - countsBat: counts recogidos del sensor de baterias.
  - counter: contador de mensaje.
- TempMsg: mensajes para informar de la temperatura.
  - moteId: identificador de la mota.

    - countsTemp: counts recogidos del sensor de temperatura.
    - counter: contador de mensaje.
- InfoMsg: mensajes para informar de los valores umbrales asignados.
    - moteId: identificador de la mota.
    - optMsg: indica que tipo de mensaje se solicita.
    - optValue: indica que valores se desea modificar.
    - maxTemp: umbral de temperatura máxima.
    - minTemp: umbral de temperatura mínima.
    - minBat: umbral de batería mínima.
    - counter: contador de mensaje.

Se ha hecho así para evitar tener que comprobar el tipo de mensaje cuando lleguen a la aplicación cliente y a la de las motas, ya que TinyOS permite diferenciar los tipos de mensajes por su cabecera. Se ha definido un número a cada tipo:

```
AM_BATMSG       = 40
AM_TEMPMSG      = 41
AM_INFOMSG      = 42
```

A continuación se indican los comandos definidos por las interfaces creadas:

- SensorsMsg: define solamente un comando “SendMsgNow()” para solicitar a un componente que envíe un mensaje en este momento.
- ValCtrl: se usa para definir comandos accesores de los valores umbrales y comprobación de si un valor pasado se trata de una alerta.
    - Bool isTempAlarm(nx_uint16_t counts) : comprueba si el valor pasado corresponde a una alerta de temperatura. En caso afirmativo será necesario informar mediante el envío de un mensaje de tipo TempMsg.
    - bool isBatAlarm(nx_uint16_t counts) : comprueba si el valor pasado corresponde a una alerta de batería. En caso afirmativo será necesario informar mediante el envío de un mensaje de tipo BatMsg.
    - nx_uint16_t getMaxTemp() : devuelve el valor de umbral máximo de temperatura.
    - nx_uint16_t getMinTemp() : devuelve el valor del umbral mínimo de temperatura.
    - void setMinTemp(nx_uint16_t minCountsTemp) : asigna el valor pasado como parámetro al umbral mínimo de temperatura.
    - void setMaxTemp(nx_uint16_t minCountsTemp) : asigna el valor pasado como parámetro al umbral máximo de temperatura.

- nx_uint16_t getMinBat() : devuelve el umbral mínimo de batería.
- void setMinBat(nx_uint16_t minCountsBat) : asigna el valor pasado como parámetro al umbral mínimo de batería.

La interface ValCtrl es implementada por el componente ValuesControl. La usan BatManager y TempManager para consultar si los valores recogidos por los sensores se trata de alertas, también la usa el componente InfoManager para modificar los valores de los umbrales a tráves de los comandos accesores.

El componente InfoManager es el único que tiene implementado la recepción de mensajes y solo puede recibir mensajes de tipo InfoManager. Esto se usa para que el usuario pueda solicitar una acción desde la aplicación en consola. Para gestionar la opción deseada se ha reservado dos variables en el tipo de mensaje InfoManager: optMsg y optValue.

OptMsg indica que tipo de mensaje se desea que se envíe en este momento, sin necesidad de que ocurra una alerta. Las opciones posibles son:

- Envío de mensaje de tipo InfoMsg → optMsg = 1
- Envío de mensaje de tipo TempMsg → optMsg = 2
- Envío de mensaje de tipo BatMsg → optMsg = 3

Si la variable optMsg es distinta de los valores arriba indicados no enviará ningún mensajes.

La variable optValue se usa para indicar que valor de umbral se desea modificar. Las opciones posibles son las siguientes:

- Temperatura mínima → optValue = 4
- Temperatura máxima → optValue = 5
- Batería mínima → optValue = 6
- Temperatura mínima y máxima → optValue = 7
- Temperatura mínima y batería mínima → optValue = 8
- Temperatura máxima y batería mínima → optValue = 9
- Temperatura mínima, máxima y batería mínima → optValue = 10

Si la variable optValue tiene otro valor diferente a los arriba indicados no modificará ningún valor.

En el caso de solicitar la modificación de un umbral con algunas de las opciones anteriores, el valor a asignar se cogerá de los campo del mensaje de tipo InfoMsg (maxTemp, minTemp y minBat).

