Trabajo Fin de Carrera

Control de voltaje y temperatura remota mediante motas.

Luis Manuel Villaverde Gómez
Ingeniería Técnica de Informática de Sistemas

Sebastià Cortes i Herms
Consultor

1.- Introducción

Voy a realizar una exposición de motivos explicando porque me he decidido por esta opción y no por otras que podría haber escogido del listado de posibilidades que se nos ofrecía para realizar el Trabajo Fin de Carrera (TFC).

El semestre pasado ya me había matriculado en el TFC y había optado por la especialidad de Sistemas Empotrados. Pero, por un cúmulo de circunstancias no me fue posible pasar de la fase de explicar a mi tutor cual era mi idea de trabajo.

Pasado el semestre y con menos problemas y ya con todas las asignaturas aprobadas, ya solo me queda finalizar este trabajo para obtener el tan ansiado título. Tenía en mente seguir con esta opción o la de realizar un trabajo sobre informática forense, que es algo que me atrae bastante también, pero no fue posible. Así que, aquí sigo de nuevo con la idea original de realizar el TFC con los sistemas empotrados.

Al principio, cuando examinaba las posibilidades, no tenía muy claro lo que era eso de "sistemas empotrados". E investigando un poco me atrajo la idea. Después de leer bastante y ver diferentes definiciones de lo que es, yo me quedo con una que es una mezcla de todas ellas. Así que yo definiría un sistema empotrado como el sistema que usa un ordenador para una tarea específica, pero que lo usamos y no lo vemos, ni asociamos a que es un ordenador.

Así que tenemos un "ordenador", donde tanto la entrada como salida de los datos no es al estilo tradicional, con las interfaces a las que estamos acostumbrados (teclado, pantalla, ratón). Ni la apariencia del "ordenador" es la habitual. Por lo que, los sistemas empotrados son un conglomerado heterogéneo de dispositivos. Y pueden ser tan diferentes, como que la interface de comunicación sea un led; o llegar a ser tan complejo como un sistema de visión artificial.

Los sistemas empotrados estarán diseñados en función del uso que se les vaya a dar, por lo tanto, pueden ser desde pequeños dispositivos que ocupan unos pocos centímetros cuadrados, a ocupar paneles y tener conectados multitud de sensores para obtener datos que usarán para su procesamiento.

La idea principal de los sistemas empotrados es que tienen que ser

1. Muy robustos, pues tendrán un uso industrial y estarán en entornos duros.
2. Fiables: ya que funcionarán autónomamente y el número de errores tiene que tender a cero.
3. Precio reducido: se deben reducir los costes a lo mínimo posible y así tener éxito comercial.
4. Tamaño reducido y consumo reducido: como en la mayoría de casos trabajarán en ambientes industriales, cuanto menos ocupen y menos consuman, más posibilidades de éxito tendrán para ser autónomos.

Los sistemas empotrados los tenemos delante todos los días y los usamos a diario y no somos conscientes de ellos, ejemplo de ellos son las lavadoras, cámaras digitales, impresoras, sistemas de frenado asistido de coches, etc.

En este caso nosotros vamos a usar el hardware facilitado por el departamento y el aportado por mi para el correcto desarrollo del mismo, y que será explicado más adelante.

2.- Descripción detallada de la aplicación a realizar

La aplicación permitirá obtener (a través de las motas[1]) información acerca de temperatura, índice de luminosidad ambiental, así como del estado de sus baterías. Dicha información será almacenada en el servidor y podrá ser consultada en tiempo real a través de una pequeña aplicación web, donde podremos comprobar dichos datos. Además se simulará mediante los leds, la activación-desactivación de dispositivos, como si fuera un relé, para emular el encendido-apagado de luces y actuación como un termostato de un sistema de climatización.
Como “bonus” se intentará diseñar un widget para Android que tenga la misma funcionalidad que la aplicación web. Todo será en función de la curva de aprendizaje que tenga todo el sistema, ya que hace muchos años que no programo.
En la figura 1 podemos ver un esquema de como es el sistema, podría tener más motas como aparece en la figura, pero nosotros solo disponemos de dos.

[1] Mota: En inglés es MOTE, acrónimo de Mobile Transmission Elements. En el lenguaje coloquial una mota es una partícula de algo que se acumula. Y este dispositivo también es como una pequeña partícula.

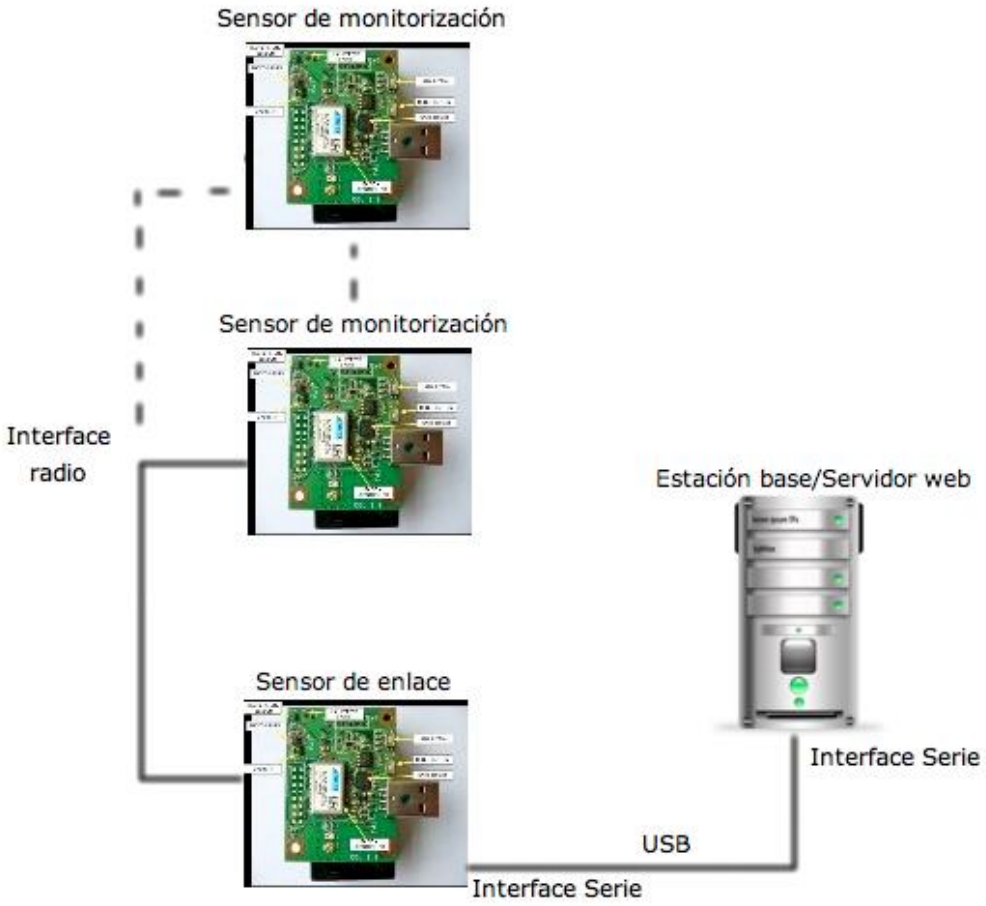


Figura 1 - Esquema general del sistema

El sistema podría extenderse para un mayor número de motas, con lo que podría llevarse el control de varios locales independientes.

Los sensores de monitorización facilitan su estado (luz, temperatura, batería) a través del interface de radio a la mota de enlace, que a su vez facilita dichos datos al servidor a través de la conexión USB.

En dicho servidor se almacenarán los datos en un archivo. Podría ser una pequeña base de datos, ya que he leído algo de TinyDB, que tengo por investigar. Y desde una pequeña aplicación web dichos datos podrán ser vistos a través de internet en un dominio que he montado al efecto, http://www.probando.com, y desde el cual podremos interactuar.

Finalmente y si logro conseguir una buena documentación y sobre todo tiempo, realizaría un widget sobre Android para poder ver esos mismos datos desde el móvil.

2.1.- Justificación

Por una parte tenemos la justificación “académica”, ya que durante los años de formación en la carrera hemos visto todos los procesos que debemos llevar a cabo en un proyecto informático. Ahora es el momento de plasmar todo el conocimiento adquirido, desde un punto de vista más holístico, ya que intentamos abarcar todos los pasos. Aunque en la vida real, lo más probable es que estemos inmersos en un proyecto que incluye más personal y departamentos.

Y por otra tenemos la justificación “operativa”, donde tratamos de ver como mediante la utilización de un hardware y un software que desarrollaremos podemos interactuar con sistemas para la mejor gestión energética. Este tipo de gestión nos permite la automatización de edificios para además de mejorar la eficiencia energética, mejoraríamos también el confort de los usuarios.

La eficiencia energética sería posible al poder determinar individualmente por áreas la temperatura deseada y gestionarla de una manera racional. Ya que a veces nos encontramos con sistemas de climatización que mantienen la misma temperatura en toda la planta, cuando a lo mejor en un ala de dicha planta está dando el sol de lleno y podría estar la calefacción desconectada, mientras que en otra es zona sombría y se están muriendo de frío, pero el termostato está justo en la zona soleada y no funciona la climatización. También serviría para detectar la luz ambiental y en función de la hora del día podría encender-apagar el alumbrado.

En un momento en que la crisis afecta a todos los sectores, un mayor control en el gasto, incrementa las posibilidades de supervivencia de una empresa e incluso de una familia.

La utilización de motas, permite un despliegue rápido ya que son inalámbricas, de muy bajo consumo y podremos colocarlas donde mejor servicio nos den. Por lo que su elección no es azarosa.

En resumen, mediante la aplicación web podremos saber la temperatura y luminosidad de una habitación, y ello nos permitiría interactuar con otros sistemas para gestionar el alumbrado y climatización.

3.- Descripción funcional.

El sistema consiste en realizar lecturas del ambiente a través de las motas, obteniendo el resultado cada determinado tiempo, se puede variar ese tiempo en la configuración de las motas. Así como obtener el estado de su batería.

En este caso actuarán como dispositivos sin inteligencia propia, y la inteligencia se la dejaremos a la aplicación web, que se podrá mejorar con más opciones. Y así no nos limitamos a los cálculos exclusivos de las motas.

3.1.- El sistema completo

Los sistemas empotrado permiten realizar tareas limitadas de forma eficiente, con un tamaño reducido, a un coste asequible y con posibilidad de situarlo en multitud de ubicaciones, así como en maquinaria industrial. Por lo que nos permite realizar mediciones de una manera segura. Y al hacerlo vía radio nos amplia más dichas posibilidades, ya que nos ahorra el cableado que requieren otros sistemas que cumplen con los mismos fines.

Las motas envían la información por radio utilizando el protocolo ZigBee a 2,4 Ghz a una dirección “broadcast”, así que la información la reciben todas las radios que sintonicen esa frecuencia.

El sistema está dividido en varias partes, tal como indicábamos en la Figura 1.

Por un lado tenemos la parte que denominaremos “EMISOR” que es la dedicada a la obtención de datos en las motas y que envía vía radio a la “BASE” que se comunica por el puerto serie/USB con el servidor. Y ahí es donde procesaremos los datos en la aplicación web, que se desarrolla con HTML5.

