Sistema de comunicación basado en una comunicación sin hilos para el control de luminosidad y temperatura

Fco. Fdez. de Córdoba

Ingeniería Técnica Informática Sistemas

Consultor: Jordi Bécares Ferrés

Fecha Entrega: Diciembre 2011

Agradecimientos

En general, a todos los que con sus aportaciones me han echado una mano y han contribuido a resolver las dudas que me han ido surgiendo a lo largo de la vida del proyecto.

En particular a mi familia, que además de sufrir mi ausencia por motivos laborales a lo largo de la semana también lo ha tenido que hacer algún fin de semana para la elaboración de este TFC.

Resumen

El trabajo consiste en desarrollar un software que permita, por un lado ordenar a un dispositivo remoto que tome valores de temperatura, luz y estado de las baterías (del propio dispositivo) y por otro evaluar dichos datos para avisar -mediante la generación de un fichero de alertas y a través de la pantalla de un ordenador- de las posibles medidas que estén fuera de rango.

Las muestras de los sensores se tomarán, bien cada determinado tiempo (establecido por el usuario al inicio del programa), bien cuando se pulse el botón de usuario en el dispositivo remoto.

De una forma más concreta, el proyecto consistirá en la programación de un módulo COU 1_2 24 A2 para la obtención de la temperatura, luminosidad y valor de sus baterías. Estos datos se enviarán a través de RF a la otra mota, que estará ejecutando el programa BaseStation y estará a su vez conectada a un ordenador..

Mediante otro programa en java, que se ejecutará desde el PC (al cual está conectada la mota que ejecuta el BaseStation), se visualizarán los datos de la temperatura y luz si estuvieran fuera del intervalo especificado por unos valores preestablecidos, a la vez que se grabaría una línea en un fichero de log, junto a la fecha, hora y tipo de alarma que ha provocado la alerta.

Los datos de temperatura máxima y mínima, así como de luz máxima y el valor mínimo de estado de las baterías, serán proporcionados por el usuario de la aplicación cuando ejecute la misma, al inicio. De esta forma, se podrán variar los mismos simplemente parando el programa y volviendo a ejecutarlo.

Área TFC: **SISTEMAS EMPOTRADOS**

Índice de contenidos

Contenido

Índice de figuras

Índice de figuras

1. Introducción

El aprovechamiento de las tecnologías inalámbricas supone un ahorro económico considerable para todos los proyectos en los que no haya cableado, bien por falta de previsión, bien por motivos físicos (si el sensor ha de estar en una nevera convencional, no sería recomendable que uno o más cables atravesaran la misma). Este proyecto utiliza esta tecnología sin cables para obtener unos datos distantes del ordenador.
Para la realización del trabajo, se programarán 2 motas utilizando las herramientas disponibles de Tinyos. También se utilizará un programa de envío/recepción de datos tanto RF como puerto serie, facilitado por el proveedor y su posterior visualización en la pantalla de un PC a modo de interfaz de usuario gracias al desarrollo de una aplicación en java.

1.1. Justificación

El proyecto se desarrolla para cubrir los aspectos básicos de programación de sistemas empotrados, de tal forma que interactúen con un PC y con un usuario final. La finalidad es obtener unos datos relativos a temperatura y luz en un momento dado para poder estimar si se han sobrepasado o no y actuar en consecuencia.

1.2. Descripción

Los requisitos del trabajo son los siguientes:

- **Enviar una orden de lectura** de sensores mediante la radio frecuencia cada cierto tiempo, desde el ordenador al dispositivo alejado. Dicha orden incluirá el valor de la temperatura máxima, la mínima, el voltaje mínimo, la luz máxima y el intervalo de tiempo entre órdenes sucesivas. Se solicitarán por pantalla al usuario y se enviarán al puerto serie del ordenador, para que la mota que está conectada a dicho puerto, mediante la ejecución del programa “BaseStation”, lo envíe por RF a la otra mota.
- **Interpretar la orden** de lectura de sensores desde la mota remota.
- **Ejecutar la orden** de lectura (leer datos de los sensores de luz, temperatura y voltaje de las baterías de forma asíncrona)
- **Enviar los datos vía RF**, tanto si al compararlos con valores de referencia están fuera de margen como si no.
- **Recuperar los datos del puerto serie.**
- **Gestionar la pulsación del botón de usuario,** para que se proceda al envío inmediato de los datos de los sensores.

- **Mostrar por la pantalla** de ordenador los datos recogidos del puerto serie.
- **Grabar en un fichero** de histórico todas las alertas generadas.
- **Gestionar un WDT** para controlar posibles caídas del dispositivo.

1.3. Objetivos del TFC

- ✓ Monitorizar la luz ambiente
- ✓ Monitorizar la temperatura
- ✓ Chequear estado de baterías del dispositivo
- ✓ Gestionar datos recibidos por RF desde la mota y desde el PC
- ✓ Enviar datos por RF desde el PC y desde la mota
- ✓ Activar/desactivar los leds de las motas según la función que se haya llevado a cabo
- ✓ Enviar alertas por pantalla y grabadas en fichero
- ✓ Establecer los tiempos de lectura de datos desde el programa del PC
- ✓ Introducir valores máximos y mínimos para temperatura, máximo para luz y mínimo para batería desde el programa java, que sirvan de márgenes para la activación de alarmas.
- ✓ Gestionar el uso del botón de usuario de la mota, para enviar los datos de manera inmediata por RF
- ✓ Programar un sistema de "WatchDogTimer" para controlar siempre el estado activo del dispositivo remoto
- ✓ Responder adecuadamente ante una reinicialización del dispositivo remoto

1.4. Enfoque y metodología empleada

El enfoque empleado para la ejecución del trabajo ha consistido en centrarse en la programación por un lado de una de las motas (ya que la otra estaría conectada al PC y simplemente se le cargaría en memoria el programa BaseStation), en el lenguaje que el proveedor del entorno propone: NesC, y por otro lado, en el PC utilizar java, para tratar todos los mensajes recibidos y mostrarlos en la pantalla al usuario, así como registrar los datos en un fichero de log.

De esta forma, se han podido separar las tareas, de manera que para la programación de la mota en NesC, apenas había necesidad de desarrollar nada en java. Una vez terminado

el programa de la mota, se ha pasado al desarrollo en java, anulando el timer que inicialmente se programo en NesC y sustituyéndolo por uno programado en java.

1.5. Planificación

Preparación del entorno de trabajo:

- Instalación SO
- Instalación herramientas de programación de la mota
- Pruebas del entorno (con ejemplos proporcionados por fabricante)

Programación del sistema:

- Acceso al sensor de temperatura
- Acceso al sensor de luz
- Acceso a estado batería
- Envío de datos vía RF mediante BaseStation, según protocolo 802.15.4
- Recepción de datos vía RF, según protocolo 802.15.4
- Control de pulsación de botón de usuario
- Elaboración del programa completo

Otros:

- Pruebas generales
- Elaboración de la memoria
- Elaboración de la presentación del proyecto

Los tiempos y fechas empleados para llevar a cabo las tareas se resumen en la figura 1: CRONOGRAMA

Tarea	Subtarea	Total días	Fecha Inicio	Fecha Fin
Preparación entorno:		10	01/10/2011	10/10/2011
	Instalación SO	4		
	Instalación Herramientas	3		
	Pruebas del entorno	3		
Programación del Sistema:		68	11/10/2011	20/12/2011
	Acceso sensor temperatura	3		
	Acceso sensor luz	3		
	Acceso a estado baterías	5		
	Encender LED's	2		
	Elaborar programa 1ª entrega código	20		15/11/2011
	Envío de datos usando BaseStation	5		
	Recepción datos vía RF	10		
	Programa, 2ª entrega código (y mejoras surgidas durante el desarrollo)	10		20/12/2011
	Pruebas del entorno (y mejoras)	5		20/12/2011
Elaboración Memoria:		55	10/11/2011	03/01/2012
Elaboración Presentación:		8	30/12/2011	10/01/2012

Fig. 1- Tabla tiempos

1.6. Recursos Empleados

- 2 dispositivos COU 1_2 24 A2 (figuras 2 y 3)

Fig. 2- Vista Motas

parte posterior:

Fig. 3- Vista Posterior Motas

- 1 PC con sistema operativo Ubuntu 10.04 y al menos un puerto USB
- 1 SO para programar los dispositivos: TinyOS
- Herramientas para generar/compilar/lincar/ descargar los programas (NesC, Meshprog, make, etc.) instaladas con Tinyos.
- Editor de textos: Gedit (visualizado en fig. 4)

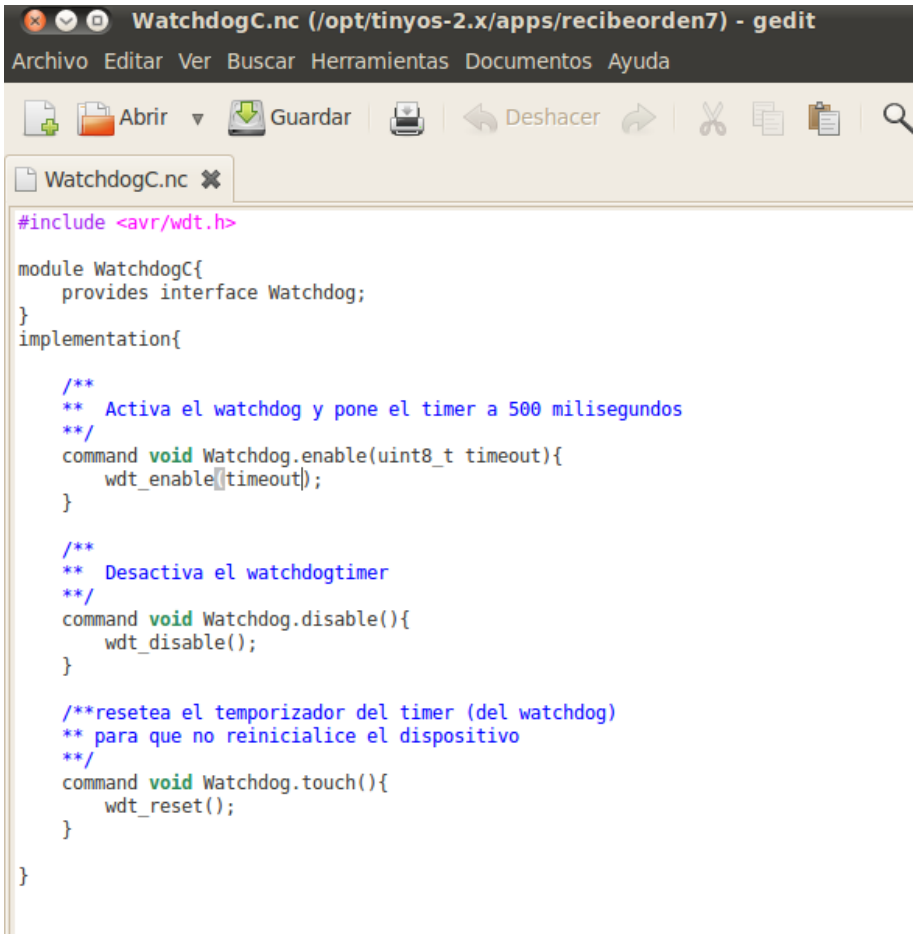

Fig. 4- Editor Gedit

- Lenguaje de programación: nesC
- Entorno de desarrollo Eclipse (fig. 5)

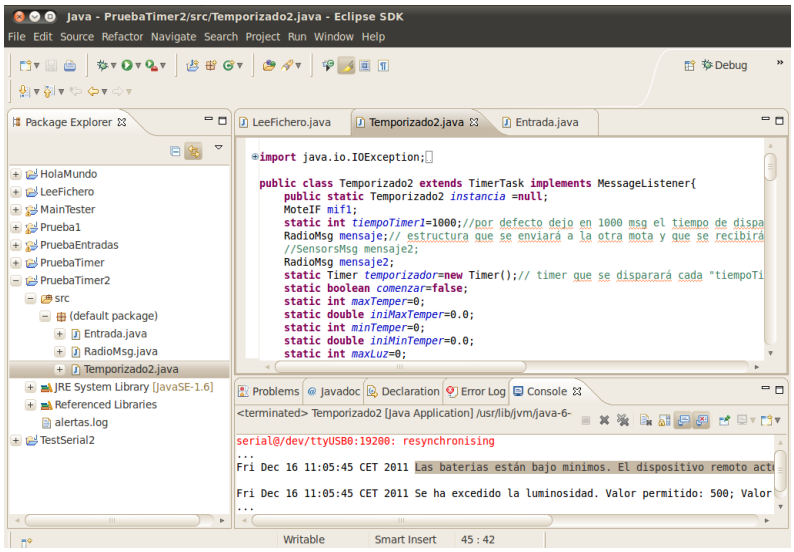

Fig. 5- Entorno Eclipse

- Programa BaseStation, proporcionado por Tinyos
- Tutoriales de Tinyos (página web con acceso a diferentes ejemplos para programación en nesC)
- tinyos-programming (libro en formato pdf sobre programación en nesC, fig. 6)

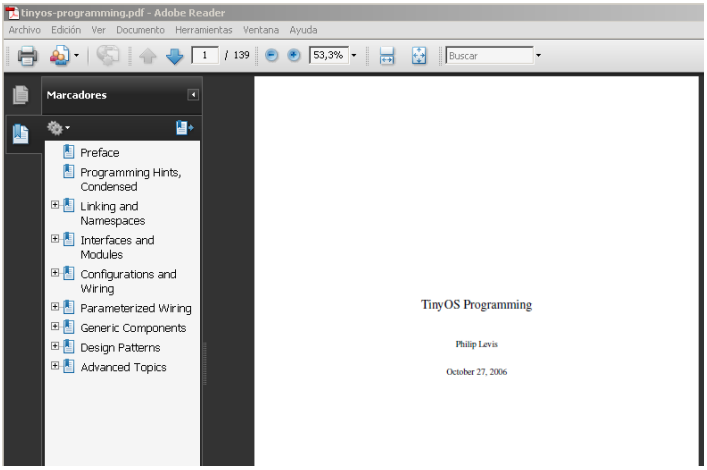

Fig. 6- Tinyos-programming pdf

El hardware básico de cada mota COU 1_2 24 A2 se ve detallado en la figura 7:

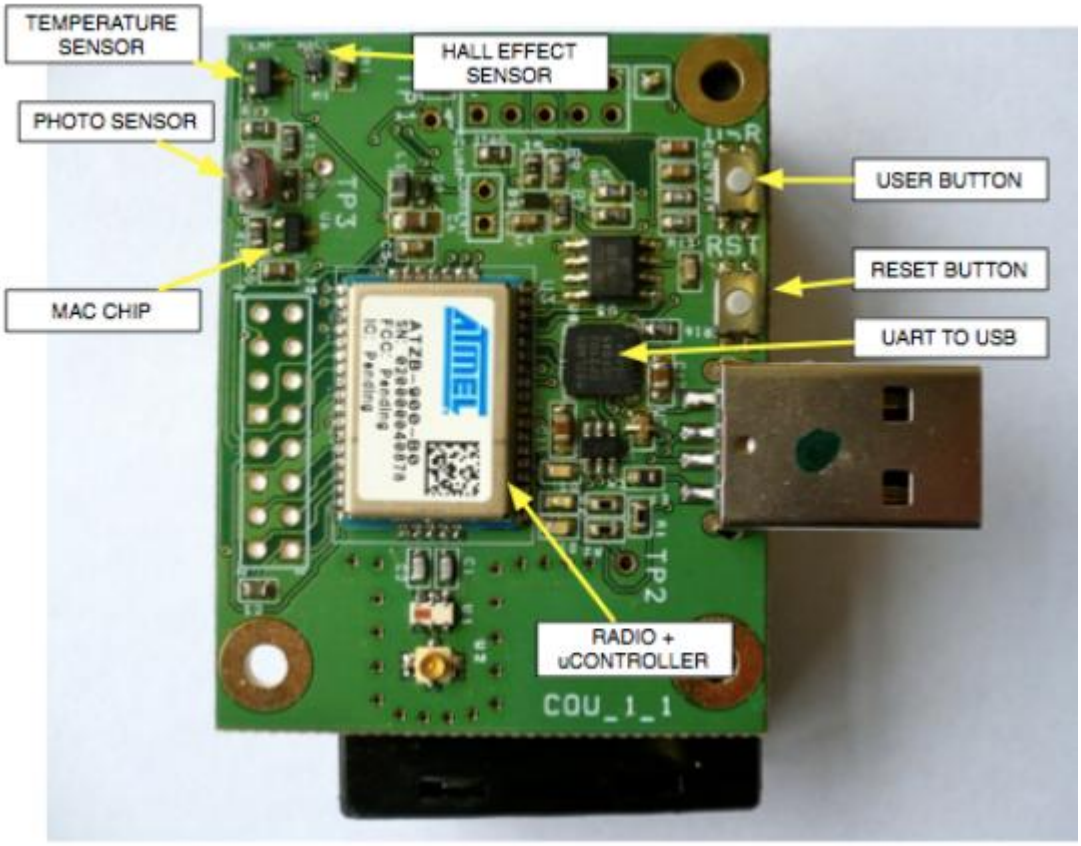

Fig. 7- Hardware básico de las Motas (imagen tomada de http://cv.uoc.edu/app/mediawiki14/wiki/Hardware_COU_1_2)

para el proyecto desarrollado utilizaremos: el sensor de temperatura, el sensor de luz, el botón de usuario, el botón de reset y el micro controlador + unidad de radio.

1.7. Productos Obtenidos

El producto final ha sido un sistema constituido por 2 motas, un ordenador y un software, capaces de tomar muestras de temperatura y de luz para enviar la información hasta un PC y registrar los datos en el mismo.

Se basa en el desarrollo de 2 aplicaciones y el uso de una aplicación proporcionada por Tinyos:

- ***Recibeorden***: aplicación escrita en nesC, que se instala en el dispositivo remoto y es la que controla que dicho dispositivo reciba una orden de lectura mediante

radio frecuencia, para proceder a leer los sensores de luz y temperatura del dispositivo y enviarlos a continuación al dispositivo conectado al PC.

- ***BaseStation***: proporcionada por Tinyos, que se encarga de transmitir por el puerto serie lo que recibe por RF y viceversa, o sea, transmitir por RF lo que recibe por el puerto serie.
- ***Temporizado***: aplicación escrita en java, que se ejecuta en el PC que tenga conectada la mota que ejecuta BaseStation y que es responsable, por un lado de enviar valores máximos y mínimos al puerto serie para que los reciba la otra mota y por otro lado, de recibir los datos que haya enviado el dispositivo remoto, para mostrar en pantalla los mismos si procede.

2. Antecedentes

2.1. Estado del arte

Los sistemas más comunes hoy en día, son simples sensores para ser insertados en otros dispositivos con el fin de cubrir una funcionalidad muy específica, por ejemplo: termómetros digitales: bien clínicos, bien para el hogar, reductores de intensidad lumínica basados en un sensor de luz tipo LDR, etc. Pero todos ellos, forman parte de un producto terminado, sin posibilidad de programarse para adaptarlo a otras tareas.

Las motas empleadas en el proyecto, disponen de esta característica de programación, de tal forma que pueden ser utilizadas para más de un fin. Si a esto le sumamos la capacidad de comunicarse de forma inalámbrica, las motas supondrán un avance en el desarrollo de nuevos sistemas que aportará grandes ventajas en el desarrollo y en la obtención del producto final.

2.2. Estudio de mercado

Actualmente existen algunas marcas que comercializan sistemas como el del proyecto, basados en la captación de datos variables y su posterior transmisión a un ordenador donde se registrarán y se gestionarán convenientemente.

Tradicionalmente, este tipo de control de temperatura e intensidad lumínica, se resolvía mediante grandes y numerosas redes de cableado, una por cada sensor, si bien, con posterioridad se ha podido aplicar el protocolo *Master-slave*, para conducir señales de varios sensores a través de un solo trazado de cable.

La funcionalidad inalámbrica supone un gran avance y una reducción de costos de instalación y mantenimiento sin precedentes, que permitirá abordar proyectos escalables sin tener que preocuparse de nuevas obras para la instalación de los nuevos puntos de control de medida.

Fundamentalmente están dirigidos al sector farmacéutico, laboratorios y empresas de conservación de alimentos. Los primeros por el elevado precio de alguno de sus productos y el segundo por la trascendencia que pudiera tener la rotura de la cadena de frío en los alimentos destinados al consumo por parte de las personas.

3. Descripción funcional del sistema

3.1. Sistema total

Para la obtención del sistema completo se seguirán las siguientes pautas:

1ª) Instalación de Tinyos (entorno de trabajo)

2ª) Instalación y ejecución del programa “recibeorden” en la mota remota

3ª) Instalación y ejecución del programa BaseStation en la mota que a su vez estará conectada al PC (figura 8)

4ª) Instalación y ejecución del programa java “temporizado”, desde el PC

5ª) Especificación del tiempo entre lecturas, temperatura máxima, temperatura mínima, intensidad lumínica máxima y voltaje mínimo de las baterías desde el PC.

6ª) Consulta de resultados obtenidos por las alertas

Fig. 8- Conexión Mota-PC

El Diagrama de bloques de la aplicación de la fig. 9, muestra el funcionamiento interno del sistema:

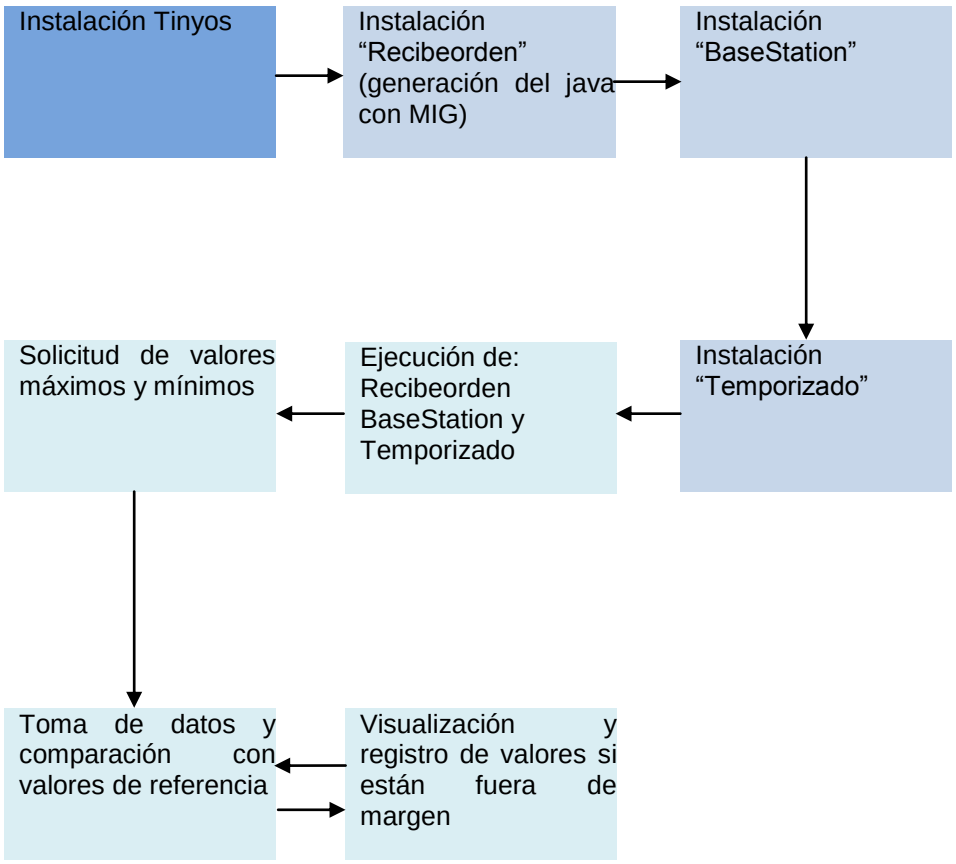

Fig. 9- Diagrama de bloques

3.2. PC

3.2.1. Temporizado

El programa desarrollado en java para ejecutarse en el PC “Temporizado” en el momento de su ejecución solicitará un tiempo, transcurrido el cual enviará periódicamente al puerto serie el resto de los datos que se han solicitado al usuario (ver fig. 10), como son: temperatura máxima, temperatura mínima, luz máxima, batería mínima. Si en el directorio desde el cual se ejecuta el programa java existiera un fichero con estos datos, el programa no los solicitaría al usuario.

Por otro lado, el programa esperará recibir algún dato por el puerto serie (procedente del dispositivo remoto). Una vez recibido, procederá a validar la autenticidad del mismo -mediante la comparación de una clave- y evaluará los distintos valores recibidos en una estructura de datos. Si dichos valores están fuera de los introducidos por el usuario como máximos y/o mínimos, mostrará una alerta por pantalla y registrará una línea en un fichero de log (ver fig. 11).

Otra misión de este programa es gestionar convenientemente si recibe un mensaje indicando el inicio del programa en el dispositivo remoto, o bien su reinicialización.