3.3. Desarrollos futuros

Como posibles mejoras y ampliaciones de funcionalidades de la aplicación se proponen los siguientes desarrollos:

- **Interfaz gráfica:** para que la aplicación de comunicación con las motas fuera más amigable se podría crear una interfaz gráfica que permitiera su control. Podría ser una interfaz web, una aplicación de escritorio o una de consola más elaborada. Se dispone de muchas tecnologías para llevar a cabo esta ampliación como puede ser: Python, C, ncurses, php, java swing, java j2ee, ...
- **Conexión con sistemas de notificación:** a partir de la aplicación desarrollada sería relativamente sencillo conectarla con un servidor smtp para notificar via email, conectarla a una pasarela sms o a una alarma visual o sonora.
- **Modo interactivo:** se propone implementar un modo interactivo en la aplicación de consola RemoteCon al estilo del interprete de python o php. Mediante éste se podría interactuar con las motas enviando parámetros y recibiendo mensajes de forma interactiva sin necesidad de ejecutar una linea por orden.
- **Envío de historial de valores para la generación de gráficas:** se podría implementar en las motas remotas la recolección de varios valores y el envío de un mensaje que los contenga cada cierto tiempo. Esto se podría utilizar en la aplicación en consola para generar estadísticas y gráficas de los valores detectados por los diferentes sensores.
- **Modificación de valores de forma masiva a varias motas a la vez:** una funcionalidad que sería de utilidad, sobre todo en despliegues de una gran número de motas, sería la modificación de los rangos de valores de manera masiva a multitud de motas a la vez.
- **Consulta de otros sensores:** ampliar la aplicación para la consulta de otros tipos de motas diferentes a la cou900 u otros sensores.
- **Detección de presencia por monitorización de luminosidad:** implementación de la detección de presencia en el CPD mediante el sensor de luminosidad.
- **Apagado de la radio** mientras no se necesite enviar un mensaje a la red, esto permitiría aumentar el tiempo de autonomía de las motas al reducir el consumo de energía.

Como puntos fuertes del sistema desarrollado se puede destacar la facilidad que supondría la implementación de las funcionalidades descritas, su flexibilidad y fácil adaptación a nuevas necesidades y cambios de hardware. Otro punto a favor es la gran cantidad de documentación y ayuda con la que cuenta al tratarse de un proyecto de software libre.

Como puntos débiles se puede destacar la falta de soporte por parte de un fabricante y la necesidad de inversión de tiempo en tener cierta destreza en el desarrollo de aplicaciones, al contrario que cuando se dispone de un producto ya desarrollado listo para funcionar.

Conclusiones

Como conclusión del trabajo desarrollado podemos decir que se trata una tecnología mediante la que se pueden cubrir multitud de frentes muy variados. Que destaca por su flexibilidad, facilidad de adaptación a diferentes situaciones y la implementación de nuevas funcionalidades. Con una inversión de tiempo inicial no demasiado grande podemos ofrecer multitud de soluciones a empresas a un precio bastante competitivo.

Quizás al principio, el tiempo que hay que invertir para tener un nivel de dominio que permita desenvolverse en esta tecnología, sea un factor negativo para muchos dirigentes técnicos de empresas para tomarla como una solución, pero también se cuenta con una gran cantidad de ayuda por parte de la comunidad para salir adelante ante problemas técnicos. Una vez que adquieres un conocimiento básico de las herramientas, se pueden crear aplicaciones rápidamente y empezar a ofrecer un servicio apoyado en este entorno.

En resumen, creo que tiene mucho potencial, y que puede ayudar a muchas organizaciones a controlar muchos aspectos físicos de forma relativamente sencilla, práctica y económica.