En las próximas páginas iré desarrollando los distintos módulos con una explicación rápida de lo que hace cada uno, primeramente mostrando la figura general y posteriormente cada uno de los módulos.

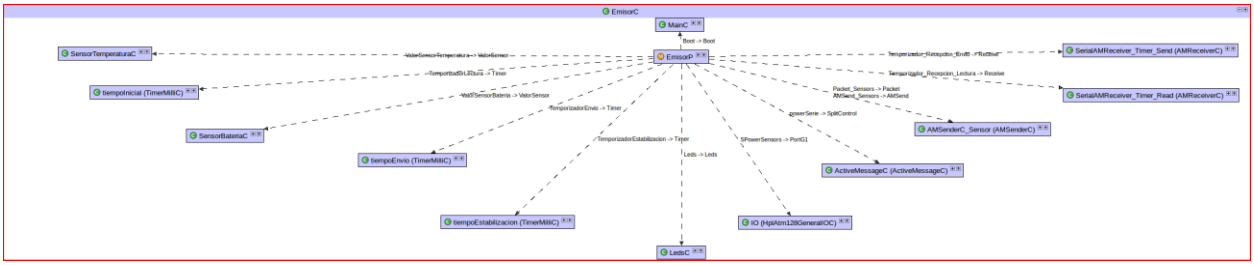


Figura 2 – Vista reducida del sistema EMISOR

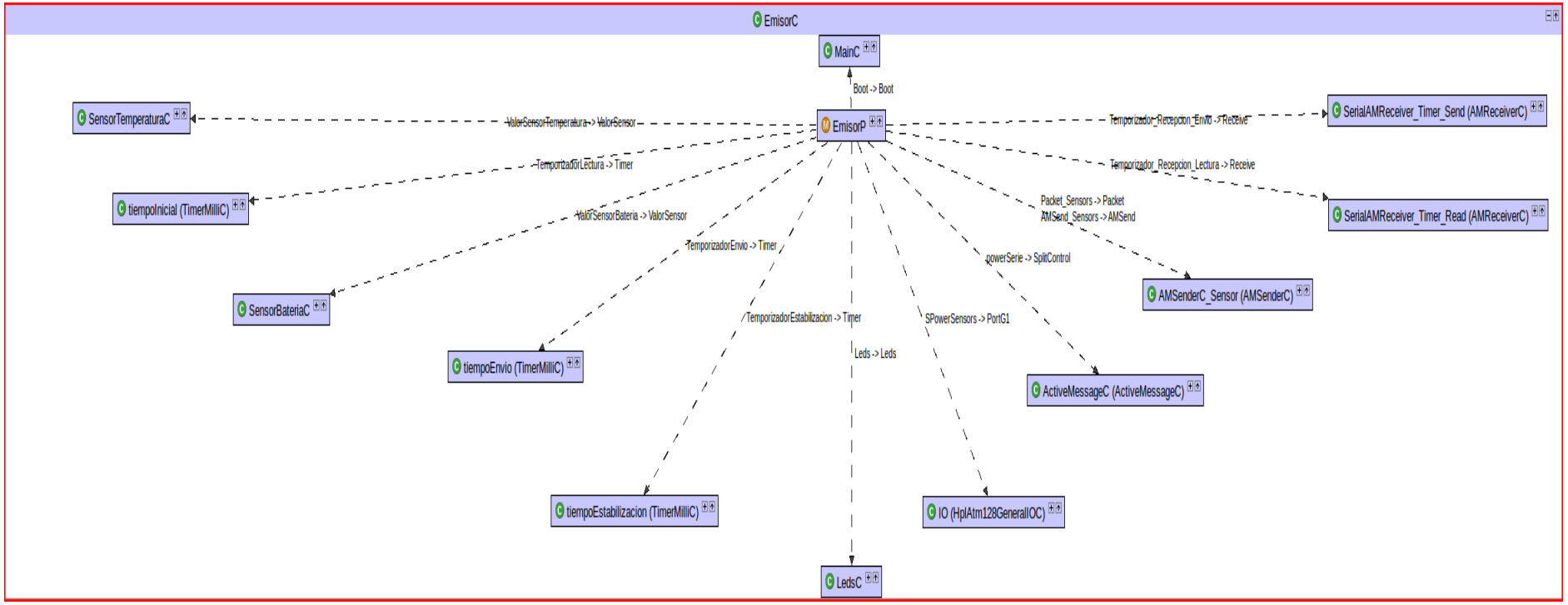


Figura 3 - Vista ampliada del sistema Emisor (EmisorC.nc)

Siguiendo la notación que se indica en la documentación sobre NesC y TinyOS , los módulos tendrán sus nombres terminados en P, ficheroP.nesc; y los ficheros de configuración, estarán terminados en C, ficheroC.nesc; y los archivos de definición de cabeceras, llevarán la extensión habitual ".h".

Ahora los iré describiendo uno a uno, y cada archivo de todas formas va con sus correspondientes comentarios, para que sean más entendibles, incluso para mi ya que es un sistema completamente nuevo y a veces me produce algo de confusión.

3.2.- El emisor

En el emisor tenemos dos **temporizadores** principales:

- *TemporizadorLectura* (-> tiempoInicial): Temporizador para indicar los tiempos de lectura, se puede modificar con la constante TEMPO_LECTURA. Para las lecturas de sensores.
- *TemporizadorEnvio* (-> tiempoEnvio): Temporizador que nos sirve para indicar el periodo de envío, se modifica con la constante TEMPO_ENVIO. Envía en cada período el mensaje a la mota base a través de ActiveMessages.

Y luego una serie de **eventos** cuando se reciben los mensajes:

- *Temporizador_Recepcion_Lectura.receive*: Recibe el tiempo de modificación de las lecturas y se modifica
- *Temporizador_Recepcion_Envio.receive*.receive: Recibe el tiempo de modificación de los envíos y se modifica.

EmisorC.nc:

Fichero principal del sistema, donde se arma el sistema, sería el módulo principal. Indicamos que componentes se usan y es donde se hace el enlace, "wiring". Podemos observalo en la figura 3, de manera más detallada y en la figura 4 a pequeña escala.

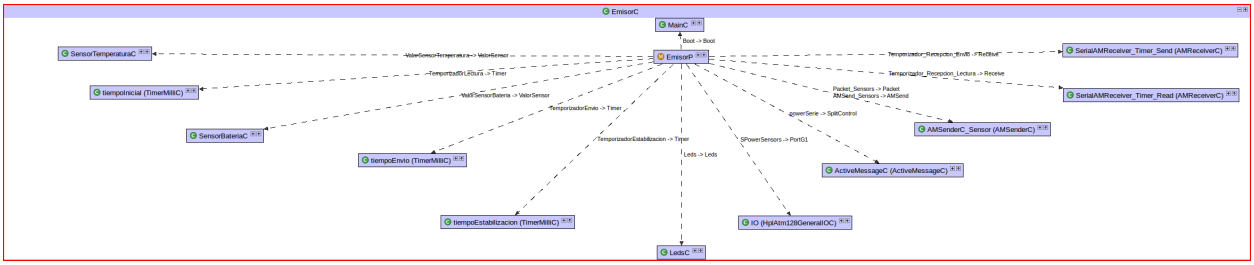


Figura 4 – Vista reducida del sistema EMISOR

EmisorP.nc:

Parte privada del componente donde se hace la gestión principal de la aplicación

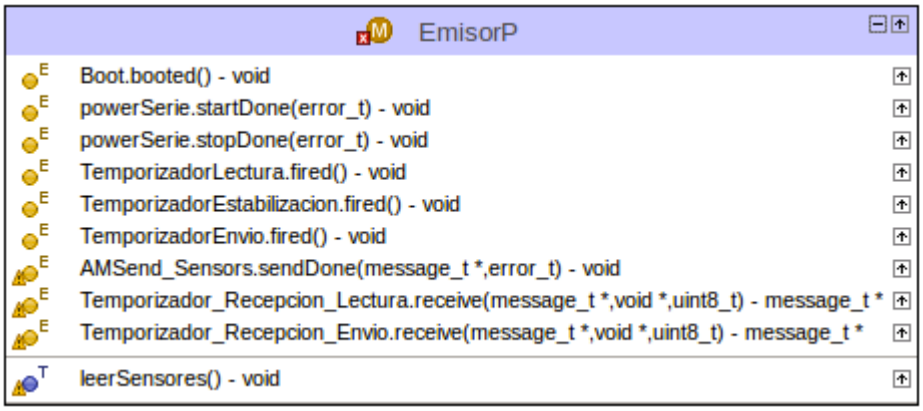


Figura 5 - EmisorP.nc

Ahora describiré los componentes principales, y mostraré su figura correspondiente.

MainC

Indica a la aplicación el arranque (boot) correcto de la mota.

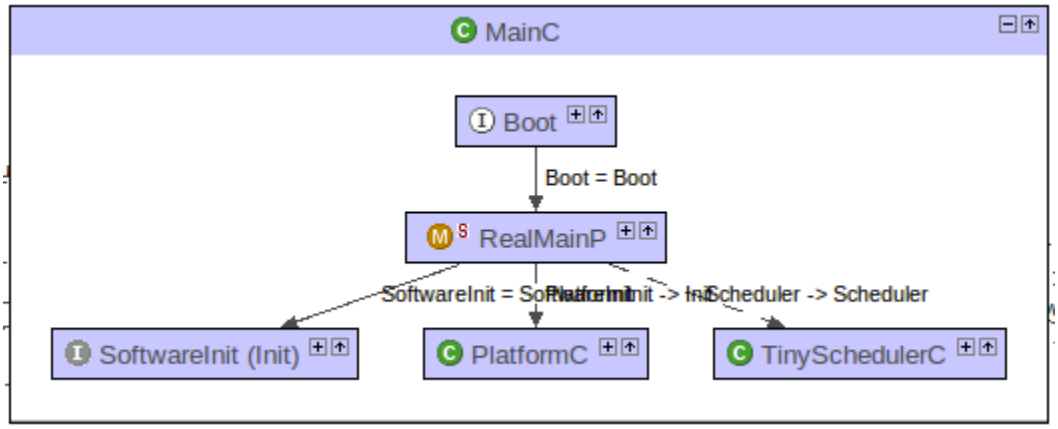


Figura 6 - MainC

TimerMilliC:

Nos permite crear los temporizadores para poder leer los sensores y envíar los valores, así como su sincronización.