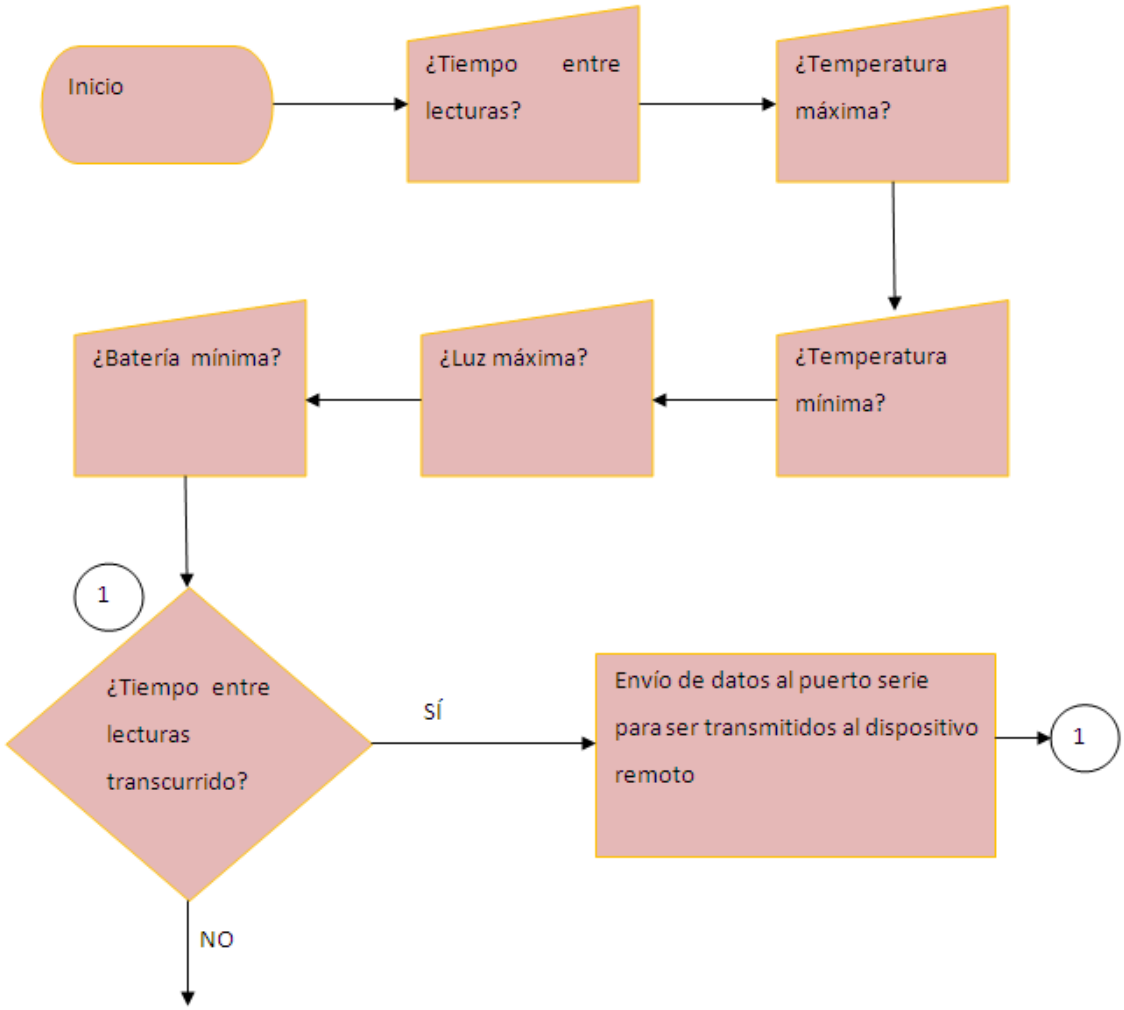

Fig. 10- Diagrama Flujo Temporizado 1

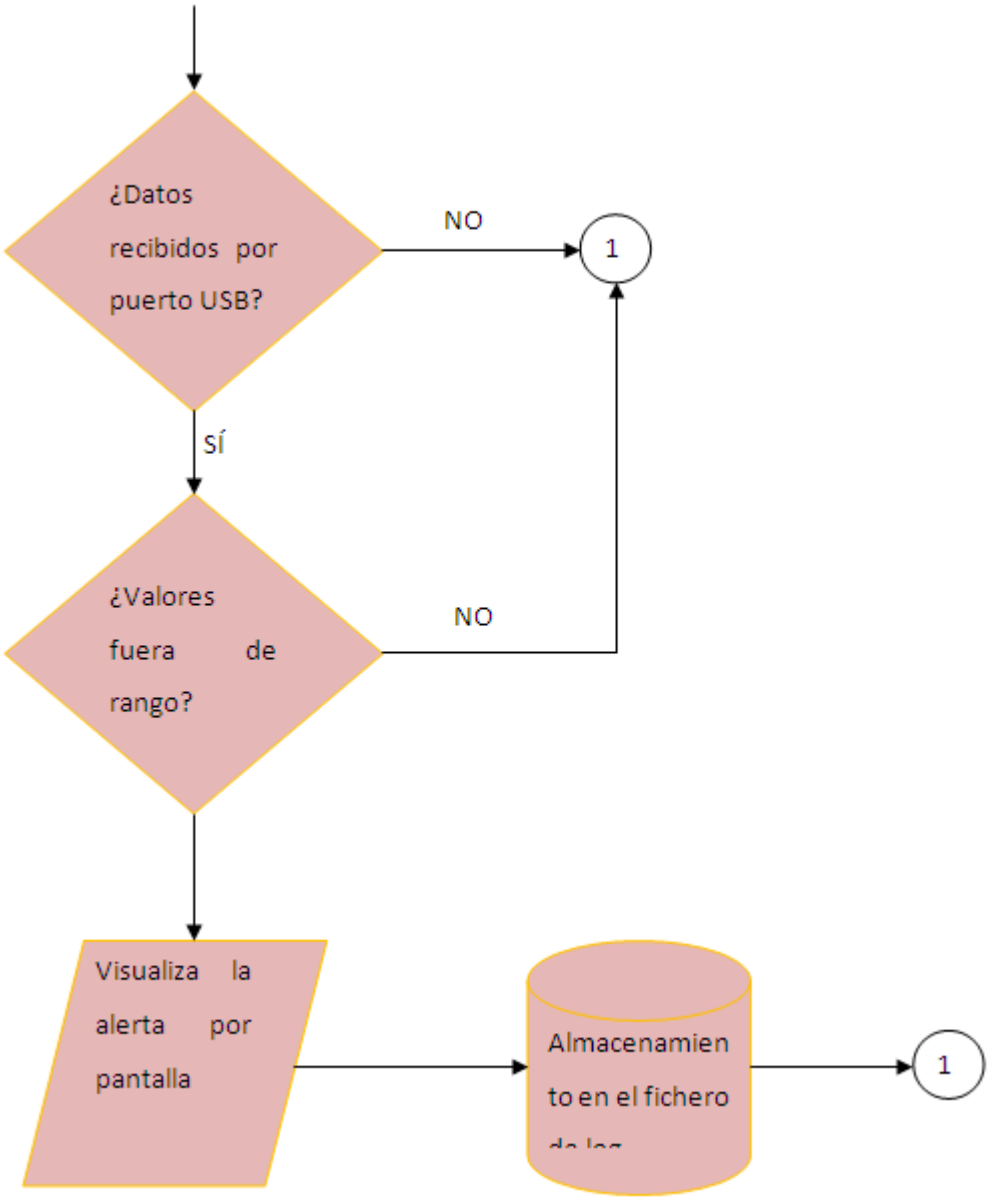

Fig. 11- Diagrama Flujo Temporizado 2

3.3. Mota

3.3.1. BaseStation

El programa BaseStation, suministrado por Tinyos, se instalará en la mota que está conectada al PC. Tendrá por misión:

1º) enviar lo que reciba por el puerto serie (del ordenador) a la mota remota vía RF

2º) enviar lo que reciba por RF (del dispositivo remoto) al puerto serie del ordenador, de tal forma que lo reciba el PC.

3.3.2. Recibeorden

El programa Recibeorden instalado en el dispositivo remoto, se encargará de leer de sus sensores de luz, temperatura y batería los valores correspondientes para enviarlos por RF. Dicha lectura la llevará a cabo siempre que reciba por RF un mensaje solicitándola. El diagrama de flujo de la fig. 12 muestra el comportamiento.

Dispone a su vez de un control de *WatchDog*, de manera que transcurridos 8 segundos sin que haya actividad se resetea el dispositivo, evitando de esta forma “cuelgues” del dispositivo sin gestionar. Para avisar de esta situación, en el arranque se enviará un “aviso” de dicha reinicialización por RF, para que sea tratada convenientemente por el programa java. El diagrama de flujo básico de esta parte del programa (fig. 12), se presenta a continuación:

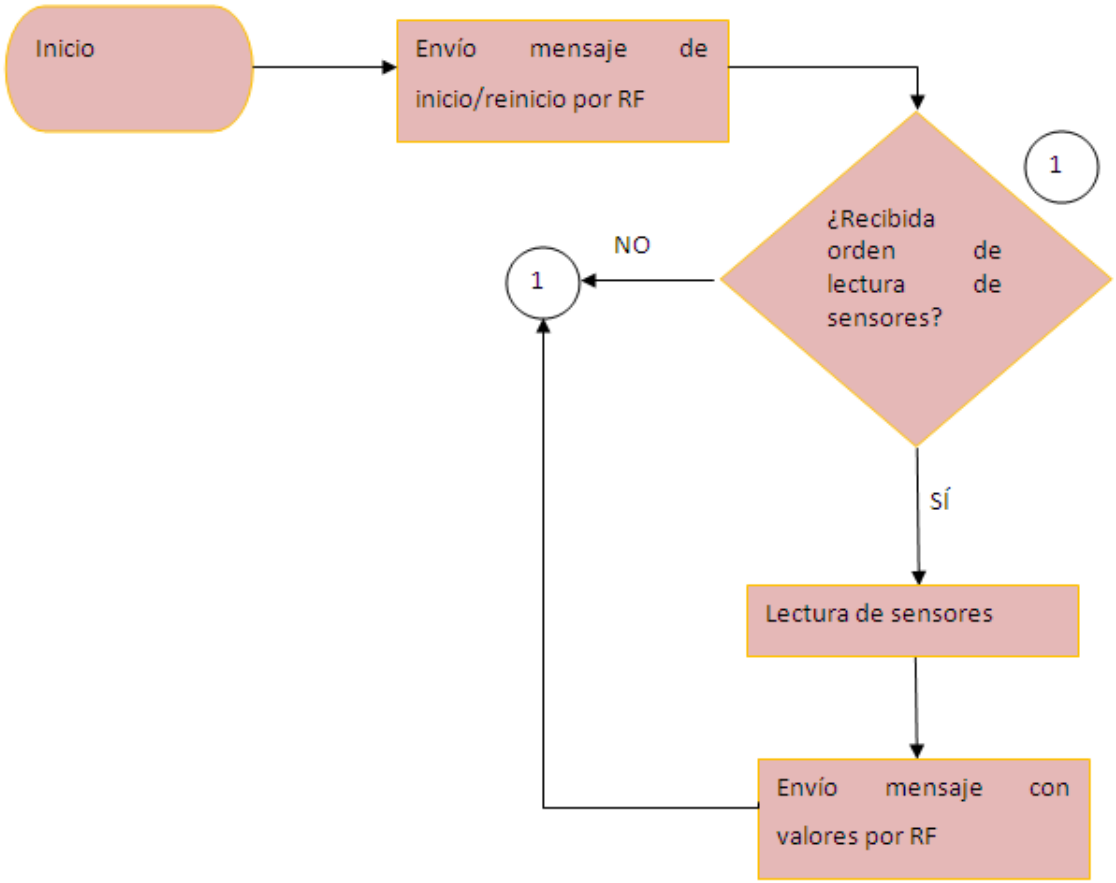

Fig. 12- Diagrama Flujo Recibeorden 1

En cuanto al *WatchDogTimer*, consistirá en la puesta de un contador a 8000 milisegundos, de tal forma que en cuanto llegue a cero reseteará la mota. Para evitar esto, se crea un "timer" que se dispara cada 2000 msg. y que pondrá de nuevo el valor del contador de reset en 8000, para evitar que se reinicialice el dispositivo. Su diagrama de flujo (fig. 13) quedaría resumido así:

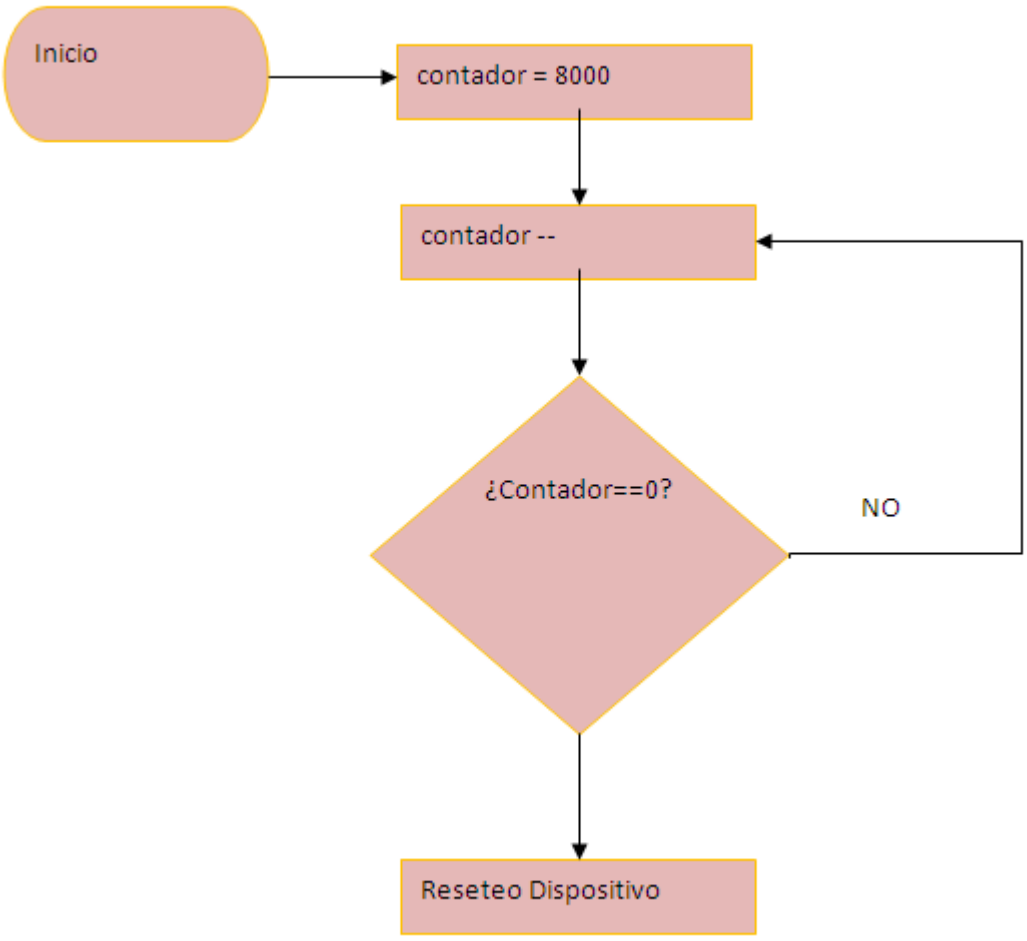

Fig. 13- Diagrama Flujo Recibeorden 2

En la figura 14, se observa el funcionamiento del *timer* que repone el contador para evitar que se reinicialice el dispositivo:

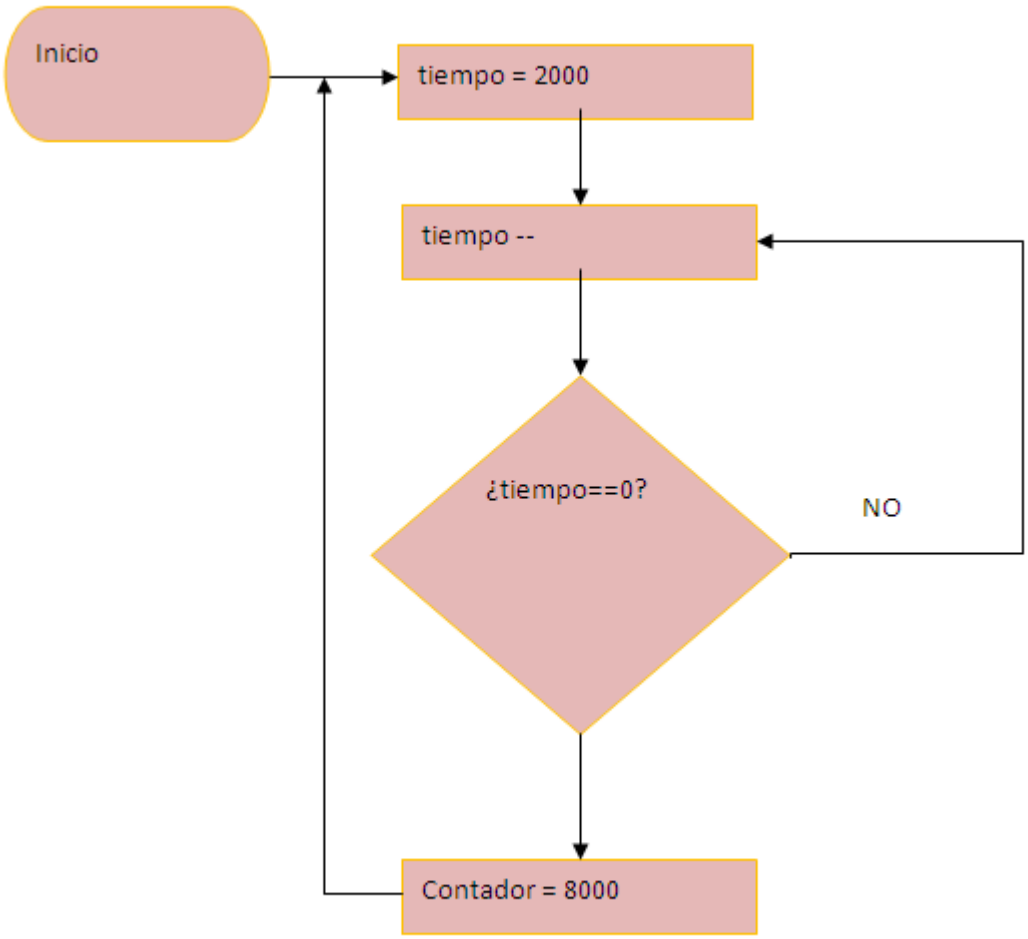

Fig. 14- Diagrama Flujo Recibeorden 3

4. Descripción Detallada

4.1. Instalación de Tinyos, BaseStation, RecibeOrden y Temporizado

4.1.1. Instalación de Tinyos

Se llevará a cabo según la documentación http://cv.uoc.edu/app/mediawiki14/wiki/Inici24,

4.1.2. Instalación de BaseStation

Para instalar el programa en el dispositivo, se ejecutan los siguientes pasos:

- Compilar fuentes del programa
- Conectar físicamente la mota al ordenador
- enviar el ejecutable a la mota

Compilar fuentes del programa

Para llevar a cabo este proceso, bastará con situarnos en el directorio donde se nos haya instalado el proyecto “BaseStation” (al instalar todo el entorno de Tinyos) y, una vez en el mismo, se tecleará “make cou24” y se generarán los archivos necesarios (sobre todo *main.srec*) en nuevas carpetas denominadas “build” y “cou24”:

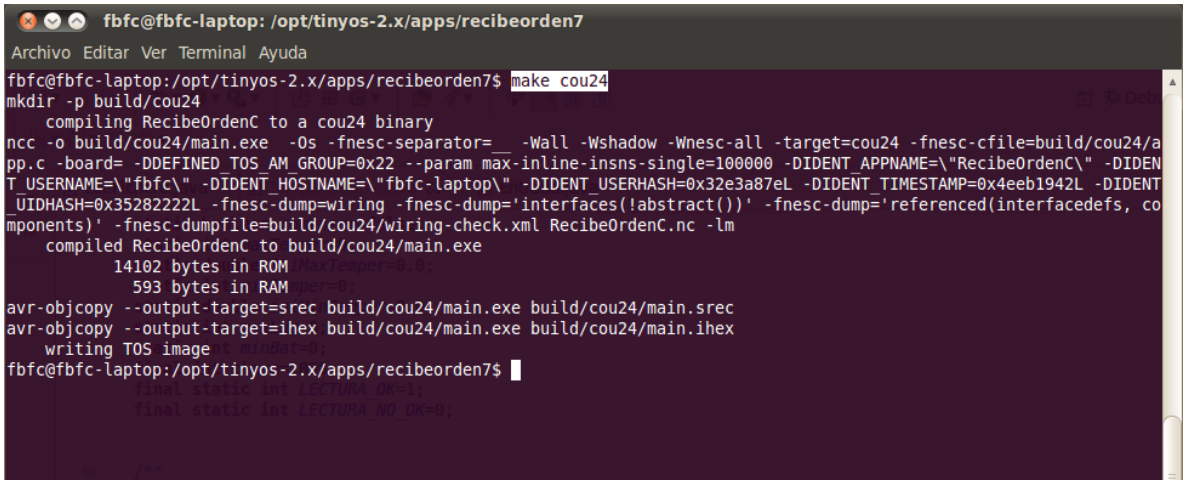

Fig. 15- Compilar con Make

Conectar físicamente la mota al ordenador

Una vez generados estos los archivos sin error, se deberá conectar el dispositivo al ordenador a través del puerto USB, tal y como se observa en la figura n y se procederá a instalarlos en la mota tal y como se explica en el paso siguiente.

Enviar ejecutable a la mota

Para esta tarea, utilizaremos “meshprog”, que es una utilidad facilitada por Tinyos. La sintaxis correcta sería “meshprog –t/dev/ttyUSB0 –f ./build/cou24/main.srec”, donde se le está especificando el puerto al que está conectada la mota (ttyUSB0) y dónde está el archivo “main.srec” obtenido en el paso anterior (proceso de compilación). En la figura 16, se observa el resultado de una ejecución inicial de “meshprog”.

Fig. 16- Cargar programa en Mota 1

Si todo va bien, la utilidad intentará abrir el dispositivo e indicará en pantalla mediante el mensaje “opened file” el éxito de la operación, tal y como se observa en la figura 16. A continuación, espera que el dispositivo esté preparado para cargar el programa en su memoria, lo cual se muestra en la pantalla con una serie de puntos suspensivos que se van incrementando en número, a la espera de que se pulse, en la mota, el botón de reset. Si no se pulsara ese botón en un determinado tiempo, se nos mostrará un mensaje de abandono de la operación, tal y como se ve en la figura 17 siguiente:

Fig. 17- Cargar programa en Mota 2

Si por el contrario, pulsamos el botón de reset de la mota a tiempo, como vemos en la figura 18

Fig. 18- Cargar programa en Mota 3

en la pantalla se nos indicará que ha comenzado el proceso de envío de datos hacia la mota con el mensaje de “Starting transmission”, hasta el momento en que termine de enviar el fichero al dispositivo, que nos lo hará saber mediante el “finished” visualizado en la figura 19

Fig. 19- Cargar programa en Mota 4

A partir de este instante, el programa que se ejecuta en esta mota es el que hemos cargado, es decir, el BaseStation.

4.1.3. Instalación de Recibeorden en la mota

Para llevar a cabo la instalación en la mota del programa Recibeorden, se seguirán los mismos pasos que los descritos en el apartado anterior (instalación de BaseStation), si bien, habrá que tener en cuenta, que el directorio en el que nos situemos para el proceso de compilación será aquel en el que hayamos guardado los ficheros fuentes que componen la aplicación.

Durante el proceso de compilación, se observará en pantalla que se crean otros ficheros, concretamente “RadioMsg.java”, que utilizaremos más adelante en la instalación del programa java.

4.1.4. Instalación de Temporizado

Esta aplicación desarrollada en java, se instalará en el entorno de desarrollo elegido para java. Si es Eclipse, se tendrá que crear un proyecto, añadirle al mismo los ficheros java desarrollados para este trabajo y, también habrá que tener en cuenta que necesitará acceso a las librerías de Tinyos en formato “jar”.

Si por el contrario no se usa el entorno de desarrollo mencionado, bastará con copiar en un directorio todos los fuentes de java (incluyendo el que se a mencionado en el apartado anterior), para que en el momento de su compilación disponga de los mismos y genere los “.class” necesarios en la ejecución.

4.2. El programa BaseStation

BaseStation es un programa que forma parte de las varias utilidades que proporciona Tinyos y que quedarán instaladas en nuestro ordenador una vez se haya instalado el entorno de Tinyos. Tal y como se refiere en los tutoriales de Tinyos, BaseStation actúa como un puente entre el puerto serie y la red RF. Cuando recibe un paquete desde el puerto serie lo transmite por la radio y cuando lo recibe por radio, lo transmite por el puerto serie.

Con esta capacidad de recepción/transmisión de uno a otro puerto y viceversa, BaseStation es una utilidad idónea para todos aquellos casos en los que interesa comunicar el ordenador con la red de dispositivos inalámbricos.

De esta forma, la aplicación tiene como objetivo fundamental el envío por radio frecuencia de lo que recibe por el puerto serie (USB) y de enviar por el puerto serie todo lo que reciba por radio frecuencia. De esta manera, conseguiremos que al enviar datos desde un dispositivo remoto -por radio frecuencia- a la mota que ejecuta el BaseStation, esta última los reciba y los transmita por el puerto serie, para que el ordenador al que está conectado pueda interpretarlos mediante un programa de lectura de datos del puerto serie. También conseguiremos que al enviar datos al puerto serie de un ordenador –mediante un programa que se ejecute en dicho ordenador-, los datos en cuestión sean leídos por el BaseStation y transmitidos por radio frecuencia para que puedan llegar a los dispositivos remotos.
Con respecto al encendido/apagado de los *leds* de la mota que ejecuta el programa BaseStation, son los que el fabricante programó en su momento, no se han modificado. Esto significa que:

- un toggle del **led rojo**--> indica mensaje recibido por puerto serie y enviado por RF
- un toggle del **led verde**--> indica mensaje recibido por RF y enviado al puerto serie
- un toggle del **led ámbar**--> indica pérdida de mensaje por desbordamiento

4.3. El programa Recibeorden

Esta aplicación es la que se ejecuta en el dispositivo remoto. Consta de los siguientes ficheros:
“enviaorden.h”, “MuestreaP.nc”, “devuelveRF.nc”, “devDataRF.nc”, “Watchdog.nc”, “WatchdogC.nc”, “RecibeOrdenP.nc”, “RecibeOrdenC.nc” y “Makefile”

- **“enviaorden.h”** Es un fichero de cabecera utilizado para definir constantes y estructuras que se compartirán entre los distintos módulos de la aplicación. La figura principal es la definición de la estructura de trabajo “RadioMsg”, la cual está formada por los miembros que contendrán los datos que se enviarán y recibirán entre las aplicaciones “java” y “nesC” (temperatura máxima, temperatura mínima, temperatura leída, luz, batería, etc.)
- **“MuestreaP.nc”** es el módulo que contiene la función de inicio de la aplicación y donde se trata de encender la radio. Mientras no lo consiga, seguirá intentándolo. También es el módulo donde se activa un WDT con un periodo de 8000 msg. transcurridos los cuales, reseteará la mota. A su vez, se crea un temporizador que

se dispara cada 1000 msg. y cuyo cometido es el reseteo del *wdt*, para que no reinicie la mota.

Otra funcionalidad llevada a cabo en este módulo, es el envío de la estructura RadioMsg por radio, indicando que se ha iniciado/reiniciado el dispositivo al poner a "1" el valor del campo "counter".

- **"devuelveRF.nc"** es un módulo en el que se declara la interfaz devuelveRF y una función "*uint8_t devuelveDatos(RadioMsg* data)*"
- **"devDataRF.nc"** es el módulo en el que se implementan las funciones de la interfaz devuelveRF. En concreto se implementa la función "uint8_*t devuelveDatos(RadioMsg* data)"* que es la responsable de enviar por RF la estructura "data" de tipo RadioMsg. También se tratan en este módulo los filtros para enviar o no los datos por RF, es decir, se comparan valores máximos y mínimos con los valores leídos por el sensor (de temperatura, de luz y batería), se comprueba si el valor de uno de sus miembros es "1" (concretamente "data->tiempo") y se sabe por consiguiente si ha habido pulsación del botón de usuario o no. Esta función retornará un valor de ENVIADO si así ha sido o NO_ENVIADO si no se hubiese llevado a cabo el envío de los datos. En concreto, en este módulo, es donde se decide si se van a enviar o no los datos por RF para que sean recibidos por el programa java.
- **"Watchdog.nc"** es un modulo de declaración de la interfaz (que lleva su nombre) así como de sus 3 funciones: "`void enable(uint8_t timeout);`", "`void disable();`" y "`void touch();`" .
- **"WatchdogC.nc"** es donde se implementan las funciones de la interfaz Watchdog. La primera de ellas "`void enable(uint8_t timeout`)" es la encargada de habilitar el WDT por un período de "timeout" milisegundos (8000 en concreto para el programa).

  "`void disable()`" como su nombre indica, deshabilita el WDT y la tercera, "`void touch()`" se encarga de poner el contador del WDT a su valor inicial para que no llegue a 0 y resetee la mota.

  Estas funciones son las utilizadas desde el módulo descrito anteriormente MuestreaP.nc y que se encargaba del control del WDT y de su anulación mediante el *timer*.

- **“RecibeordenC.nc”** es el fichero de configuración, donde se detallan los módulos utilizados y se llevan a cabo los enlaces (wiring) entre los módulos y sus interfaces.
- **“RecibeOrdenP.nc”** es donde reside la lógica de la lectura de los sensores y el momento de su lectura.
- **“Makefile”** es un archivo necesario a la hora de compilar la aplicación. En el mismo, residen datos para saber qué se ha de hacer y en qué orden. En concreto, se le indica cual es el archivo de configuración de la aplicación “RecibeOrdenC.nc” y además se invoca al programa “mig” para que genere, a través de la estructura RadioMsg almacenada en “enviaorden.h” un módulo java con los accesores necesarios para todos los miembros de la citada estructura, de tal forma que luego este fichero pueda ser incluido en el proyecto “java” y facilite el acceso a dichas funciones.