Glosario

- **Transceptor:** dispositivo que realiza, dentro de la misma caja o chasis, funciones tanto de transmisión como de recepción, utilizando componentes de circuito comunes para ambas funciones.
- **CMOS:** Complementary Metal-Oxide-Semiconducto, estructuras semiconductor-óxido-metal-complementarias, es una de las familias lógicas empleadas en la fabricación de circuitos integrados. Su principal característica consiste en la utilización conjunta de transistores de tipo pMOS y tipo nMOS configurados de tal forma que, en estado de reposo, el consumo de energía es únicamente el debido a las corrientes parásitas.
- **RISC:** (Reduced Instruction Set Computer) es un tipo de microprocesador que posee instrucciones de tamaño fijo y presentadas en un reducido número de formatos, y en el que solo las instrucciones de carga y almacenamiento acceden a la memoria de datos. Poseen muchos registros de propósito general. Su objetivo es posibilitar la segmentación y el paralelismo en la ejecución de instrucciones y reducir los accesos a memoria. Está a favor de conjuntos de instrucciones pequeñas y simples que toman menor tiempo para ejecutarse. Procesadores RISC: PowerPC, DEC, Alpha, MIPS, ARM,...
- **UART:** Universal Asynchronous Receiver-Transmitter, Transmisor-Receptor Asíncrono Universal. Controla los puertos y dispositivos serie. Se encuentra integrado en la placa base o en la tarjeta adaptadora del dispositivo. Las funciones principales de chip UART son manejar las interrupciones de los dispositivos conectados al puerto serie y convertir los datos en formato paralelo, transmitidos al bus de sistema, a datos en formato serie, para que puedan ser transmitidos a través de los puertos y viceversa. El UART toma bytes de datos y transmite los bits individuales de forma secuencial.
- **GPIO:** General Purpose Input/Output, son básicamente puertas programables de entrada y salida de datos. Son utilizadas para proveer una interfaz entre periféricos y los microcontroladores / microprocesadores.

Bibliografía

[1] **Levis, Philip / Gay, David. (2009)**
TinyOS Programming. Cambridge University Press.

[2] **Levis, Philip / Gay, David / Culler, Davil / Brewer, Eric. (2009)**
NesC 1.3 reference manual

[3] **Sistema Operativo TinyOS**
www.tinyos.net

[4] **Lenguaje de programación NesC**
www.nesc.sourceforge.net

[5] **Protocolo ZigBee**
www.zigbee.org

[6] **Datasheet microcontrolador atmega1281**
http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf

[7] **Datasheet Radio at86rf212**
www.atmel.com/dyn/resources/prod_documents/doc8168.pdf

[8] **Datasheet ATZB-900-B0**
www.atmel.com/dyn/resources/prod_documents/doc8227.pdf

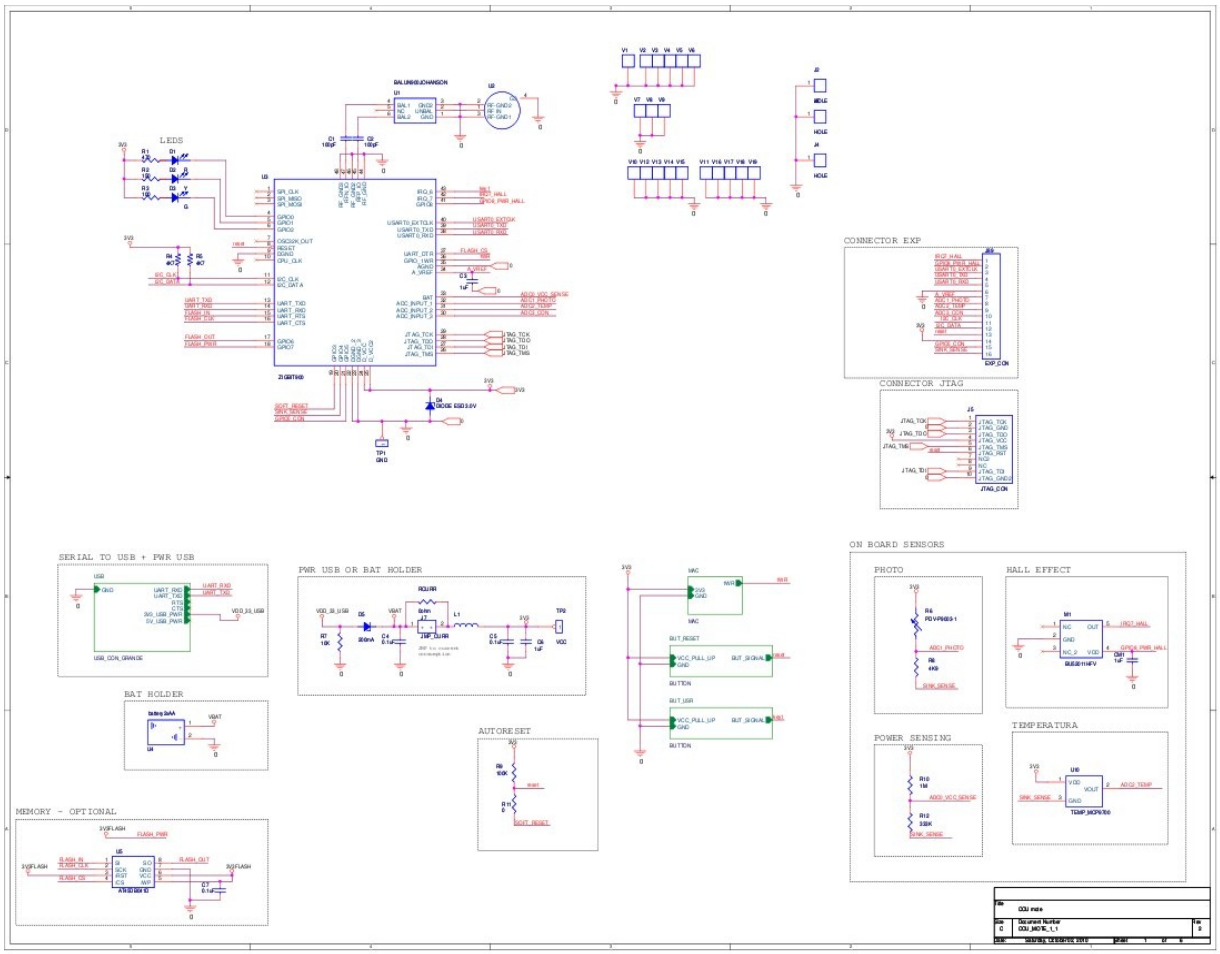
[9] **Datasheet del sensor de luminosidad PDV-P90003-1**
http://www.advancedphotonix.com/ap_products/pdfs/PDV-P9003-1.pdf