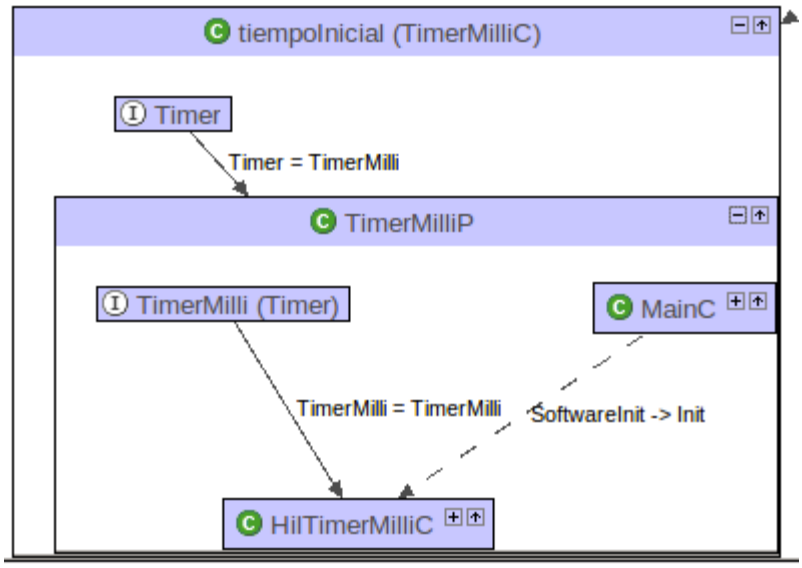


Figura 7 - Ejemplo de temporizador

LedsC:

Nos permite realizar el encendido/apagado/conmutación de los leds de las motas.

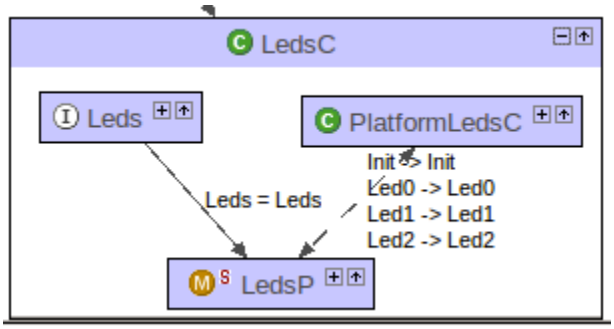


Figura 8 - LedsC

HplAtm128GeneralIOC:

Maneja el hardware necesario para encender/apagar los sensores.

ActiveMessageC:

Component necessari per poder encendre i parar la r.dio.

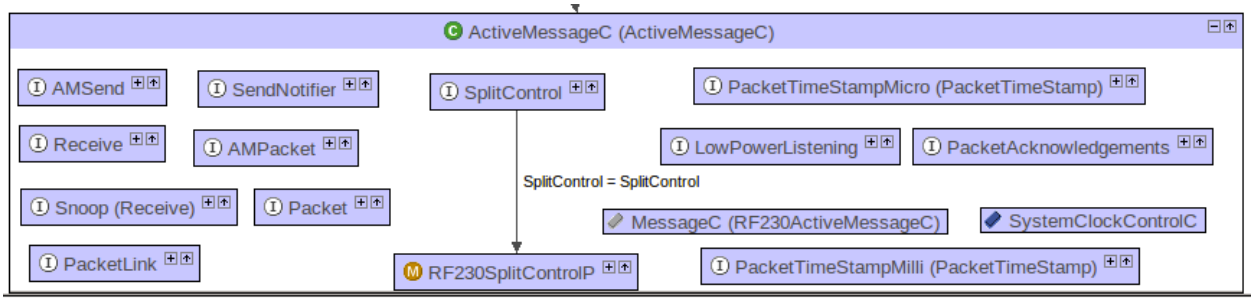


Figura 9 - ActiveMessageC

AMSenderC:

Responsable del envío de los paquetes de radio.

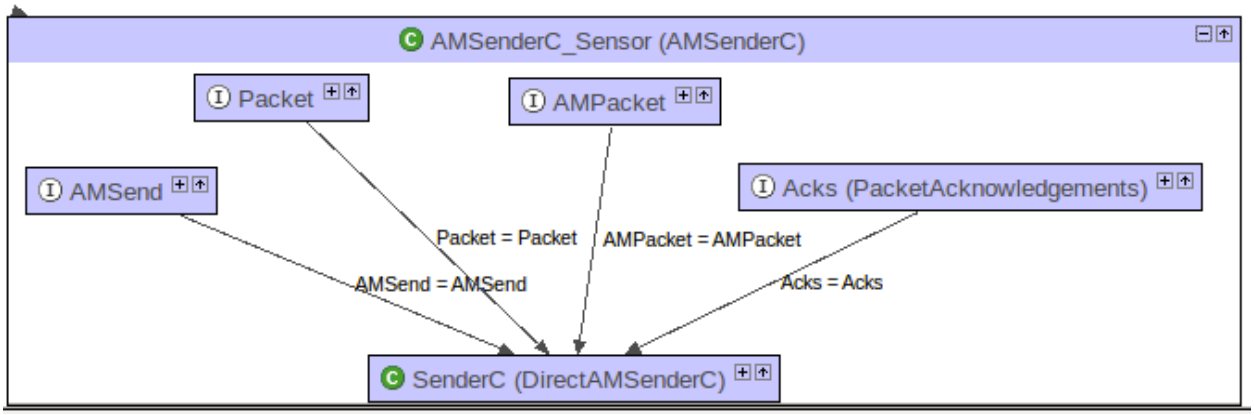


Figura 10 - AMSenderC

AMReceiverC:

Componente necesario para poder recibir mensajes por radio.

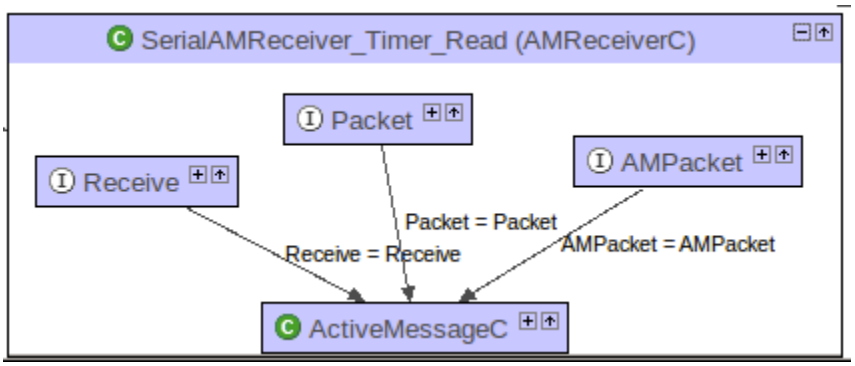


Figura 11 - AMReceiverC

SensorBateriaC:

De desarrollo propio, sirve para leer el sensor de la batería.

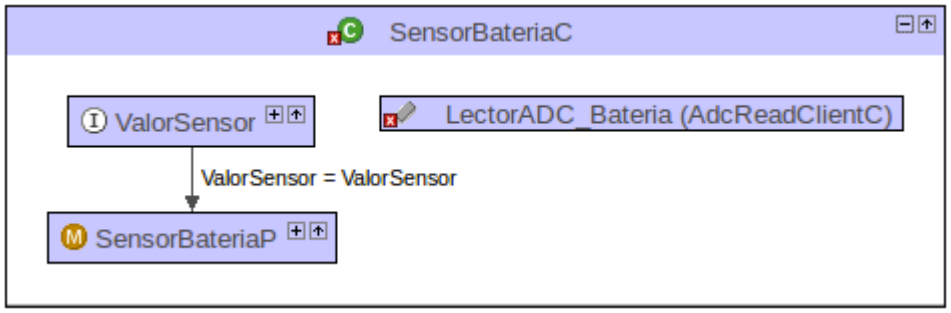


Figura 12 - SensorBateriaC

SensorTemperaturaC:

Al igual que el anterior pero nos sirve para leer la temperatura.

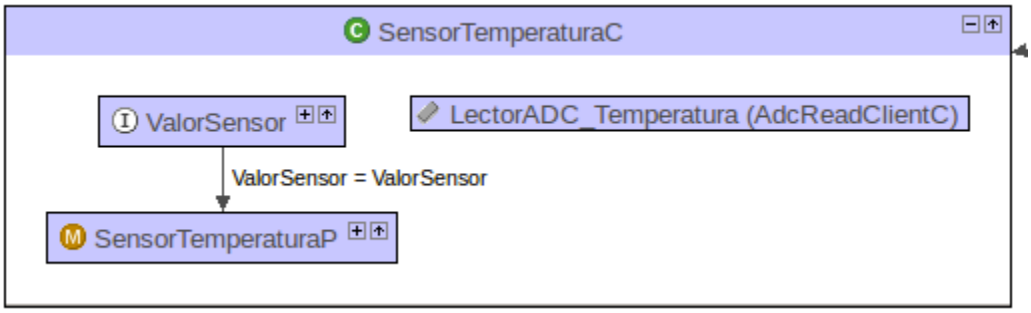


Figura 13 – SensorTemperaturaC

CONSTANTES - comunes.h:

Los tiempos de las constantes son milisegundos, y los voltajes milivoltios.

AM_SENSORMSG: nos sirve para distinguir los distintos Active Message (AM), sino no sabría con cual está.

AM_TIMERREADMSG : ídem para la lectura de los mensajes de AM.

AM_TIMERSENDMSG: ídem para el envío de los mensajes de AM.

TEMPO_LECTURA: tiempo que pasará entre cada bucle de lectura de los sensores.

TEMPO_ENVIO: tiempo que pasará entre cada envío de datos a la base.

VREF: es el voltaje de referencia de trabajo de la mota (2560 mV)

READY: sistema listo para comenzar lectura.

ERROR: hay algún error.

LEER_BAT: el sistema puede leer bacteria.

LEER_TEMP: el sistema puede leer temperatura.

ESTRUCTURAS DE DATOS: comunes.h

Nos permite envíar los datos de los sensors al receptor

```
typedef nx_struct SensorMsg {
        nx_uint8_t moteID;      //ID mota
        nx_uint16_t cBat;       //Batería
        nx_uint8_t vRef;        //Voltaje referencia (mV)
        nx_uint8_t cTemp;       //Temperatura
} SensorMsg;
```

Nos permite cambiar el periodo de lectura los datos de los sensores.

```
typedef nx_struct TimerReadMsg {
        nx_uint32_t timerVal;           //Estructura mensaje Temporizador Lectura
} TimerReadMsg;
```

Nos permite cambiar el periodo de envío los datos de los sensores a la base.

```
typedef nx_struct TimerSendMsg {
        nx_uint32_t timerVal;          //Estructura mensaje Temporizador Envío
} TimerSendMsg;
```

Interface – ValorSensor

Nos permite obtener valor del sensor.

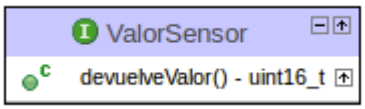


Figura 14 - ValorSensor.nc

3.3.- La aplicación JAVA en el PC

El sistema de gestión de la mota está llevado por una interface gráfica que he llamado GUI.java y gestionado por ControlMotas.java que explicaré más abajo.

Las librerías que nos proporciona TinyOS incluyen una llamada *MoteIF* que nos permite interrelacionarnos con las motas a través del socket que se crea con la aplicación que también incluye el sistema *SerialForwarder*.

Tal como se indica en la página 120 y siguientes de TinyOS Programming podríamos usar una mejora a esta librería como es *ReliableMoteIF* que mejora la anterior ya que *MoteIF* no retransmite el paquete en caso de fallo. No lo he hecho precisamente para eliminar robustez al sistema y probar hasta que punto fallan el envío de paquetes.