La estructura principal de “Recibeorden” es “RadioMsg”. Se utiliza para almacenar los valores leídos de los distintos sensores, los valores máximos y mínimos introducidos por el usuario y para otro tipo de indicadores de flujo de programa (si se ha pulsado botón de usuario o no, si se reseteado el dispositivo...). Consta de los siguientes elementos

- *nx_uint16_t nodeid* Entero de 16 bits utilizado para identificar a la mota.
- *nx_uint16_t counter* Entero de 16 bits que se usa para saber si se ha reiniciado el dispositivo remoto (si su valor es 1) o no, en cuyo caso valdrá 0.
- *nx_uint16_t clave* Entero de 16 bits que se utiliza para saber si el mensaje pertenece al conjunto de mensajes de la aplicación o bien es propiedad de otro dispositivo, o son datos corrompidos, etc. Se usa con el valor de “1098” para saber que sí es un mensaje del programa, otro valor indicará que no lo es.
- *nx_uint16_t tiempo* Entero de 16 bits, si el valor es 1, significará que se ha pulsado el botón de usuario.
- *nx_uint16_t maxtemp* Entero de 16 bits destinado a almacenar el valor que el usuario quiere que sirva de referencia para la temperatura máxima.
- *nx_uint16_t mintemp* Entero de 16 bits en el que se almacenará el valor de la temperatura mínima, según haya especificado el usuario de la aplicación.
- *nx_uint16_t maxluz* Entero de 16 bits en el que quedará guardado el valor máximo de luminosidad, por encima del cual se generará la alarma de exceso de luminosidad.

- *nx_uint16_t minbateria* Entero de 16 bits destinado al almacenamiento del valor mínimo que deben tener las baterías del dispositivo remoto antes de generar una alerta por baterías bajo mínimos.
- *nx_uint16_t Temperatura* Entero de 16 bits en el que se almacenará el valor de la temperatura leído del sensor del dispositivo.
- *nx_uint16_t Luz* Entero de 16 bits para almacenar el valor de luz leído desde el sensor de luz del dispositivo.
- *nx_uint16_t Bateria* Entero de 16 bits que recoge el valor del estado de las baterías del dispositivo.
- *nx_uint16_t vrefmilivolts* Entero de 16 bits en el que se almacena el voltaje de referencia para este tipo de dispositivo.

El funcionamiento detallado del programa sería:

1º) Nada más iniciarse la aplicación, se intenta encender la radio. Hasta que no se consiga, se repite indefinidamente. Si se ha conseguido encender, se activa el control del *WatchDogTimer* con un tiempo de 8000 msg. También se programa el disparo de un *timer*, que servirá para evitar el reseteo involuntario del dispositivo, cada 1000 msg.

A partir de este instante, el programa controla que cada 1000 msg. se pondrá el contador de reset a su valor de inicio y evitará así que entre en funcionamiento el reinicio por parte de WDT. De esta forma, se garantiza que sólo se reseteará la mota, cuando realmente se "descontrole" el programa del dispositivo y se quede en un bucle, del cual saldrá gracias al evento del *WDT* cuyo resultado es el reinicio de la mota.

Otra función importante asociada al evento de "*startDone*" de la radio es el envío de un mensaje por RF. Este mensaje, tiene por finalidad avisar de que el dispositivo se ha arrancado, simplemente. De esta forma, el programa java, sabrá cómo actuar ante esta situación y registrará convenientemente esta información.

Para proceder al envío por RF de este mensaje, primero comprobará el estado de una variable booleana "busy", si está a FALSE, procederá a igualar un puntero creado anteriormente, del tipo RadioMsg con el *payLoad* del mensaje. De esta forma podrá dar el valor adecuado al miembro "clave" –ponerlo a 1098 para validar la autenticidad del mensaje- y almacenar un 1 en el miembro de la estructura "counter" avisando de esta forma de que se trata de una inicialización/reinicialización del dispositivo. A continuación procede al intento de envío del mensaje mediante la función "*send*" de la interfaz "*AMSend*". Si esta función devuelve SUCCESS se pone "busy" a TRUE (para que no se envíe nada mientras tanto) y se trata el evento "*sendDone*". Si la función anterior no

devuelve SUCCESS, se pone el valor de “busy” a FALSE, para que se pueda utilizar la radio y transmitir sin problemas.
En el momento en que se trata el evento “*sendDone*”, lo que se hace el liberar el semáforo “busy”, y ponerlo a FALSE para que se puedan enviar mensajes a través de la radio.
2º) A partir de este instante, la aplicación se queda a la espera de recibir un mensaje por RF. Mientras tanto no hace nada, sólo esperar a dicho mensaje.
Cuando recibe un mensaje, prepara un puntero del tipo RadioMsg para tratar los datos y ante todo, comprueba que el valor del campo “clave” sea igual a 1098 dado que si no fuera así, se trataría de la recepción de un paquete que no sería esperado. En este último caso, haría un *toggle* del *led* naranja.
En el caso de que la clave sea la correcta, lo primero sería hacer un *toggle* del *led* rojo (con esto indica que ha recibido un mensaje válido). A continuación, asigna los valores leídos en el mensaje, a la estructura de trabajo con la que trabaja, de tal forma que almacena convenientemente:

- el valor de la batería mínima
- el valor de la luz máxima
- el valor de la temperatura máxima
- el valor de la temperatura mínima

Con estos datos (enviados desde el programa java cada n milisegundos) la aplicación ya conoce los rangos mínimos y máximos.
lo siguiente que lleva a cabo, es una comprobación para saber si el botón de usuario está listo, es decir, todo lo que concierne para la gestión del botón de usuario.
Para ello, evalúa la variable “botonOk” y, si su valor es NO_READY, procede a ejecutar la tarea “*preparaBoton()*”. Dicha tarea, en primer lugar asigna el botón como entrada (*ButtonPin.makeInput()*) y “limpia” el valor que pudiera tener hasta ese momento (*ButtonPin.clr()*). Más tarde, se resetea y se activa la interrupción asociada al botón de usuario (*Button.edge()*, *Button.clr()* y *Button.enable()*). Por último, se pone a BOTON_READY la variable “botonOk”, de tal forma que ya no se vuelva a preparar esta interrupción para el botón de usuario a lo largo del programa, salvo que se reseteara el dispositivo, pues en este último caso se volvería a preparar el botón de usuario.
Una vez preparado de manera conveniente el botón de usuario para poder gestionar su pulsación, el programa pone la variable “status” a READY y ejecuta la tarea *nextStep()*.
Al ejecutar *nextStep()* y encontrar en el case que el valor de “status” es READY, la aplicación interpreta que es el inicio de una solicitud de lectura de sensores. Prepara un

puntero para el *payload* del mensaje y cambia el "status" a WARMED_UP. Acto seguido llama de nuevo a la tarea *nextStep()*.

Cuando se evalúa dentro de la tarea el valor de "status" y este es WARMED_UP, primero cambia el valor del mismo a READING_BAT y hace una llamada a la función de la interfaz *LecturaAdc.read()*, que si devuelve distinto de SUCCESS pondrá "status" a ERROR y ejecutará de nuevo una llamada a la tarea *nextStep()*.

Sin embargo, *si LecturaAdc.read()* devuelve un SUCCESS, cuando se trate la interrupción de lectura de sensores, se verificará que el "status" es READING_BAT y se devolverá el valor de las baterías en una variable denominada "val".

Esta variable será asignada al miembro de la estructura con la que se trabaja, denominado Bateria en el evento *LecturaAdc.readDone*. En este mismo evento (donde se habrá evaluado previamente el valor de "status" y se habrá comprobado que es READING_BAT) se asignará a "status" el valor BAT_READ.

Para terminar este último evento, se llamará de nuevo a la tarea *nextStep*.

En la tarea *nextStep*, al comprobar que el "status" es BAT_READ, se le cambiará su valor a READING_PHOTO y se tratará como en el caso de READING_BAT, es decir:

- se llama a *LecturaAdc.read*,
- se trata la interrupción *Atm128AdcConfig.getChannel()*
- se almacena el valor de la luz en el miembro "Luz"
- se cambia "status" a PHOTO_READ
- se llama otra vez a *nextStep()*

(al tratarse el evento *LecturaAdc.readDone*).

Al tener un valor de PHOTO_READ la variable "status", en la tarea *nextStep* se le asignará el valor de READING_TEMP y se llamará otra vez a *LecturaAdc.read()*.

Como en los dos casos anteriores, si el resultado es SUCCESS, indicará que ha almacenado en el miembro de la estructura "Temperatura", el valor de la misma, habrá cambiado el "status" a TEMP_READ y habrá llamado de nuevo a *nextStep()*.

Si el resultado fuera distinto de SUCCESS, entonces asignaría ERROR a "status" y llamaría a *nextStep()*.

De nuevo en la tarea *nextStep()*, con un "status" = TEMP_READ, esto estará indicando que se han leído los 3 sensores (luz, batería y temperatura) por tanto, cambia el valor de "status" a SENSORS_READ y llama a la tarea *nextStep()*.

Cuando desde *nextStep()*, se encuentra que el valor de "status" es SENSORS_READ, se intenta enviar la estructura en la que se han almacenado, entre otros datos, los valores

leídos de los sensores, por RF. Para ello, se hace uso de la función *devuelveDatos()* implementada en la interfaz *devuelveRF* (y que se tratará más adelante en el punto **4º)**. Si la llamada a esta función devuelve ENVIADO, se pone el "status" a DATA_SENT y se ejecuta la tarea *nextStep()*. Si, por el contrario no se recibe ENVIADO, ha habido un error en el envío y por consiguiente no se han enviado. Se pone "status" a ERROR y se ejecuta *nextStep()*;

Una vez que el "status" es DATA_SENT y se entra en la tarea *nextStep()*, se hace un *toggle* del *led* verde, indicando el éxito en el envío de datos de la mota remota a la mota conectada al PC.

Si el estado encontrado en *nextStep()* es ERROR simplemente cambia el mismo a READY.

3º) Con respecto a la pulsación del botón de usuario, el programa trata esta interrupción desde el evento *Button.fired()* y lo que hace es:

a) *toggle* del *led* naranja

b) pone el "status" a READY

c) llama a *nextStep()*

d) asigna al miembro de la estructura "tiempo" el valor de BOTON_PULSADO, de tal forma que desde el otro lado (el programa java) se sepa que ha sido una pulsación de usuario.

Con todo ello lo que consigue es que se envíe el mensaje a través de RF.

4º) La función de envío de la estructura del tipo RadioMsg "*devuelveDatos(RadioMsg*)*" devuelve un entero indicando si el envío se ha podido realizar o no y recibe como parámetro un puntero del tipo RadioMsg.

Básicamente esta función se ocupa de evaluar el contenido de la estructura RadioMsg y enviar o no la misma por RF. Es en este módulo ("devDataRF.nc"), por tanto, donde se podrían cambiar los criterios de evaluación para establecer si se ha de realizar o no el envío de datos en función de los valores contenidos en los elementos de la estructura, que indican tanto los datos obtenidos por la lectura directa de los sensores de luz, batería y temperatura como los datos de valores máximos y mínimos introducidos por el usuario.

Los casos a tratar por la función son 3:

1) que se haya pulsado el botón de usuario
2) que la batería esté en buenas condiciones
3) que la batería esté por debajo del mínimo

Dentro del escenario 1) lo primero que haría la función, sería comprobar que no estuviese "ocupada" la radio, gracias a un booleano "busy", que estará a TRUE (indicando radio ocupada) o a FALSE (indicando radio libre). Acto seguido asigna valores a un puntero de estructura de trabajo del tipo RadioMsg y evalúa el valor del miembro "tiempo" junto al valor de "Bateria". Si el primero de ellos es igual a 1 y el segundo es mayor que 0, la función opta por la pulsación de botón de usuario, pues así se lo indica el valor de "tiempo" (a 1). En cuanto a evaluar el valor de "Bateria" (que sea >0) es para eliminar lecturas erróneas de los sensores y no tratarlas.

Procede al envío de la estructura mediante la llamada a la función "send" de la interfaz "AMSend" que retornará un valor distinto de SUCCESS si no se ha podido realizar con éxito y en ese caso la función "devuelveDatos" devolverá un NO_ENVIADO.

Si por el contrario todo ha ido bien, se pone "busy" a TRUE (para que no se ocupe la radio de momento) y en el evento AMSend.sendDone (que indica el envío del paquete OK) se libera el semáforo de envío restableciéndolo: "busy" = FALSE.

En este caso tratado, la función retornará el valor de "resultado", que como no se ha cambiado, tendrá el valor que se le asigna al inicio de la declaración: ENVIADO.

En el supuesto de que no haya habido pulsación de usuario, se trata el caso 2), es decir, que la batería esté en buenas condiciones.

Como en el caso anterior, la estructura de trabajo tiene los valores necesarios para enviar (en su caso) y el semáforo "busy" está a FALSE. Dado que la batería está en condiciones óptimas, se enviarán los datos por RF, sin valorar si están o no por encima/debajo de los valores de referencia. Para ello, mediante una llamada a "AMSend.send" se inicia el proceso de envío de la estructura (que será el *payload* del mensaje) y se gestionará convenientemente el resultado de la función llamada: si es distinto de SUCCESS se pone "resultado" a NO_ENVIADO. Si por el contrario, es SUCCESS, se trata el evento "AMSend.sendDone" y se libera el semáforo "busy" poniéndolo a FALSE.

El retorno de la función indicará si se ha enviado o no el paquete de datos, es decir, el mensaje.

El caso 3) tiene en cuenta la situación en la cual, el valor obtenido de la lectura de las baterías está por debajo de un mínimo previamente establecido por el usuario al ejecutar el programa "java". En dicho caso, la aplicación de la mota, entraría en un modo de envío de datos (de uso de la radio) más optimizado, dado que sólo procedería a enviar los datos si los mismos estuviesen fuera de rango, si no, no procedería a utilizar la radio para enviarlos.