[10] **Datasheet del sensor de temperatura MCP9700A**
http://ww1.microchip.com/downloads/en/DeviceDoc/21942e.pdf

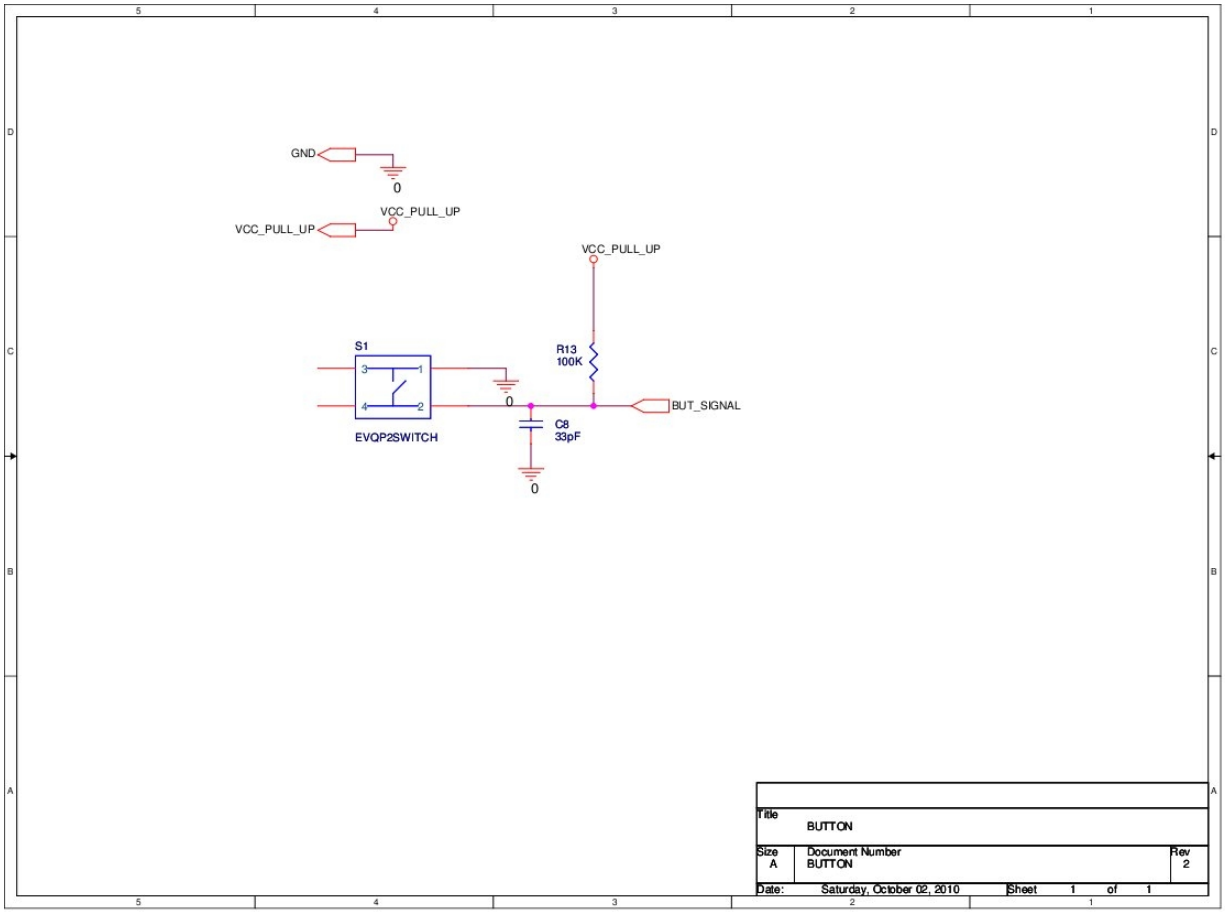
[11] **Datasheet del sensor de magnetismo BU520001gul-e**
http://www.rohm.com/products/databook/sensor/pdf/bu52001gul-e.pdf