- *ControlMotas.java:* es la parte encargada de envíar/recibir los mensajes con los valores que proporciona la mota y la interfaz gráfica.
- *GUI.java:* es la interfaz gráfica con la que interactuamos con el sistema. Donde actualizamos los tiempos y la alarma de temperatura.

Para crear la interfaz gráfica, he tenido que instalar un componente en el entorno Eclipse, y que está desarrollado por Google, llamado Google Window Builder, ya que el Visual Editor con el que estaba acostumbrado a trabajar ha dejado de estar soportado, con lo que el diseño no es el más adecuado ya que no es tan flexible como lo era el anterior, ya que está orientado a la programación de dispositivos móviles.

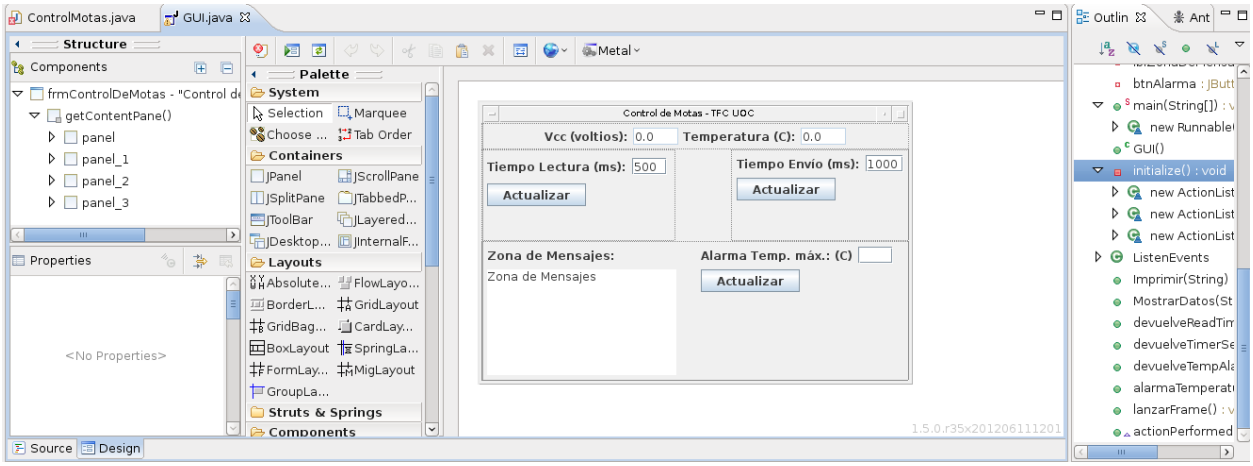


Figura 15 - Entorno de Window Builder

Al compilar la parte creada en NesC con MIG (Message Interface Generator), el sistema genera unas clases que son las que usamos en la programación de nuestro sistema y que nos facilita todo el trabajo de interacción con las motas.
En concreto yo hago uso de 3 clases que he empaquetado en un fichero JAR por comodidad.

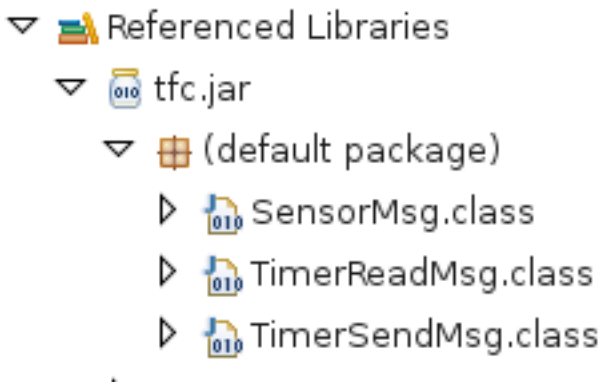


Figura 16 - Clases creadas con MIG

SensorMsg.class → Para monitorizar los valores de los sensores.
TimerReadMsg.class → Para saber el tiempo de lectura de los sensores.
TimerSensMsg.class → Para modificar los tiempos de lectura de los sensores.

3.4.- La mota base y el PC

En la mota que está conectada al PC instalo el programa BaseStation tras compilarlo tal y como explicaré más adelante en el apartado instalación. Y además en el PC estaremos corriendo la aplicación SerialForwarder que es el que nos permite mantener las comunicaciones entre MOTA y PC.

Podemos resumir el sistema en un diagrama de bloques como el que se muestra a continuación.

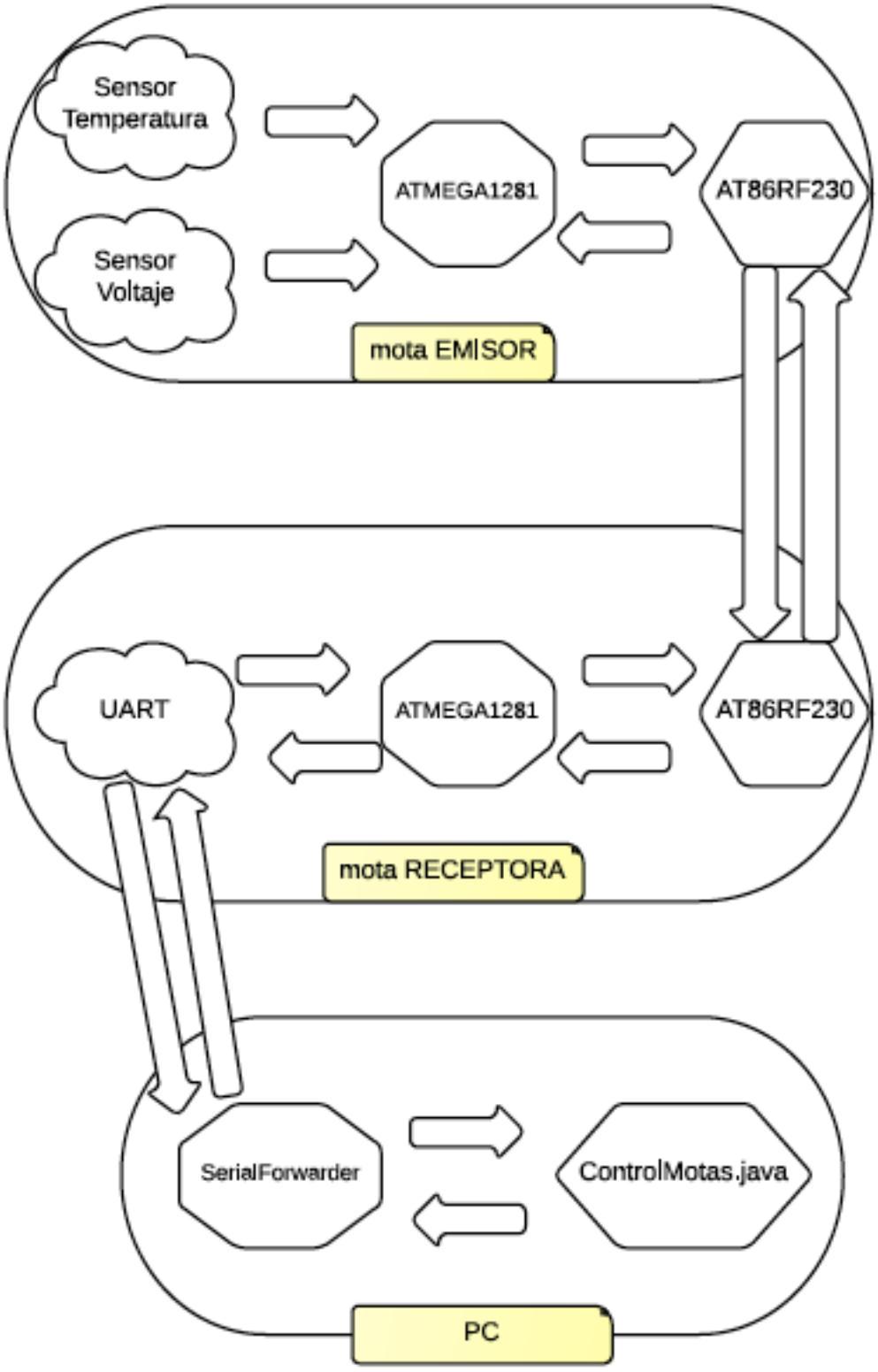


Figura 17 - Diagrama de bloques

3.5.- La web

Ya que los diversos problemas con el entorno de programación y el cambio de distribución en el sistema operativo no me ha permitido más, la parte web ha quedado reducida a una simple demostración de la lectura del fichero XML que se genera desde la aplicación JAVA.

Consiste en un entorno sencillo, y para simplificar las cosas he usado las librerías de "prototype" que permiten un uso mucho más cómodo y eficaz del javascript para el manejo del XML.

Prototype: es un conjunto de librerías, framework, creado en Javascript que podemos obtener desde su sitio web http://prototypejs.org/ . Nos permite implementar técnicas de AJAX de una manera cómoda y sencilla el desarrollo de nuestras webs.

El formato del fichero XML es el siguiente:

motas.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xml>
 <mota>
  <numero>1</numero>
  <temperatura>11</temperatura>
  <voltaje>10,25</voltaje>
 </mota>
 <mota>
  <numero>2</numero>
  <temperatura>22</temperatura>
  <voltaje>22</voltaje>
 </mota>
 <mota>
  <numero>3</numero>
  <temperatura>33</temperatura>
  <voltaje>31,3</voltaje>
 </mota>
</xml>
```

Figura 18 - Formato fichero XML

Figura 19 - Ejemplo de la web con datos aleatorios en el fichero XML

4.- Tecnología usada

Haré una descripción genérica, donde se entremezclarán hardware y software, ya que será en función de los equipos.

Portátil Apple Macbook Pro, donde se instalará la máquina virtual Linux con **VMware Fusion versión 4.1.1 (536016)**. La distribución de **Linux** que lleva es DEBIAN, y se va actualizando a medida que salen nuevas mejoras, en estos momentos es la que vemos en la Figura 20.

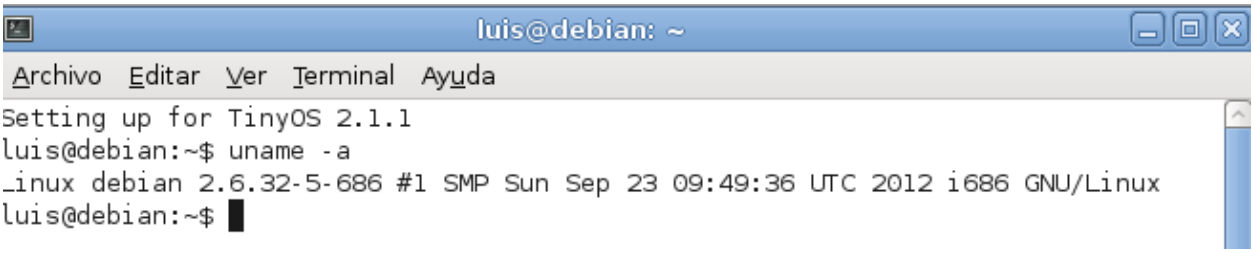


Figura 20 - Versión del sistema operativo

Y como podemos observar ya hemos instalado el **TinyOS 2.1.1**. y programaremos con el lenguaje **NesC**, ambos los detallaremos al final del apartado.

En este equipo será donde se realice el proceso de redacción y edición de la Memoria; la Presentación; y, todo el proceso de aprendizaje y pruebas preliminares del sistema. Una vez esté montado y funcionando trataremos de hacerlo funcionar en producción en un **servidor** que tengo funcionando y abierta a Internet, sobre el dominio http://www.probando.com, que también está sobre Ubuntu y que podemos observar en la siguiente captura de pantalla también.

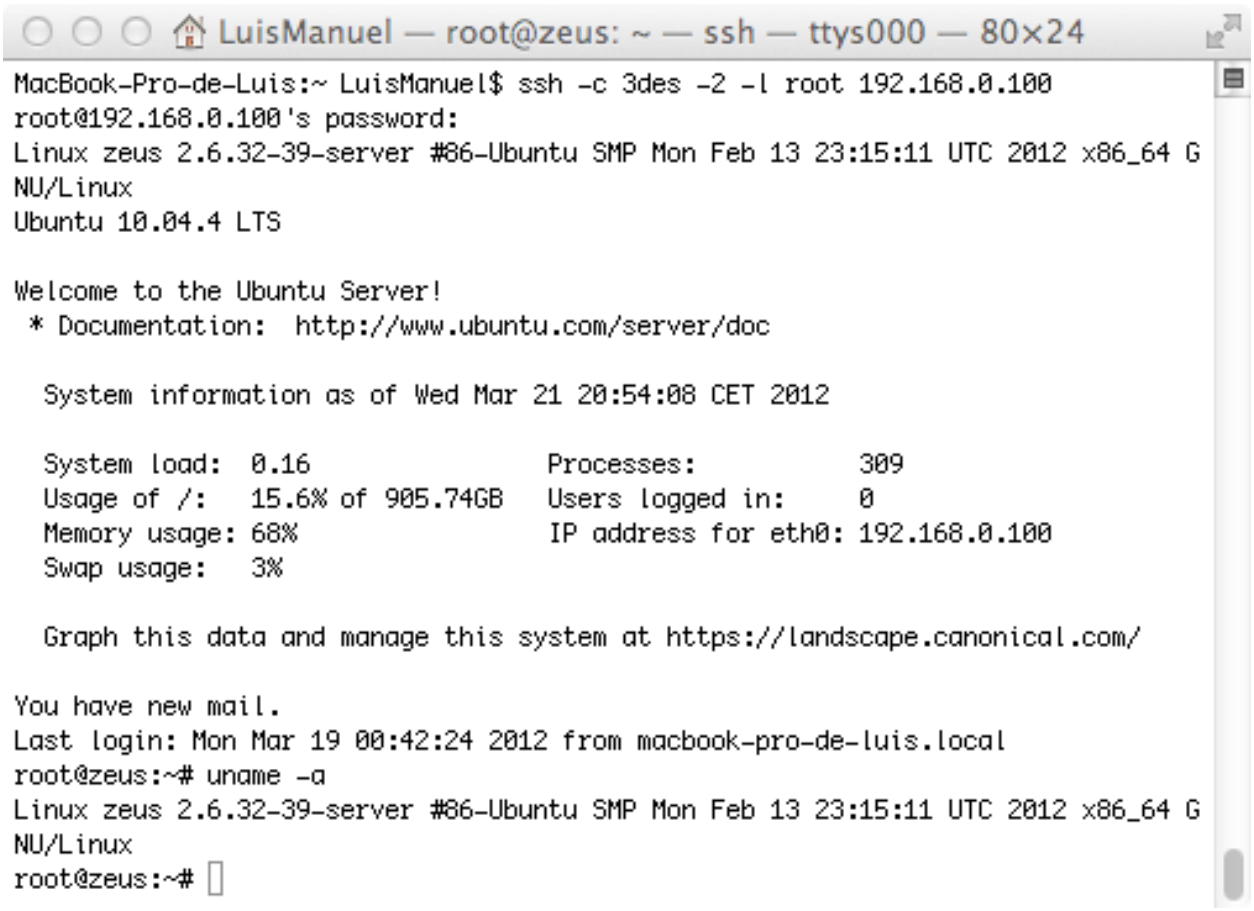


Figura 21 - Datos del servidor web

Este servidor está corriendo un **servidor web** Apache 2.2.14, que será el encargado de servir la web de gestión del sistema, tal como se observa en la Figura 21.

Luego como parte fundamental del sistema tenemos las motas que nos ha facilitado el departamento, en concreto son 2 unidades, COU_1_2, equipadas con sensor de temperatura, luz y efecto hall. Una será la que sirva para el muestreo de datos y la otra como pasarela de comunicación con el servidor. En la Figura 22 podemos ver sus componentes más importantes y que cito a continuación:

- Módulo radio ZigBit 2,4 Ghz (Atmega181+AT86RF230+antena),
- Puente USB-Serie
- Sensor de temperatura MCP 9701 (rango: -10 a 125 grados)
- Sensor de luz PDV P9003-1 (400 a 700 nm)
- Sensor Hall (detector campo magnético)
- Leds rojo, naranja y verde.
- Pulsadores de Reset y Usuario.

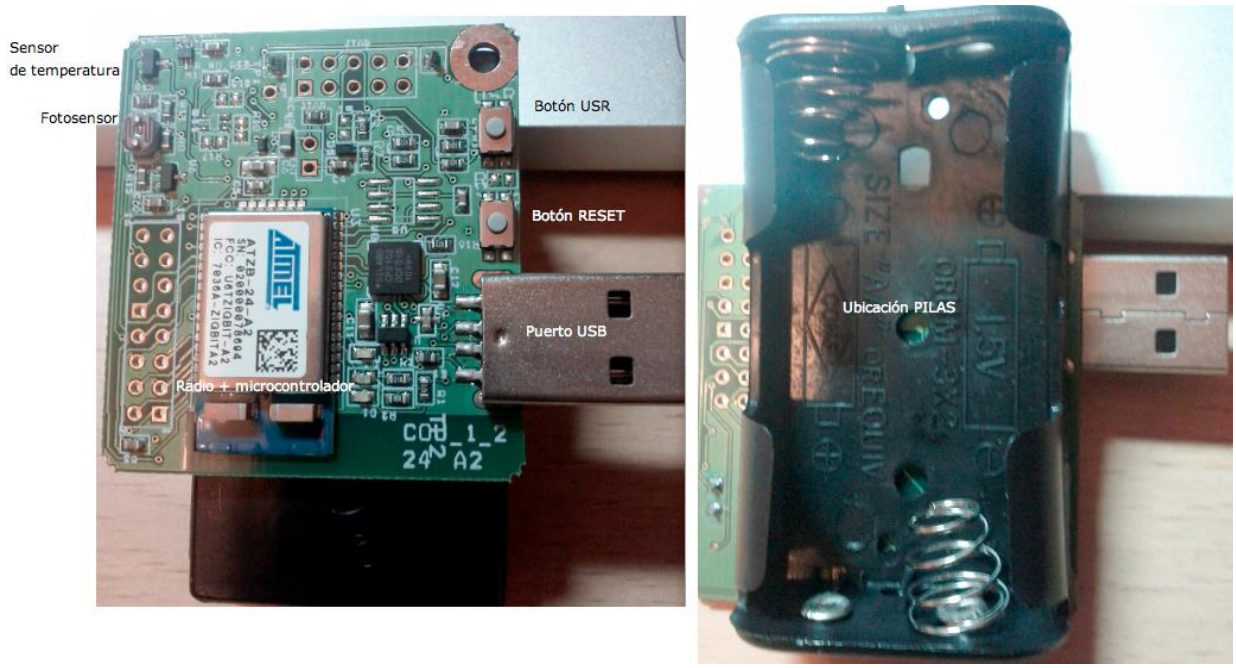


Figura 22 - Partes de la mota COU_1_2

Podemos obtener más información sobre el hardware de las motas en sus hojas técnicas, a las cuales hago referencia en los siguientes enlaces:

- Atm1281: http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf
- Sensor luz: http://www.advancedphotonix.com/ap_products/pdfs/PDV-P9003-1.pdf
- Sensor temperatura: http://ww1.microchip.com/downloads/en/DeviceDoc/21942e.pdf
- Efecto hall: http://www.rohm.com/products/databook/sensor/pdf/bu52001gul-e.pdf

TinyOS y NesC, podemos decir que son el alma del proyecto junto con las motas.

TinyOS, como su propio nombre indica, es un sistema operativo libre basado en componentes y creado para sistemas empotrados inalámbricos. Y como tal cumple con los requisitos de utilizar poca memoria y permitir un muy bajo consumo en las motas y se programa sobre el lenguaje NesC.

TinyOS

Como decía en el párrafo anterior, es un sistema operativo basado en componentes y que nos permite un desarrollo rápido y con el mínimo tamaño en el código, así como su reutilización. La

aplicación se compila con el sistema operativo y luego se carga en la mota. Es un sistema basado en eventos, comandos y tareas.
*Un evento consiste en que un componente de bajo nivel notifica a uno de un nivel superior que ha ocurrido algo. Los eventos tienen prioridad sobre las tareas y se ejecutan inmediatamente.
*Los comandos, en cambio, van de arriba a abajo.
* Las tareas son llamadas indirectas a procedimientos, y funcionan como una cola FIFO. Esto permite que, si la cola está vacía, el sistema entra en reposo y así se ahorra batería. El sistema despierta cuando ocurre un evento. No permiten parámetros, ni devuelven valores.

Debido a esta manera de funcionar podemos ver que el funcionamiento es asíncrono al responder a interrupciones y síncrono para tareas programadas. Por lo que hay que realizar secciones atómicas (tareas) para evitar problemas, y durante su funcionamiento deshabilitan las interrupciones, y al finalizar las vuelven a activar.
El sistema operativo incluye una biblioteca de componentes, protocolos de red, drivers de sensores, herramientas de adquisición de datos, que podemos usarlas tal como vienen, o modificarlas para adecuarlas al uso que le queramos dar.

NesC

Está basado en el lenguaje C y orientado a componentes. El propio lenguaje proporciona componentes primitivos, y luego podemos obtenerlos compuestos de librerías.
La estructuración es un poco diferente a lo que estamos acostumbrados y puede inducirnos a error.
Los componentes nos proporcionan interfaces pero también las requieren para acceder a ellos. Los comandos son funciones definidas por una interfaz que viene implementada, y los eventos son funciones de la interfaz que debemos implementar. O sea, para que el componente pueda utilizar los comandos de la interfaz, debe tener implementados dichos eventos
Como mínimo se necesitan 2 ficheros básicos. El primer fichero es el de la configuración y el segundo es el módulo. A su vez el fichero de configuración está dividido en 2 partes, configuración (configura el componente) e implementación (enumeración de los componentes que se necesitan). Luego ese establece el “wiring”, o conexión de las interfaces de unos componentes con las interfaces de otros.
Como no es la intención explicar aquí el lenguaje, mostraré en la Figura 23 sus partes

principales.

MiPrograma.nc	MiProgramaM.nc	paquete.h

includes paquete;
 	
configuration MiPrograma{}
 	
implementation {
  components
  Main
  MiProgramaM;
 	
Main.StdControl->
MiProgramaM.StdControl;
	includes paquete;
	Module MiProgramaM{
	  provides{
	  Interface StdControl;
	  }
	  uses {
	  interface SendMsg;
	  }
	  }
	implementation {
	  // C code
	  }
		enum {
		  PKT_REQ=1;
		  PKT_HELLO=2
		};
		
		typedef struct paq{
		  int address;
		  int pkt_type;
		  ….
		} paquete;

Figura 23 - Componentes principales de un programa en NES-C

En la primera columna, vemos el fichero de módulo y en la última el fichero de definición de constantes y estructuras.

La compilación se hace sobre el directorio de trabajo con la sentencia `make cou24`, y si queremos instalar agregando `install`, o sea, `make cou24 install`, y si queremos identificar la mota se indica con `make cou24 install, 1`.

Javascript

Este lenguaje interpretado fue creado por Brendan Fich de Netscape con el nombre de Mocha, y posteriormente rebautizado como LiveScript, y finalmente quedó con el nombre de Javascript, en el momento en que Netscape le dio soporte a su navegador a finales del año 1995. Microsoft en su momento creo su propio dialecto al que llamó JScript, para evitar problemas de marcas, y así poder usarlo en su navegador Internet Explorer.

Está orientado a objetos, se basa en prototipos, es imperativo y de tipado débil, así como dinámico. Principalmente se usa del lado del cliente para mejorar la interfaz del usuario y crear páginas web dinámicas. También existe una versión del lado del servidor. Aunque por su nombre podría parecer que está relacionado con Java, tanto su semántica como propósito son diferentes, aunque sí adopte algún nombre y convenciones del mismo, tienen diferencias que los hacían incompatibles.

Hoy en día todos las navegadores web son capaces de interpretarlo y para evitar incompatibilidades se adoptó como estándar de la European Computer Manufacturers Association (ECMA) Y desde 1997 es un estándar llamado ECMAScript y posteriormente

también adoptado por ISO. Actualmente es una marcar registrada por Oracle y de amplia difusión.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

El uso en este proyecto será para crear la página dinámica para la visualización e interacción con las motas, en colaboración con el diseño en HTML5 de las páginas web, para tratar de darle una mayor dinámica y sin tener que recurrir a aplicaciones en Java, tratando de sacar el máximo provecho a su dinámica de funcionamiento.

HTML 5

Es la quinta versión del lenguaje HTML (HyperText Markup Language, versión 5), que es el lenguaje básico con que funciona la World Wide Web,HTML. Aunque en estos momentos no está aprobado todavía esta versión, trataré de aproximarme en lo más posible a las normas de esta evolución, ya que se conocen muchas de sus nuevas características y los nuevos navegadores poco a poco los van implementando ya que sus avances y mejoras son de gran utilidad.

Existen dos variantes para la sintaxis:

- el «clásico» HTML (text/html), que es la que conocemos como HTML5 y
- XHTML conocida como sintaxis XHTML5 servida como XML (application/xhtml+xml).

El desarrollo de este lenguaje de marcas es regulado por el Consorcio W3C

Dentro de las multiples posibilidades que ofrece esta nueva version podemos destacar como novedades las siguientes:

- Incorpora etiquetas (canvas 2D y 3D, audio, video) con codecs para mostrar los contenidos multimedia. Actualmente hay una lucha entre imponer codecs libres (WebM + VP8) o privados (H.264/MPEG-4 AVC).
- Etiquetas para manejo de datos: Datagrid, Details, Menu y Command. Permiten generar tablas dinámicas que pueden filtrar, ordenar y ocultar contenido en cliente.
- Mejoras en los manejos de formularios. Nuevos tipos de datos (eMail, number, url, datetime …) y facilidades para validar el contenido sin el uso de Javascript.

- Visores: MathML (fórmulas matemáticas) y SVG (gráficos vectoriales). En general se deja abierto a poder interpretar otros lenguajes XML.
- Drag & Drop. Nueva funcionalidad para arrastrar objetos como imágenes.

5.- El entorno de trabajo

Nuestro ecosistema de trabajo dentro del ordenador se basa en 3 elementos principales: Eclipse , Meshprog y Yeti2

Eclipse

Es un entorno de desarrollo que nos permite trabajar de una manera mucho más cómoda que usando un simple editor de texto, ya que nos permite llevar un major control de todo el proyecto. Si deseas más información de dicho entorno así como de otros productos de dicha organización puedes visitar http://www.eclipse.org/org/

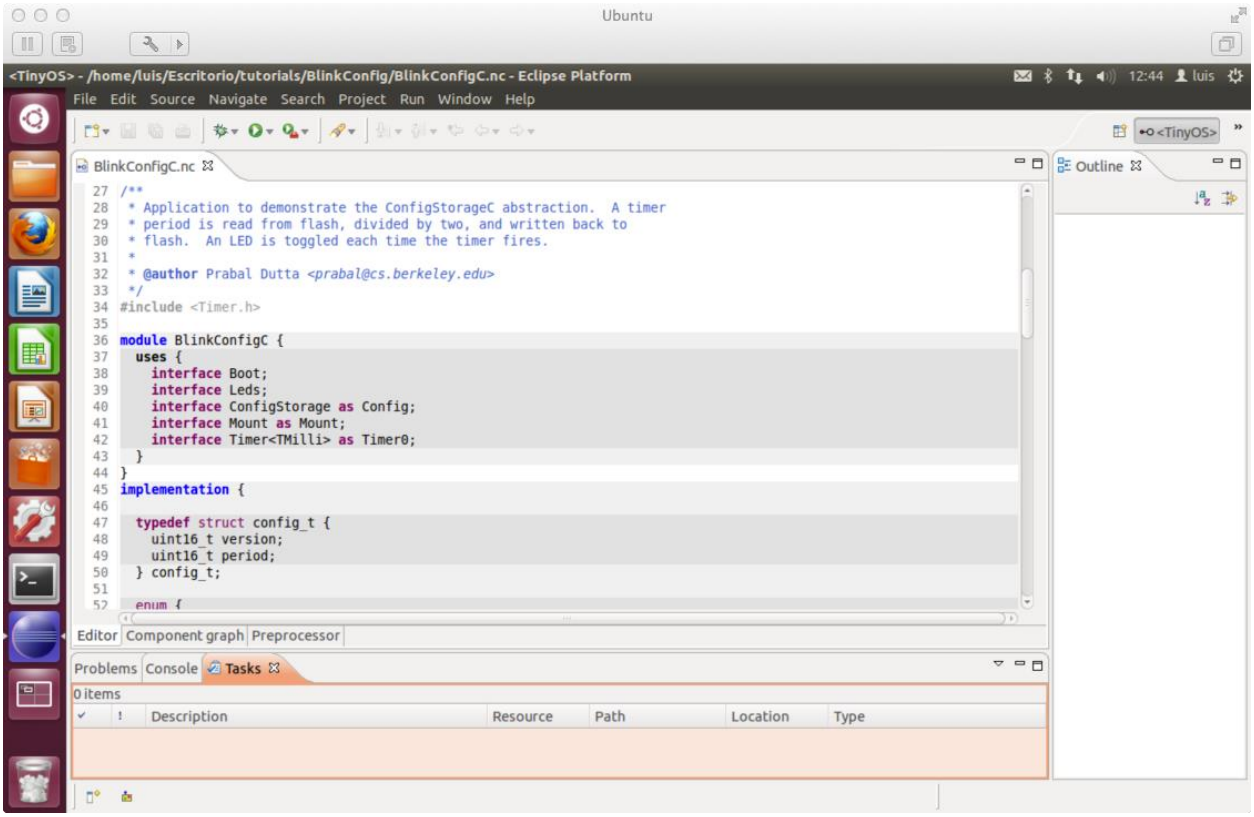


Figura 24 - Entorno Eclipe con Yeti2 parcialmente instalado

Podemos ver en la imagen de la Figura 24 un ejemplo del entorno de desarrollo Eclipse donde ya se han instalado casi todas las opciones para un mejor funcionamiento con NesC/TinyOS.

Meshprog

Es la utilidad que nos permite programar las motas, por ejemplo para programar una mota con un programa llamado main, haríamos lo siguiente

`make cou24` (esto compilaría para nuestro hardware)

`meshprog -t/dev/ttyUSB0 -f./build/cou24/main.srec`

(esto programaría nuestra mota que está conectada a nuestro puerto USB)

y después pulsar el botón de reset.