De esta forma, evalúa el valor de “Temperatura” y la compara con “maxtemp”, a su vez compara “Temperatura” con “mintemp” y también compara “Luz” con maxluz”. Si cualquiera de las tres comparaciones se cumple (es un “OR”) significará que o bien la temperatura máxima se ha excedido, o bien la temperatura mínima se ha sobrepasado (por debajo) o bien se ha excedido el valor máximo de luz permitido y por tanto, comenzará el proceso de envío de datos mediante RF.
Si no se cumplieran las condiciones de la comparación, no se enviaría nada por la radio y el “resultado” sería NO_ENVIADO.
En el caso de recurrir al envío de datos en modo optimizado, se procedería exactamente igual que en los casos anteriores: gestión del valor devuelto por “*AMSend.send*”, semáforo de ocupación de radio, tratamiento del evento “*sendDone*” y liberación del semáforo de la radio. Por supuesto, si el valor de *AMSsend.send* es distinto de SUCCESS, se devolverá un NO_ENVIADO como valor de retorno.

Hasta aquí llega la funcionalidad completa de Recibeorden.
Respecto al encendido/apagado de sus *leds*, se utiliza el naranja para 2 tipos de situaciones diferentes y tendríamos lo siguiente:

- un toggle del **led rojo**--> indica que se ha recibido un mensaje por radio en el evento Receive.receive válido para la aplicación (valor del campo “clave” = 1098)
- un toggle del **led verde**--> indica que se ha enviado un mensaje por RF para ser recibido por el programa del PC
- un toggle del **led ámbar**--> recibido un mensaje en el evento Receive.receive que no es de la aplicación (valor del campo “clave” distinto de 1098)
- un toggle del **led ámbar**--> indica que se ha pulsado el botón de usuario

4.4. El programa Temporizado

Es el programa desarrollado en java, que se ejecutará en el PC.

Consta de los siguientes ficheros:

"RadioMsg.java", "Entrada.java" y "Temporizado2.java" como se visualiza en la figura 20.

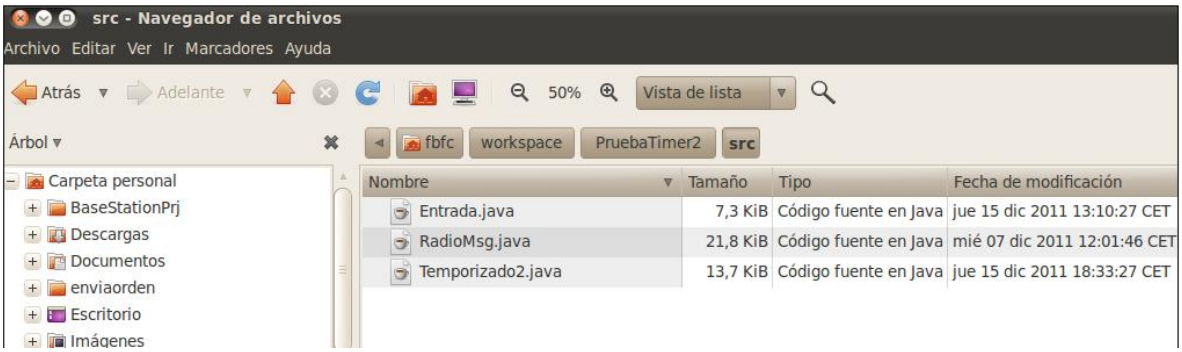

Fig. 20- Ficheros java

- **"RadioMsg.java".** Es un fichero generado por la herramienta MIG (de Tinyos). Dicha herramienta, crea este tipo de ficheros java a partir de una estructura previamente definida (en nuestro ejemplo RadioMsg en el fichero "enviaorden.h"). El contenido del fichero son las funciones necesarias para leer el valor de los campos de la estructura (desde java), para asignar valores a dichos campos, saber si son o no *arrays*, conocer su tamaño en *bits*, etc. En resumen, facilita todo lo necesario para poder leer/escribir en los miembros de la estructura principal de nuestra aplicación, sin tener que programarlos el desarrollador.
  En este proyecto se hace la llamada a la herramienta MIG desde dentro del fichero Makefile utilizado para compilar el programa de la mota:

```
"RadioMsg.class: RadioMsg.java
        javac RadioMsg.java
RadioMsg.java:
        mig java -target=null -java-classname=RadioMsg
        enviaorden.h RadioMsg -o $@"
```

donde se observa cómo se facilita el nombre del fichero java resultante que se usará para compilar con el "javac" que a su vez creará el RadioMsg.class. También se facilita el fichero donde está definida la estructura RadioMsg: "enviaorden.h".

- **"Entrada.java"**. Este fichero se usa para implementar en él la clase que contendrá distintas funciones auxiliares que se necesitarán en el programa principal.
  Dichas funciones son:
  - `public static int LeerEntero(String cadena).` Esta función solicita la entrada de un entero por teclado y lo devuelve como "int". Muestra el texto contenido en el parámetro "cadena" previo a la introducción del dato.
  - `public static String LeerCadena(String cadena).` Esta función solicita la entrada de una "cadena" por teclado y lo devuelve como "String". Muestra el texto contenido en el parámetro "cadena" previo a la introducción del dato.
  - `public static double LeerDouble(String cadena).` Esta función solicita la entrada de un doble por teclado y lo devuelve como "double". Muestra el texto contenido en el parámetro "cadena" previo a la introducción del dato.
  - `public static char LeerCaracter(String cadena).` Esta función solicita la entrada de un carácter por teclado y lo devuelve como "char". Muestra el texto contenido en el parámetro "cadena" previo a la introducción del dato.
  - `public static void MuestraFecha().` Imprime en la pantalla una línea en blanco y a continuación, otra con la fecha actual (sin avance de línea ni retorno de carro).
  - `public static void GrabaEnFichero(String datos).` Graba en un fichero denominado "alertas.log" el contenido del String datos. Le antepone la fecha y hora. El fichero debe encontrarse en el directorio por defecto de la aplicación.
  - `public static void GrabaLineaEnFichero().` Graba una línea en blanco en el fichero "alertas.log".
  - `public static String LeeDatosDelFich(int valor).` Lee datos del fichero "datos.txt", que deberá encontrarse en el directorio por defecto. Según el valor del parámetro "valor", devuelve el contenido de la línea referida a: tiempo entre lecturas de la mota (valor = 0),

temperatura máxima para alertar (valor = 1), temperatura mínima para alertar (valor = 2), luz máxima (valor = 3), o batería mínima para enviar la alerta (valor = 4). En caso de error devuelve un *null*.
Esta función se utiliza para leer los datos máximos y mínimos de un fichero, en vez de solicitarlos por pantalla al usuario.

- **"Temporizado2.java"**. Es el fichero principal del programa, donde se encuentra la función main(). Lo primero que hace al ejecutarse, es comprobar que se ha pasado el argumento con la información necesaria de conexión con la mota (que ejecuta el BaseStation) conectada a su puerto USB. Si no se pasara el argumento, se visualizaría por pantalla un mensaje como:

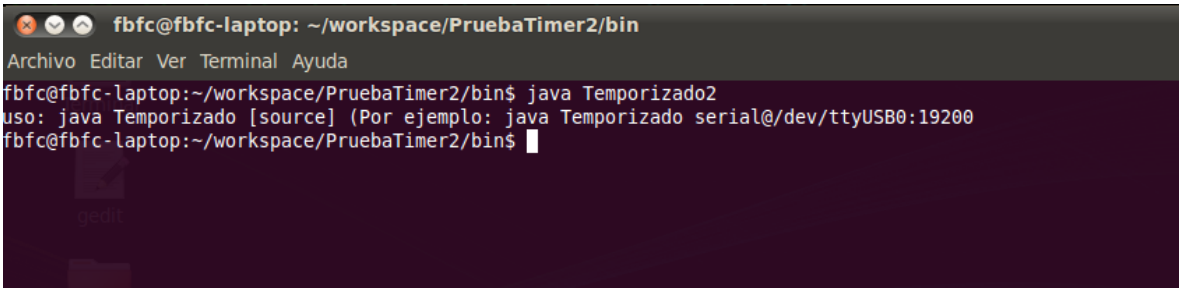

Fig. 21- Ejecución de Temporizado2

esta información es relativa al "port" al que está conectada y a la velocidad de conexión en baudios. En el ejemplo sería el "ttyUSB0" a "19200" bps. Una vez mostrada esta información, saldría del programa, para volver a ejecutarlo pero con el parámetro correcto.
A continuación, crea una instancia de *MoteIf*, utilizada para el envío y recepción de datos al/del puerto serie y comienza con la búsqueda del fichero "datos.txt" en el directorio por defecto. Para ello, llama a la función leeDatosFicheroDatos(). Dicha función con ayuda de la anteriormente descrita "LeeDatosDelFich(int valor)" intenta abrir el fichero "datos.txt" (ver figura 22) y recoger en la variable apropiada el dato especificado en el fichero "datos.txt" para el tiempo en milisegundos que debe transcurrir entre una solicitud de lectura de sensores y otra.
Lo mismo se hará para obtener del fichero la temperatura máxima, pero en este caso debe convertir el valor a un "double", dado que se permite la introducción tanto de la parte entera como de la parte decimal (separadas con un punto). También se ha de convertir este valor a la escala utilizada por el sensor de la mota. Para ello, se utiliza la función "convierteTemp(double)", que recibe como parámetro el "double" que especifica la temperatura máxima y le aplica la fórmula

de conversión apropiada (multiplicándolo por 0.1, sumándole 0.5, y dividiendo entre 0.002348633). Con ello, dispondremos de un valor en la misma escala que utiliza el µC para las lecturas del sensor de temperatura y se podrán hacer las comparaciones oportunas.

Para el caso de la temperatura mínima, por debajo de la cual saltará una alerta, se seguirá el mismo procedimiento que el descrito para la temperatura máxima: recogida del dato en formato cadena, paso a *double* y conversión a escala del sensor.

Continúa leyendo del fichero, el valor asignado para la luz máxima, que no tendrá conversión ninguna (solo de cadena a entero) ya que se trabaja directamente con la misma escala que utiliza el micro controlador. Por último, obtiene del fichero el valor mínimo para considerar que la batería está OK, sin conversión, como en el caso de la luz máxima.

El formato del fichero de valores “datos.txt” es el mostrado a continuación en la figura 22:

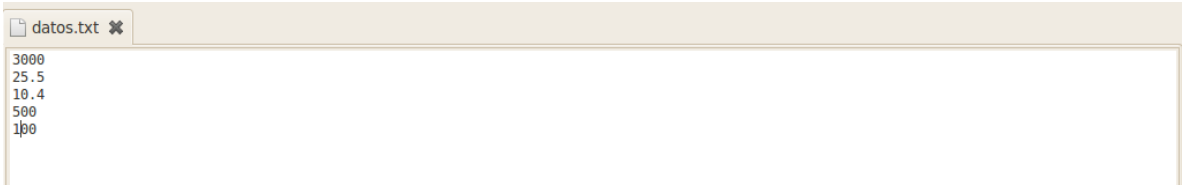

Fig. 22- Fichero

la primera línea contiene el valor para el intervalo de tiempo entre órdenes de lectura de sensores (3000), la segunda la temperatura máxima (25.5), la tercera la temperatura mínima (10.4), la cuarta la luz máxima (500) y la quinta el valor mínimo de las baterías (100).

Si todo ha ido correctamente, la función *leeDatosFichero deDatos*, devolverá un LECTURA_OK, si no, devolverá un LECTURA_NO_OK.

Si devuelve un LECTURA_OK, significa que ya tiene los valores que necesita para enviar a la mota distante. Si no, comienza la solicitud de datos por la pantalla, para que el usuario introduzca a mano los valores, así, en primer lugar, solicitará el tiempo que debe transcurrir entre una orden y la siguiente para la lectura de los sensores, como se representa en la figura 23:

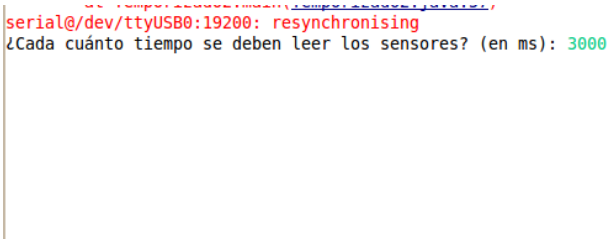

Fig. 23- Toma datos Timer

Acto seguido, solicita la temperatura máxima, por encima de la cual disparará una alerta y que vemos en la figura 24:

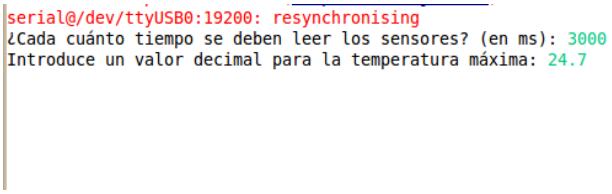

Fig. 24- Toma datos Temperatura máxima

Lo mismo sucederá con el valor para la temperatura mínima, que se aprecia en la figura 25:

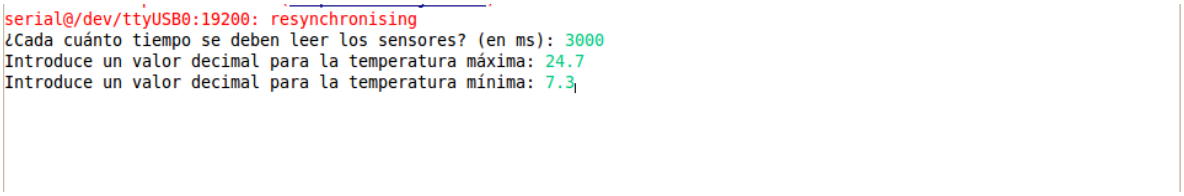

Fig. 25- Toma datos Temperatura máxima

Con respecto a la luz máxima a la que debe estar sometida el dispositivo, la forma de solicitarla es igual que en los casos anteriores (figura 26):