Anexos

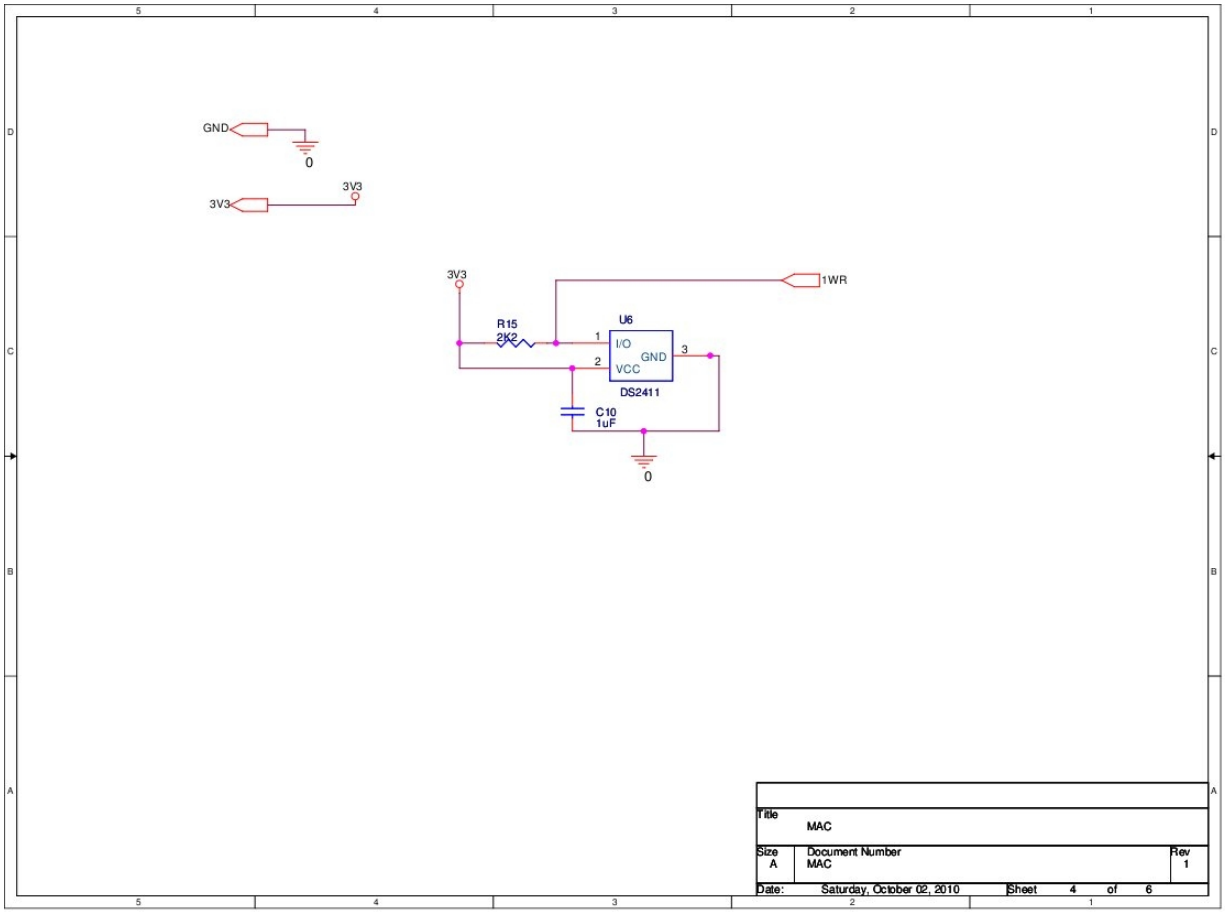
Anexo 1: Diagrama COU900



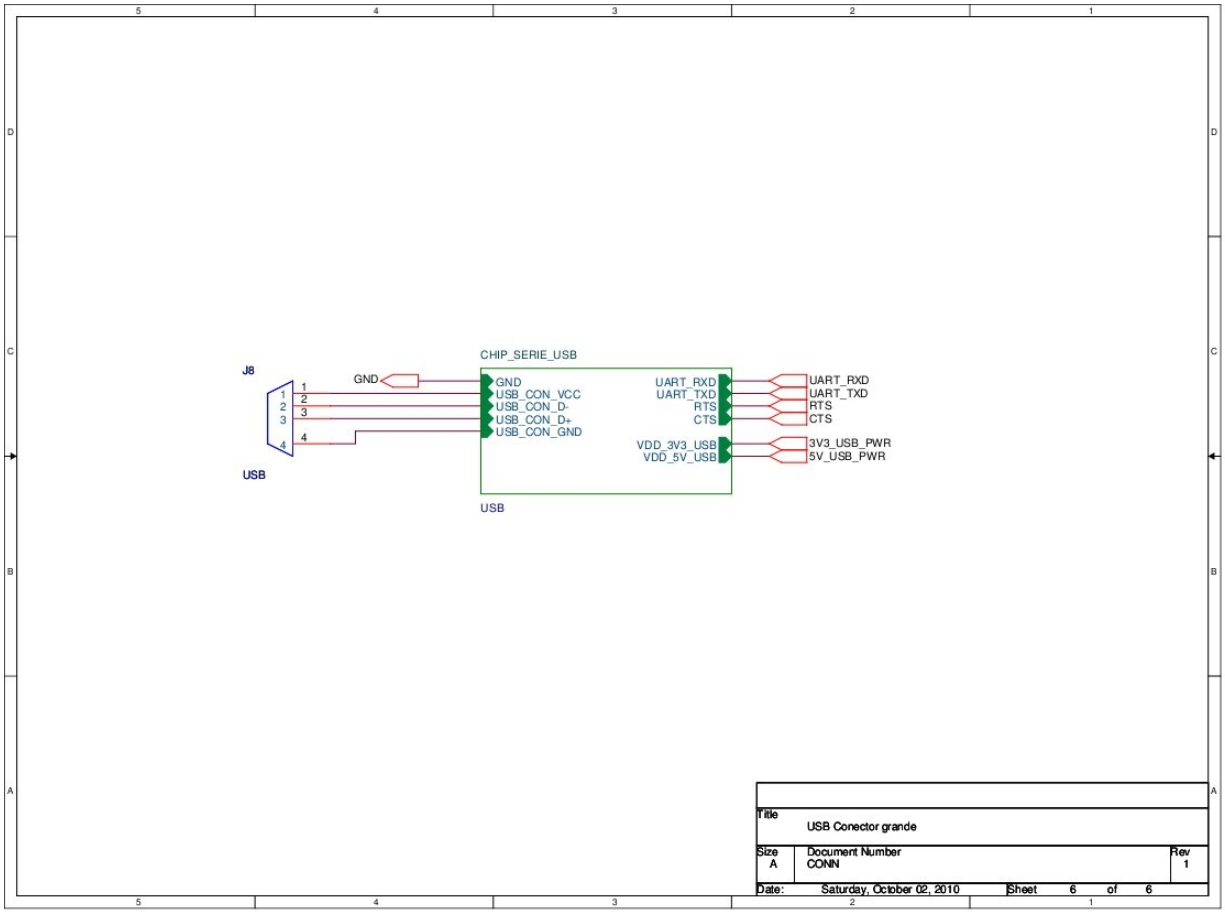
Anexo 2: Diagrama Button



Anexo 3: Diagrama MAC



Anexo 4: Diagrama conector USB



Anexo 5: Diagrama serie