```
luis@ubuntu-twips:/opt/tinyos-2.x/apps/BlinkToRadio/src$ make cou24
mkdir -p build/cou24
    compiling BlinkToRadioAppC to a cou24 binary
ncc -o build/cou24/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all -t
arget=cou24 -fnesc-cfile=build/cou24/app.c -board= -DDEFINED_TOS_AM_GROUP=0x22 --
param max-inline-insns-single=100000 -DIDENT_APPNAME=\"BlinkToRadioApp\" -DIDENT_
USERNAME=\"luis\" -DIDENT_HOSTNAME=\"ubuntu-twips\" -DIDENT_USERHASH=0x544c9ccaL
-DIDENT_TIMESTAMP=0x50a2cc0cL -DIDENT_UIDHASH=0x8de6f159L -fnesc-dump=wiring -fne
sc-dump='interfaces(!abstract())' -fnesc-dump='referenced(interfacedefs, componen
ts)' -fnesc-dumpfile=build/cou24/wiring-check.xml BlinkToRadioAppC.nc -lm
    compiled BlinkToRadioAppC to build/cou24/main.exe
           11502 bytes in ROM
             469 bytes in RAM
avr-objcopy --output-target=srec build/cou24/main.exe build/cou24/main.srec
avr-objcopy --output-target=ihex build/cou24/main.exe build/cou24/main.ihex
    writing TOS image
luis@ubuntu-twips:/opt/tinyos-2.x/apps/BlinkToRadio/src$ meshprog -t/dev/ttyUSB0
-f ./build/cou24/main.srec
using tty /dev/ttyUSB0
reading from file ./build/cou24/main.srec
sending ./build/cou24/main.srec -> /dev/ttyUSB0