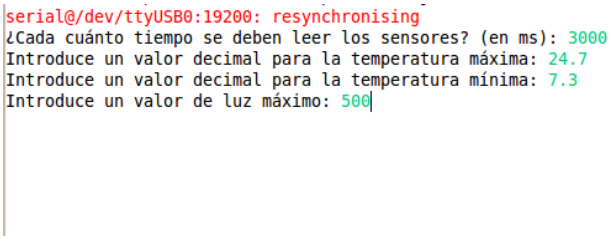

Fig. 26- Toma datos luz máxima

y el valor por debajo del cual, se entenderá que las baterías se están agotando y

obligará al dispositivo remoto a utilizar el envío de datos restringido u optimizado comentado anteriormente se introducirá como se muestra en la figura 27:

```
serial@/dev/ttyUSB0:19200: resynchronising
¿Cada cuánto tiempo se deben leer los sensores? (en ms): 3000
Introduce un valor decimal para la temperatura máxima: 24.7
Introduce un valor decimal para la temperatura mínima: 7.3
Introduce un valor de luz máximo: 500
Introduce el valor mínimo de la batería a partir del cual la mota entra en modo optimizado: 500
```

Fig. 27- Toma datos Batería mínima

Una vez obtenidos los datos, bien haya sido de forma manual o de forma automática a través del fichero, se activa un temporizador que se disparará cada n milisegundos. El valor de n, es el que se ha especificado como intervalo de lectura en msg. para leer los sensores (en la pantalla de ejemplo: 3000).
Con los valores de referencia guardados en las variables correspondientes, el programa graba una línea de inicio en el fichero de "alertas.log", junto a la fecha, hora y valores máximos y mínimos, tal y como se aprecia en la figura 28:

```
Fri Dec 16 11:05:39 CET 2011; Inicio programa. Valores establecidos por el usuario:
Fri Dec 16 11:05:39 CET 2011 Orden de lectura cada: 3000; Temp. máxima: 25.5; Temp. mínima: 10.4; Luz máxima: 500; Batería mínimo: 700
```

Fig. 28- Grabado en log "Inicio programa"

para su posterior consulta.
A partir de este momento, cada vez que se dispare el temporizador transcurrido el tiempo fijado por el usuario, el programa prepara la estructura de datos con sus valores (*clave*=1098, *maxtemp*, *mintemp*, *maxluz*, *minbat*) ayudándose de las funciones creadas por la herramienta MIG en el fichero "RadioMsg.java" (como por ejemplo *set_maxluz* para grabar la luz en el miembro de la estructura *maxluz*, etc.) También pone a cero las variables que indican la pulsación del botón de usuario y de reinicio de aplicación (*counter* y *tiempo*) e intenta el envío en un bloque *try...catch*. Si el envío se ha realizado con éxito, muestra un "." en la pantalla. Por este motivo, en la pantalla, siempre y cuando no se muestren alertas, se verán muchos "puntos" seguidos (uno por cada vez que se ha disparado el temporizador y se han enviado los datos). Estos puntos nos ayudan

a saber que el programa está funcionando aunque no se visualice nada más en la pantalla (porque no tiene que generarse ninguna alerta).
Conviene recordar que este envío de datos es al puerto serie y desde allí, es el programa instalado en la mota conectada al ordenador (BaseStation) el que lo transmite por radio para que lo reciba el dispositivo remoto.
Si hubiese algún error en el intento de envío de la estructura al puerto serie, se visualizaría un mensaje por pantalla: `("Error en el envío del mensaje de lectura a la mota...");`
Por tanto, este ciclo de envío de los datos al puerto serie, se repite indefinidamente hasta que se cierre el programa, es decir, el dispositivo remoto está recibiendo cada n milisegundos, la orden de lectura de sensores junto a los valores de referencia.
Por otro lado, el programa java está siempre a la espera del evento de recepción de mensajes por su puerto serie (al que se ha conectado la mota) y por tanto, cada vez que se reciba un mensaje, lo podrá tratar convenientemente. El método en el que se gestiona esta circunstancia se denomina *messageReceived*. En realidad, para no tratar mensajes antes de que se hayan establecido convenientemente los valores de referencia y se hayan enviado al dispositivo remoto, se utiliza un semáforo (*"comenzar"*) para indicar cuándo se debe atender a este tipo de mensajes de entrada.
En primer lugar, verifica que el mensaje se corresponda con el tipo esperado, es decir *RadioMsg*, para mostrar una línea de aviso si no lo fuera: `"Mensaje de tipo desconocido"` y continuar a la espera de un nuevo evento de entrada de mensaje.
Si el mensaje sí es del tipo *RadioMsg*, comprobará el valor de "clave" que como ya se ha comentado, sirve para filtrar mensajes con el contenido corrompido. Si dicho valor es igual al de la constante "clave" o sea, 1098, se continuará tratando el mensaje. Si no, no haría nada, continuaría pendiente de la recepción de otro mensaje.
Supuesto que la clave está OK, se comprueba si es o no un mensaje de reinicio de dispositivo remoto. Para saberlo, se evalúa "counter", si es =1 indicará que sí, que es un mensaje enviado por la mota distante indicando que se ha iniciado/reiniciado, por tanto, el programa java muestra en pantalla `("Reinicio del dispositivo remoto... Se reenvían los valores`

máximos y mínimos" y graba en el fichero de alertas una línea similar indicando tal situación (figura 29):

```
serial@/dev/ttyUSB0:19200: resynchronising
............
Fri Dec 16 11:00:40 CET 2011 Alerta de temperatura fuera de rango. Debe estar entre: 12.4 y 25.5; Valor leído: 25,86
..............
Fri Dec 16 11:01:26 CET 2011 Boton de usuario pulsado, datos: Bateria: 577; Luz: 192; Temperatura: 22,10
............Reinicio del dispositivo remoto... Se reenvían los valores máximos y mínimos
............
```

Fig. 29- Grabado en log "reinicio dispositivo"

Si no fuera un aviso de reinicio, comprueba si se ha tratado de una pulsación de botón de usuario leyendo el valor de "*tiempo*" que, si como en el caso anterior, es 1, indicará que sí se trata de este caso. Así, mostrará en pantalla el aviso correspondiente a la figura 30

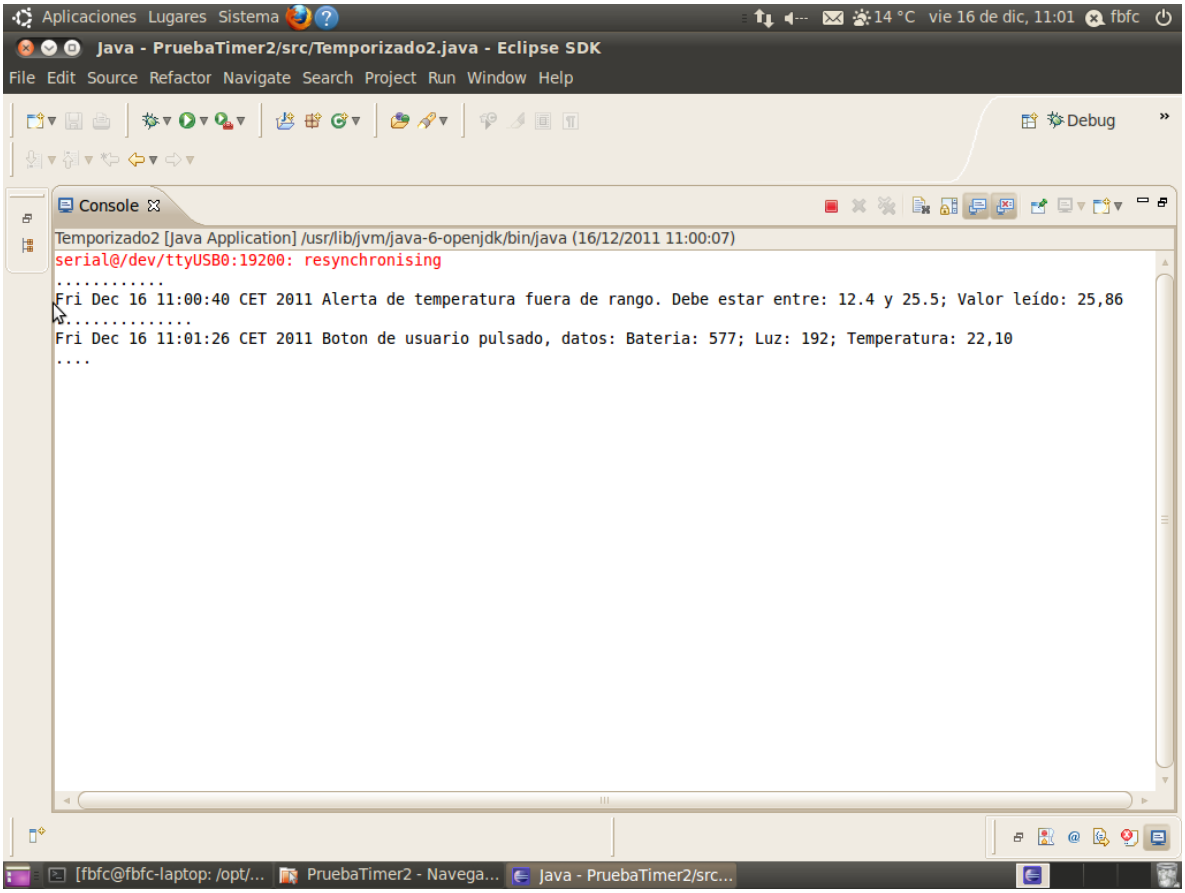

Fig. 30- Alerta botón de usuario

junto a los datos de los sensores. También grabará en el fichero "alertas.log" el

aviso correspondiente, como vemos en la figura 31.

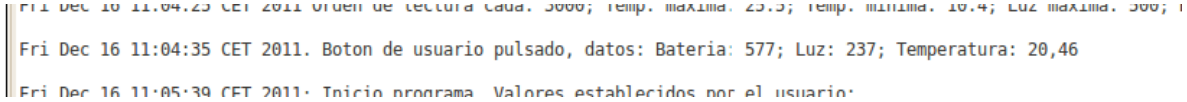

Fig. 31- Grabado en log “Botón de usuario”

Esta utilidad de pulsación de botón de usuario, por tanto, servirá para obtener una lectura inmediata de los sensores y dejar registrados los datos en el fichero de alertas, así como para saber si la mota está funcionando o no en un determinado momento.

Durante la comprobación de valores, se evalúa también que el voltaje de la batería sea >0 (no tiene ningún sentido que el valor sea<=0, dado que no tendría voltaje suficiente ni para enviar los datos, por eso se presupone que es un error en los datos), dado que de vez en cuando se envían datos del sensor que no tienen la información real.

En el caso de que no se haya pulsado el botón de usuario, lo que se comprueba es el valor de las baterías, para saber si están bajo mínimos (figura 32). Para ello utiliza el método get_Bateria y compara su resultado con minBat.

Supuesto que get_Bateria es menor, muestra un mensaje de aviso por pantalla, indicando que a partir de ese momento sólo se recibirán envíos por RF si alguno de los valores está fuera de margen:

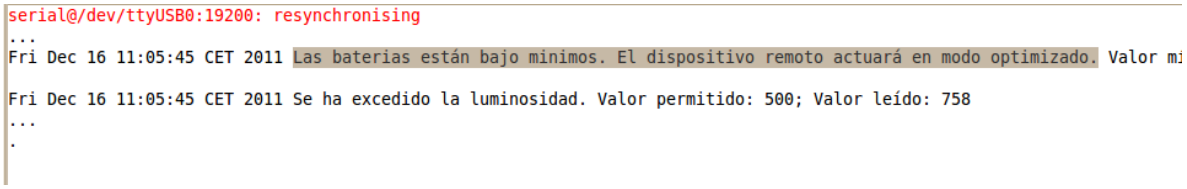

Fig. 32- Grabado en log “Batería bajo mínimos”

también grabará en el fichero de log “alertas.log” una línea con el detalle anterior.

Una vez evaluada la situación anterior, evalúa la temperatura, es decir, que la temperatura esté entre la mínima y la máxima indicadas. Para ello se vale de las función get_Temperatura() (creada automáticamente mediante MIG en el fichero RadioMsg.java) , cuyo valor comparará con maxTemper y con minTemper, que representan respectivamente la temperatura máxima y la temperatura mínima. Si la temperatura obtenida del sensor no está en ese rango, se visualizaría en la pantalla algo similar a lo mostrado en la figura 33:

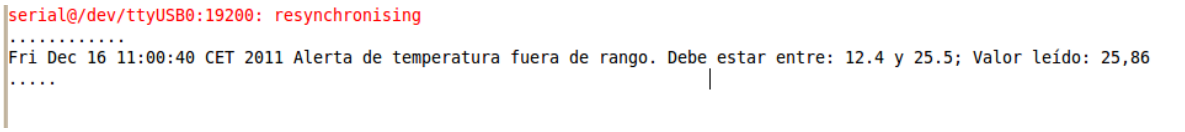

Fig. 33- Alerta temperatura máxima excedida

para el caso de un exceso de temperatura, y:

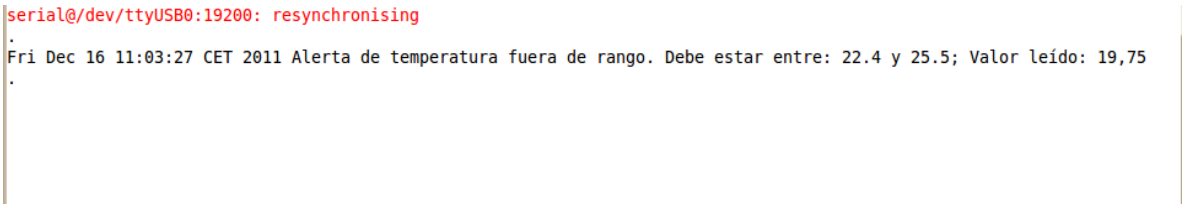

Fig. 34- Alerta temperatura mínima excedida

si la temperatura está por debajo de la mínima indicada.