Opened file ./build/cou24/main.srec
Opened device /dev/ttyUSB0
.................,[i◆◆&]
Starting transmission.
finished!
```

Figura 25 - Ejemplo de compilación y programación con meshprog

Yeti2

Es un Plugin para el entorno IDE Eclipse que nos permite trabajar de una manera mucho más cómoda para el desarrollo de nuestro proyecto.

En estos momentos su desarrollo y mantenimiento ha sido abandonado por parte de sus creadores.

Podemos obtener más información en su web http://tos-ide.ethz.ch/wiki/index.php

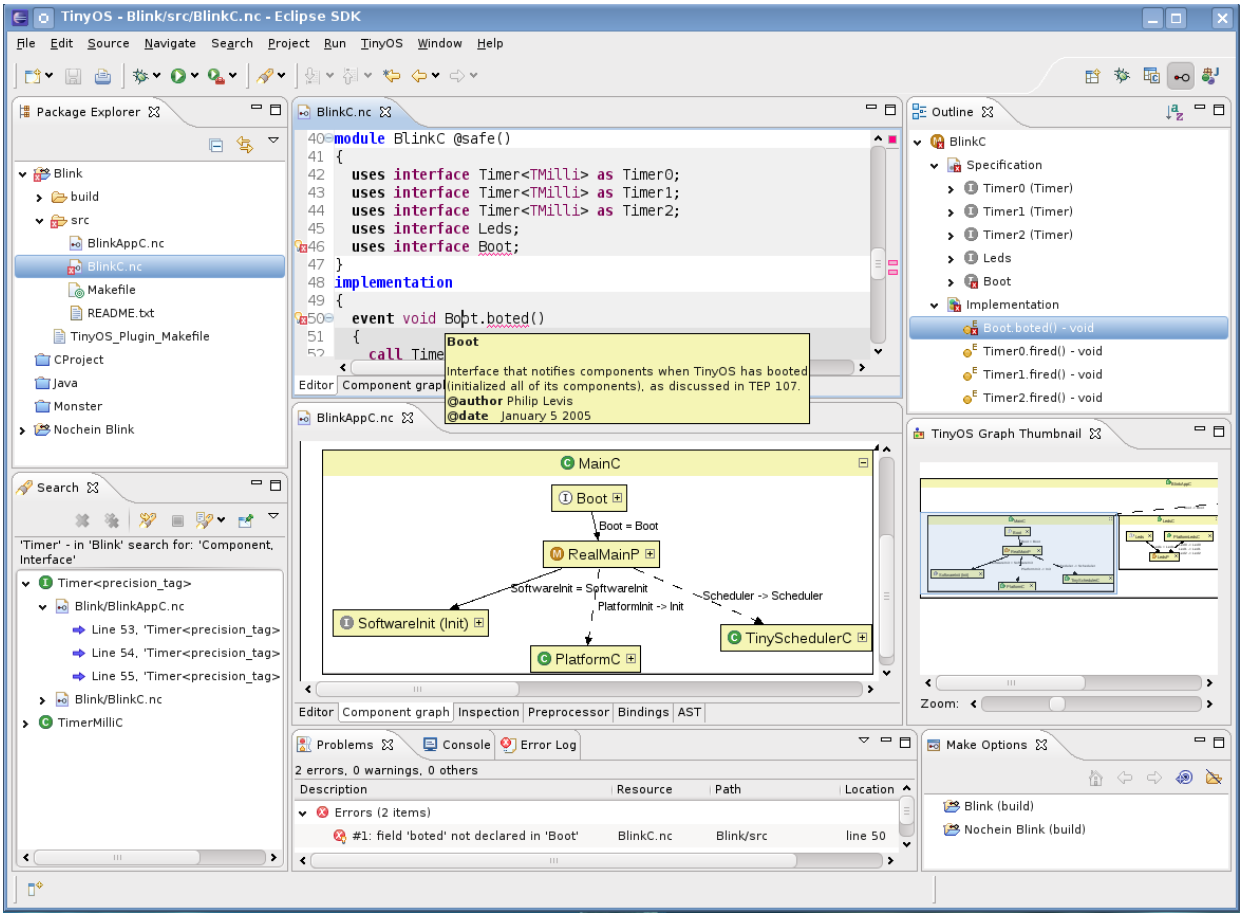


Figura 26 - Eclipse con Yeti2 completamente operativo

En la Figura 26 podemos ver todas las posibilidades que da el plugin al entorno de desarrollo, y el "Component graph" nos permite tener una vision rápida de los módulos, componentes e interfaces.

Podemos configurar el Eclipse para ejectuar "meshprog" directamente desde el entorno, simplemente hay que ir a . Run→External Tools → External Tool Configuration , pulsar el botón derecho del ratón sobre Programe, y luego → New y configurar según los parámetros que tenga cada uno en su sistema de archivos. Podemos verlo en la Figura 27

A veces la reprogramación de la mota falla, porque no se sincroniza bien, con lo que hay que reintentarlo, hasta que lo haga.

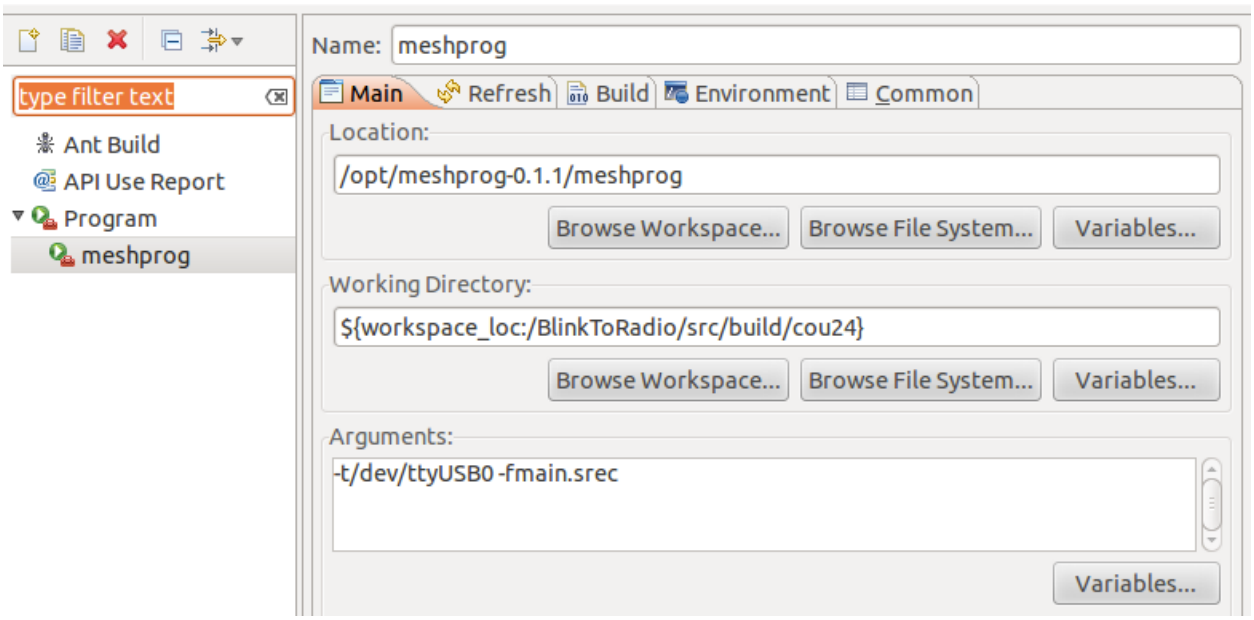


Figura 27 - Configurando Eclipse para Meshprog

Lucidchart

Es una aplicación basada en web que he instalado en el navegador Chrome de Google. Es muy útil para realizar pequeños diagramas y tenerlos siempre disponibles, así como para compartirlos en la nube a través de la cuente de Google Drive.

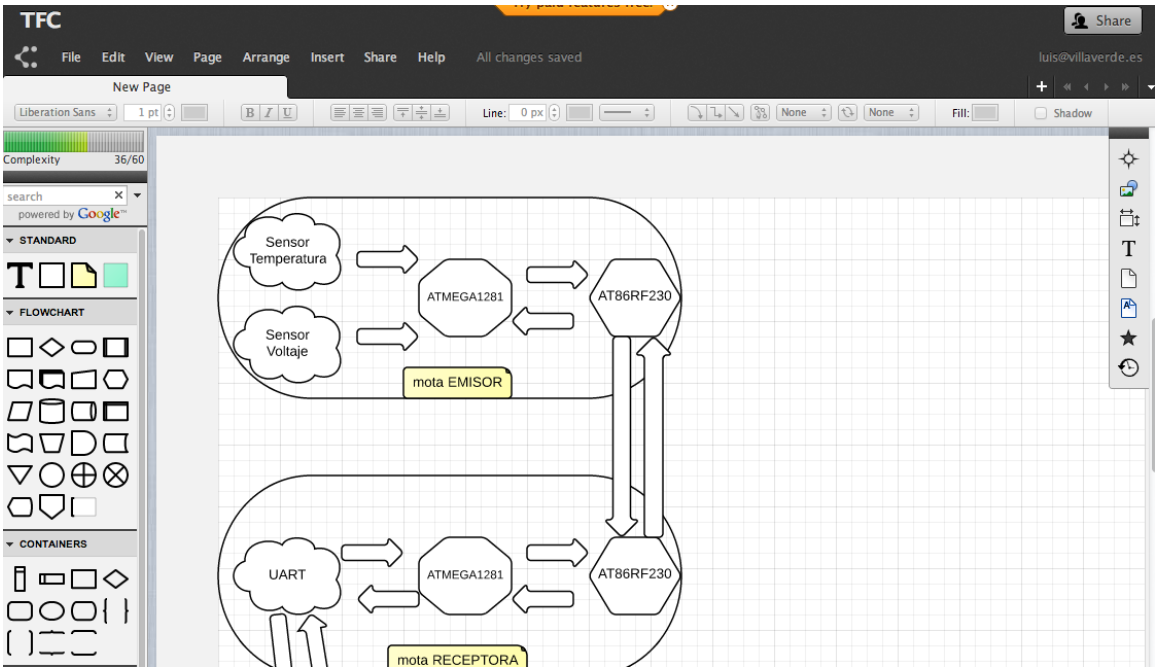


Figura 28 - Pantalla de LucidChart

ScreenFlow

Esta aplicación permite realizar la grabación de la pantalla del ordenador y es útil para la realización de tutoriales, así como presentaciones. Con ella he realizado la presentación del proyecto.

6.- Objetivos del proyecto

Podemos resumir los objetivos de la aplicación en:

1. Obtener los datos de los sensores de la mota: luminosidad y temperatura.
2. Gestionar esa información: si la luminosidad supera cierto umbral, se apagará/encenderá la luz; si la temperatura alcanza cierto umbral, se encenderá la climatización para enfríar o calentar.
3. Mostrar los resultados a través de los leds y de la aplicación web.

Como objetivos subyacentes, evidentemente están el implementar la comunicación entra la mota y el servidor y la interface de usuario.

Para todo ello abarcaremos el proceso de I+D+I completo: Investigación (estudiando y documentándome sobre el hardware y software a utilizar), Desarrollo (diseñando el sistema) e Implementación (poniéndolo a funcionar en una situación real). Aunque la última fase requeriría un diseño más comercial del producto, con unas motas “encapsuladas”, para poder ofrecer un aspecto más serio y profesional del producto.

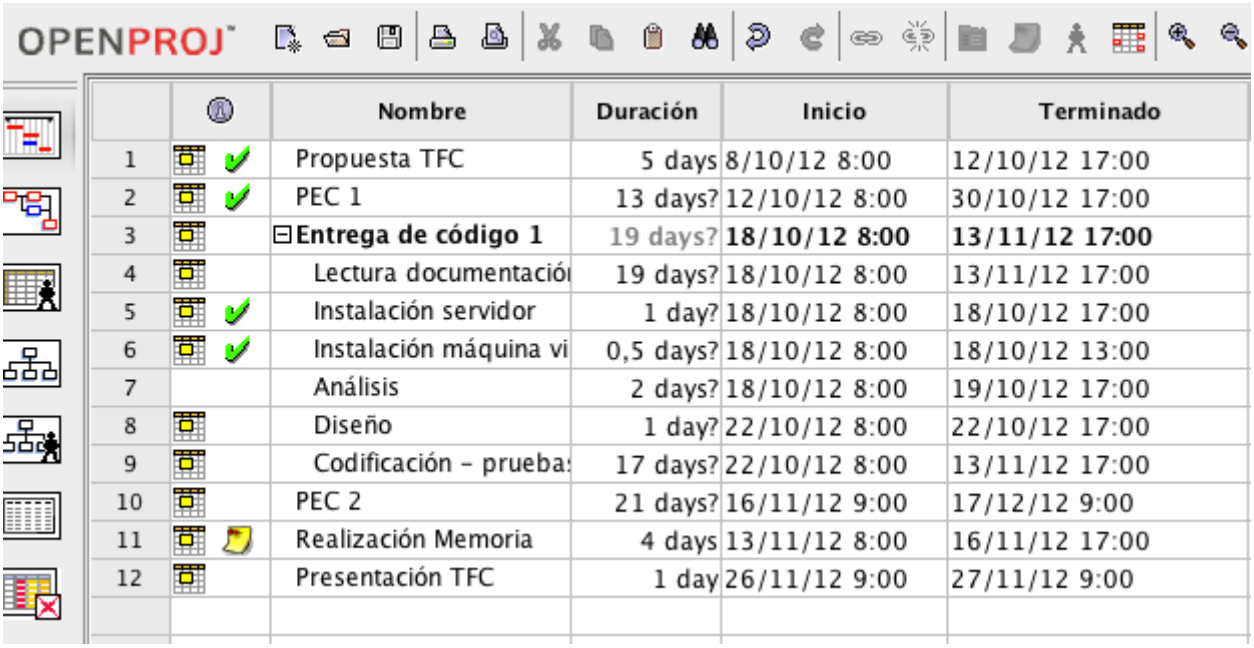
Y como objetivo fundamental del proyecto es saber integrar todos los conocimientos adquiridos durante los años de estudio de la carrera, desarrollando un sistema completo, para demostrar se han adquirido las habilidades necesarias.

7.- Planificación del proyecto

El tiempo que tenemos para el proyecto es hasta el 14 de junio, así que aproximadamente son casi 4 meses. En la Figura 29 podemos ver las tareas asignadas. Vemos que por ejemplo la lectura de documentación se hará durante todo el proceso de realización de la entrega del código 1. También podemos observar más claramente en la Figura 30, que la realización de la memoria será durante todo el ciclo de vida del proyecto.

La verdad, es que jamás en ningún momento de mi carrera nadie me ha enseñado a utilizar este tipo de diagramas, por lo que es una novedad para mi. Sí, se han mencionado pero en ningún momento ha sido objeto de estudio para mi. Por lo que no entiendo muy bien los conceptos profundos, aunque sí la generalidad de la idea.

En resumen el proyecto sufrirá una continua retroalimentación a medida que vaya avanzando y será completamente imposible seguir la planificación, ya que por motivos laborales solo sé de mi presente y no del futuro, por lo que planificar en el sentido estricto no tiene mucho sentido, sino más bien en el sentido de tareas POR HACER. Ya que en este caso la teoría es muy bonita, pero la realidad es otra. Y aunque yo indique de tal hora a tal hora tengo que hacer esto y en tal día, no puedo predecirlo. Así que no me queda más remedio que hacer en los huecos todo lo que se pueda. Los diagramas los dejaré realmente porque hay que cumplir con el "programa", no porque pueda cumplirlos.



	Nombre	Duración	Inicio	Terminado
1	Propuesta TFC	5 days	8/10/12 8:00	12/10/12 17:00
2	PEC 1	13 days?	12/10/12 8:00	30/10/12 17:00
3	**Entrega de código 1**	19 days?	**18/10/12 8:00**	**13/11/12 17:00**
4	Lectura documentació	19 days?	18/10/12 8:00	13/11/12 17:00
5	Instalación servidor	1 day?	18/10/12 8:00	18/10/12 17:00
6	Instalación máquina vi	0,5 days?	18/10/12 8:00	18/10/12 13:00
7	Análisis	2 days?	18/10/12 8:00	19/10/12 17:00
8	Diseño	1 day?	22/10/12 8:00	22/10/12 17:00
9	Codificación - prueba:	17 days?	22/10/12 8:00	13/11/12 17:00
10	PEC 2	21 days?	16/11/12 9:00	17/12/12 9:00
11	Realización Memoria	4 days	13/11/12 8:00	16/11/12 17:00
12	Presentación TFC	1 day	26/11/12 9:00	27/11/12 9:00

Figura 29 - Imagen de la programacion con OpenProj

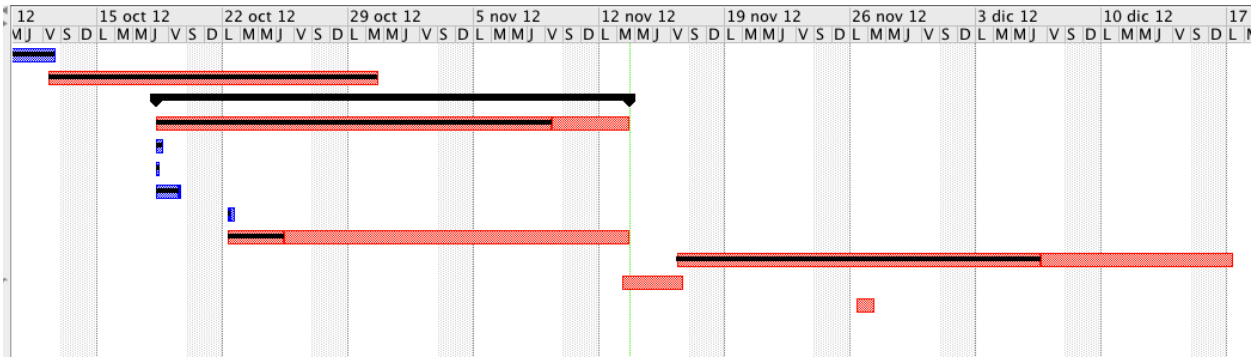