Por supuesto, esta situación, como se puede apreciar en la figura 35, quedaría registrada convenientemente en el fichero de alertas.log:

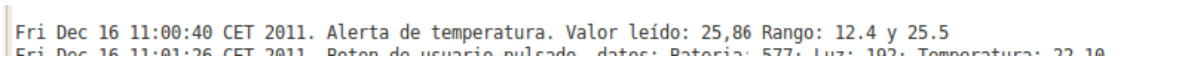

Fig. 35- Grabado en log “Alerta de temperatura”

Por último, se evalúa la luz. Gracias a la comparación entre get_Luz() y la variable maxLuz, el programa podrá determinar si se ha sobrepasado el valor de luz máxima que el usuario decidió en el inicio del programa. Si así fuera, se visualizaría en pantalla como la figura 36:

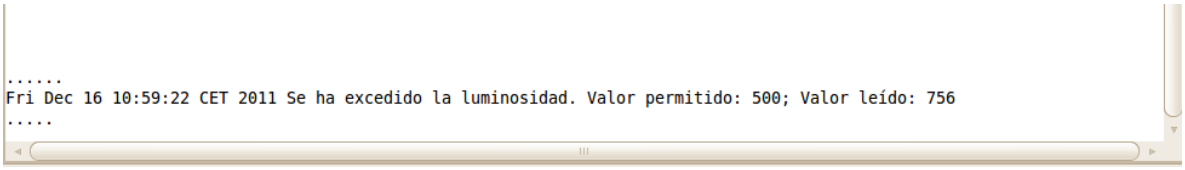

Fig. 36- Alerta luminosidad máxima excedida

y se grabaría en el fichero de log (figura 37):

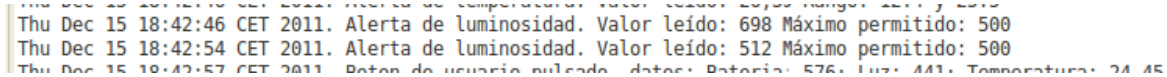

Fig. 37- Grabado en log “luz excedida”

A partir de aquí, se vuelve a la situación de espera, bien hasta que haya vencido el temporizador y se vuelva a enviar la información relativa a valores máximos y mínimos, bien hasta que se reciba un nuevo mensaje del dispositivo remoto, para su evaluación y posible aviso de alerta.

5. Viabilidad técnica

El proyecto desarrollado es perfectamente viable y se puede utilizar para controlar los valores de temperatura y luz de un solo equipo cualquiera (por ejemplo una nevera) y recoger los resultados en cualquier ordenador siempre y cuando cumpla los requisitos de instalación descritos.

6. Valoración económica

Como en muchos de los supuestos económicos, esta valoración es aproximada, dado que los precios en el mercado varían mucho de un proveedor a otro y, por supuesto, pueden depender de la cantidad de elementos adquiridos.
No obstante, como la partida más importante es la derivada del desarrollo, que conllevaría unos 3 meses de trabajo (5 horas diarias * 22 días * 3 = 330 horas), el presupuesto no estaría tan desviado de la realidad.
En cuanto al precio de las motas, se ha obtenido un precio aproximado de la página web http://www.advanticsys.com/cm5000.html?gclid=CMru2fjQpK0CFUQMfAodtl1Fng

La figura 38 muestra un posible desglose de precios y su montante total.

Concepto	Cantidad	Importe unitario	Total
Motas	2	70	140
Sistema Operativo Ubuntu	1	0	0
Sistema Operativo Windows	1	150	150
PC	1	500	500
Tinyos	1	0	0
Editores de texto	1	0	0
Entorno de programación Eclipse	1	0	0
Desarrollo (horas)	330	50	16500
TOTAL			**17290**

Fig. 38- Presupuesto

7. Conclusiones

7.1. Conclusiones

Los objetivos básicos, así como los secundarios y otros que han ido surgiendo a lo largo del desarrollo del proyecto, se han podido cumplir sobre todo, gracias al aprendizaje del nuevo lenguaje de programación de las motas y de su forma de enlazar unos módulos con otros para aprovechar (reutilizar) funcionalidades ya desarrolladas. También ha sido necesario aprender y dejar perfectamente operativo el sistema Tinyos y el entorno de trabajo (Eclipse, Gedit, etc.) bajo el sistema operativo Ubuntu, cuestión ésta que se ha podido llevar a cabo no sin pocos problemas técnicos que se pudieron solucionar convenientemente.

7.2. Propuesta de mejoras

La mejora más importante sería añadir por un lado la funcionalidad para identificar cada mota, es decir, grabar en el momento de cargar el programa en la mota, el número que identifica a cada una. Esto implicaría modificación en algunas de las funciones de la aplicación, pero bastante sencillas. Por otro, en el fichero “datos.txt” habría que añadir una línea especificando que los datos de valores máximos y mínimos que se reflejan a continuación en las líneas siguientes del fichero, se corresponden con la mota cuyo valor figura en esa misma línea. O mejor aún, utilizar una tabla de una base de datos, dado que el volumen de información a almacenar sería ya considerable para manejarla en un simple fichero de texto.
De esta forma, se podrían gestionar muchos dispositivos en vez de uno solo.
Otra mejora importante, sería poder almacenar información en memoria del dispositivo. Así, en caso de un reset, en vez de obligar al programa “java” a enviar continuamente los valores (una vez disparado su temporizador) la mota podría consultar esos valores y “recuperarse” directamente. No obstante, esta mejora no sé si realmente merecería la pena, dado que tal y como se ha diseñado se cubre la funcionalidad sin ningún problema.
El desarrollo de una aplicación WEB para consultar los resultados de las lecturas sería sin duda la evolución natural de este proyecto hacia su proyección en el mercado, dado que este tipo de funcionalidad es básica hoy en día.
También habría que contemplar el envío tanto de *sms* como de *e-mails* para alertar de los valores fuera de rango.
Estos dos últimos apartados serían bastante sencillos de llevar a cabo, sobre todo si la información se vuelca en una base de datos, como se ha comentado anteriormente y su desarrollo totalmente independiente del conocimiento de Tinyos ni NesC, dado que bastaría tratar con los resultados guardados en BBDD para llevar a cabo los programas.

7.3. Autoevaluación

Lo aprendido a lo largo del desarrollo de TFC ha sido muy interesante. Si bien y debido sobre todo a la falta de tiempo y a la presión de entregar a tiempo los objetivos marcados para cumplir con el TFC, a veces ha llegado a ser un poco estresante, la realidad es que esta nueva tecnología deja muchas oportunidades abiertas para que se puedan abordar con éxito a lo largo de los años venideros proyectos verdaderamente interesantes en el mercado.

Gracias a haber trabajado en este entorno, es probable que continúe con el desarrollo de este tipo de sistemas inalámbricos, tanto a nivel de hardware (añadiendo nuevos sensores, memorias, etc.) como a nivel de software (construyendo nuevas librerías –interfaces- para conseguir un desarrollo mucho más rápido).

8. Glosario

- ⇒ ***Array***: una matriz o vector. Es una zona de almacenamiento contiguo, que contiene una serie de elementos del mismo tipo.
- ⇒ ***BaseStation***: utilidad facilitada por Tinyos para enviar por RF lo que recibe por el puerto serie y para enviar por el puerto serie lo que recibe por RF
- ⇒ ***Eclipse***: entorno de desarrollo integrado para programar en java (principalmente)
- ⇒ ***Gedit***: editor de textos bajo SO Linux
- ⇒ ***jar***: extensión que se utiliza en determinados modos de compresión de archivos
- ⇒ ***LDR***: resistor dependiente de la luz, es decir, que varía el valor de su resistencia en función de la intensidad de la luz que recibe
- ⇒ ***leds***: diodo emisor de luz
- ⇒ ***Master-Slave***: protocolo de comunicación en el cual uno de los dispositivos es el master y controla a otros dispositivos denominados slaves
- ⇒ ***Meshprog***: utilidad facilitada por Tinyos para enviar el programa desarrollado a la mota
- ⇒ ***MIG***: herramienta facilitada por Tinyos para generar los archivos java (o C) de manera automática a partir de una estructura de datos. Dichos archivos contendrán los métodos necesarios para acceder a los miembros de la estructura en cuestión
- ⇒ ***mota***: pequeño computador para obtener lectura de sensores a distancia. Conocido también como “smartdust”
- ⇒ ***NesC***: lenguaje de programación muy similar al C utilizado para programar las motas en el entorno de Tinyos
- ⇒ ***pdf***: cierto tipo de archivos, comúnmente utilizado por su acceso de lectura exclusivamente
- ⇒ ***payload***: parte principal de un mensaje (desechando cabeceras, longitudes, etc.)
- ⇒ ***reset***: puesta de un sistema en sus condiciones iniciales
- ⇒ ***RF***: radio frecuencia
- ⇒ ***timer***: temporizador
- ⇒ ***Tinyos***: sistema operativo de código abierto basado en componentes, diseñado para plataformas de redes de sensores inalámbricos
- ⇒ ***toggle***: acción de cambiar el estado, si está abierto a cerrado y viceversa (o de encendido a apagado y viceversa)
- ⇒ ***WDT***: watchDogTimer, sistema basado en un temporizador cuya misión principal es hacer un reset en un dispositivo cada cierto tiempo. Se utiliza para reiniciar los dispositivos en caso de caída

9. Bibliografía

Página de la UOC MediaWiki
http://cv.uoc.edu/app/mediawiki14/wiki/P%C3%A0gina_principal

Página principal de Tinyos
http://www.tinyos.net/

Manual de programación de Tinyos:
http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf

Programación en NesC:
http://nescc.sourceforge.net/papers/nesc-ref.pdf

Tutoriales de Tinyos:
http://docs.tinyos.net/tinywiki/index.php/TinyOS_Tutorials

Datasheet del ATmega1281 (de ATMEL)
http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf

Sensor de Luz:
http://www.advancedphotonix.com/ap_products/pdfs/PDV-P9003-1.pdf

Sensor de temperatura:
http://ww1.microchip.com/downloads/en/DeviceDoc/21942e.pdf

Utilizar WDT:
http://atmel.com/dyn/resources/prod_documents/doc2551.pdf

10. Anexos

10.1. Ejecución y Compilación

Respecto al software para instalar en la mota remota, se envían los siguientes archivos: "***MuestreaP.nc***", "***RecibeOrdenP.nc***", "***RecibeOrdenC.nc***", "***enviaorden.h***", "***devuelveRF.nc***", "***devDataRF.nc***", "***Watchdog.nc***", "***WatchdogC.nc***" y "***Makefile***".

Al ejecutar "*make cou24*" desde el entorno de Tinyos, se crearán primero 2 archivos: "***RadioMsg.java***" y "***RadioMsg.class***" (gracias a la herramienta MIG), que luego habrá que copiar al entorno java. A continuación se compilará el programa en nesC y quedará listo para grabarlo en la mota mediante MESHPROG, por ejemplo: "*meshprog -t/dev/ttyUSB0 -f ./build/cou24/main.srec*"

Respecto al software java, que se ejecutará en el PC, tendremos los siguientes archivos: "***Entrada.java***", "***RadioMsg.java***", "***Temporizado2.java***"y "***datos.txt***", este último deberá estar dónde se ejecute el programa. Según el entorno de desarrollo que utilicemos, se compilarán estos archivos (menos RadioMsg.java, que ya se hizo en el paso anterior) y se crearán los ".class" correspondientes.

Una vez listos los archivos, se ejecuta el programa llamando al módulo java que contiene la función "main", en este caso: "Temporizado2".

Como argumento habrá que especificar la conexión de la mota con el BaseStation al ordenador, por ejemplo: "***java temporizado2 serial@/dev/ttyUSB0:19200***". En cualquier caso, el programa avisa si no se ha pasado dicho argumento y muestra un ejemplo:

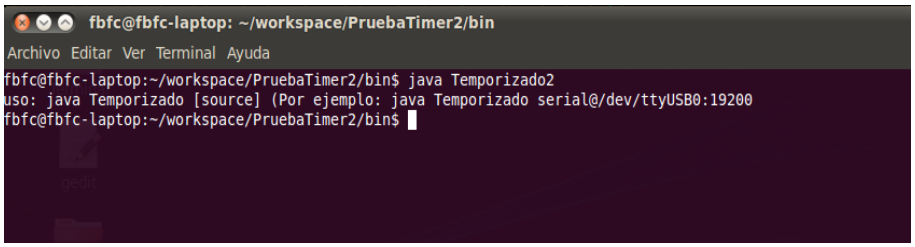

Fig. 39- Ejecución de Temporizado2 sin argumentos

Por supuesto, habrá que instalar el programa BaseStation en la mota conectada al PC.

Es importante que el usuario que ejecuta el programa java tenga **derechos de creación y lectura de archivos** en el directorio donde se hallen los ficheros java que se ejecutan, ya que el programa creará si no existe un archivo (alertas.log) como el de la figura 40 y accederá al mismo para grabar nuevos datos.

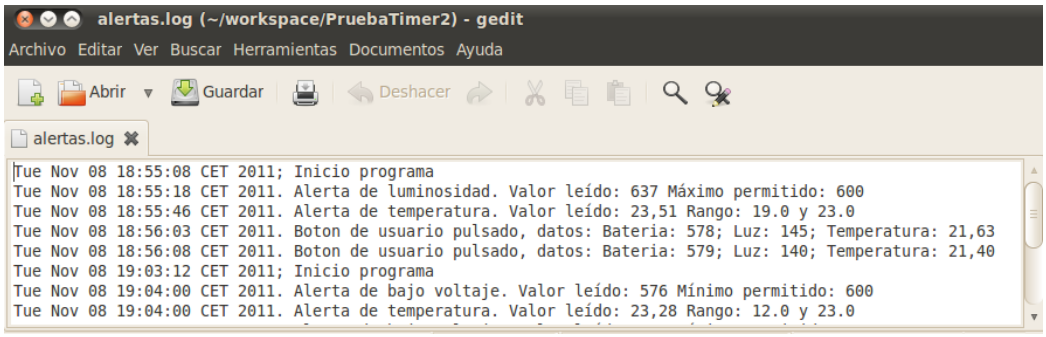

Fig. 40- Fichero de alertas