Figura 30 - Otra vista de la misma programación

Durante todo el proceso de análisis, diseño e implementación habrá una continua retroalimentación, ya que al ser algo tan diferente a lo que estoy acostumbrado, pues aún no tengo claro del todo el concepto profundo de funcionamiento. Y es lo que estoy leyendo para poder hacer una planificación con sus casos de uso más concreta y poder establecer unos tests adecuados.

8.- Resultado y valoración personal

Tengo que reconocer que me ha costado muchísimo más esfuerzo del que yo tenía pensado, sobre todo por la multitud de problemas que me han aparecido a lo largo del desarrollo del mismo.

En este caso el proyecto lo realizo yo solo, por lo que las dificultades son superiores a cuando realizas un proyecto real, donde normalmente colaboras con más personas y la ayuda y retroalimentación es muchísimo mayor.

Un pequeño resumen de problemas :

- La máquina virtual donde trabajaba se dañó y no tenía copia de la misma solo de mis archivos. Lograr conseguir posteriormente las carpetas correspondientes a las motas COU24 llevó bastantes semanas, a pesar de haber contactado directamente con la casa de las mismas, pero no obtuve respuesta alguna.
- Tras instalar nuevamente la máquina virtual e instalar el sistema entero, el sistema no era capaz de compilar, ni reconocía las trayectorias de los archivos del sistema. A pesar de contactar con Erick Decker y Matt Kelly y recibir su valiosa ayuda, no fue posible solucionar los problemas.
- Tras tener que cambiar la distribución Linux nuevamente, y tener que instalar Debian en vez de Ubuntu, todo funciona a la perfección (casi).
- Nuevamente aparecen problemas a la hora de programar las motas y con el SerialForwarder, ya que no es capaz de sincronizar los puertos, curiosamente tras alguna actualización de la distro se ha solventado el problema (he parado las actualizaciones para evitar más líos).
- Otro problema adicional, mientras en la otra distro con los permisos “666” sobre el puerto USB era suficiente para manejar las motas, con esta distro es necesario usar “777” sino no funcionan. No sé si es problema de la distro o de la máquina virtual.
- En el momento de realizar la aplicación JAVA, en concreto el entorno gráfico, acostumbrado a usar Visual Editor, y tras llevar varios años sin tocar la materia, dicho sistema ha desaparecido y he tenido que usar el de Google, llamado Google Window Builder. Tras múltiples problemas en la compilación, y gracias a un compañero que trabaja en Google, descubrimos que dicho plugin para Eclipse por defecto considera que el entorno es para tablets y activa propiedades que no son válidas para PC, por lo que generaba errores a la hora de compilar, ya que no consideraba válido el entorno de

trabajo. Este error también me ha tenido parado varias semanas, entre pruebas e investigación

Como si fuera un regalo de reyes, por fin descubrimos este último error y ya cuando había tirado la toalla del proyecto, me he puesto con él de nuevo a todo trapo. Por la falta de tiempo, y los continuos problemas, e incluso los amagos de abandono ante tanto problema, no he podido realizar todo lo que me hubiera gustado y tenía proyectado, necesitaría unas semanas más.

Conclusiones

La enseñanza que puedo sacar a todo esto, más bien las enseñanzas a nivel de forma de trabajo son:

- Por mucho que planifiques, siempre surgen errores
- Es mejor trabajar en equipo, compartir tareas, problemas, errores y éxitos.
- Como decía un viejo profesor que tuve hace muchos años de programación, cuando planifiques y calcules el tiempo y plazos… al final multiplícalo por 3, y aún así fallarás.

Sobre el TinyOS, también he sacado como conclusiones:

- Considero que es un sistema que está verde. Por lo que he visto a lo largo de estos meses de suscripción a la lista de correo del sistema, no soy yo el único que ha tenido graves problemas a la hora de lograr montar un sistema estable.
- Montar un sistema estable es complicado y una vez conseguido, lo mejor es desactivar las actualizaciones ya que si estaba funcionando y se actualiza pueden ocurrir errores impredecibles como que las librerías no sirvan de una versión de compilador a otro, y el sistema no ha sabido actualizar todo correctamente y tienes que hacer “fontanería” para poder solucionar los problemas.
- La comunidad de usuarios del sistema es principalmente de un par de profesores/desarrolladores que participan en el proyecto y el resto son universitarios que les han puesto este sistema para realizar sus prácticas en la facultad.

Quizás sería más fácil y tendría más posibilidades de desarrollo usar sistemas basados en ARDUINO (http://www.arduino.cc/es/) que tiene mucho más hardware y con más posibilidades. No solo motas. Tiene una base de usuarios mucho mayor por lo que la retroalimentación es muchísimo mayor y más rápida.

Aunque TinyOS se presenta como el primer sistema operativo diseñado desde cero para WSN, creo que deberían reforzar más la parte de crear un entorno de trabajo ESTABLE, porque se

pierde muchísimo tiempo en lograr empezar a trabajar, y viendo las listas de correo del sistema, es el pan de cada día, la guerra de todos sus usuarios novatos, que pierden mucho más tiempo en empezar a trabajar, que en conocer el sistema. Hay a gente que siguiendo los pasos de instalación de sitios como el de Matt Kelly http://www.keally.org/tinyos.php , el propio wiki que facilita la UOC en su web http://cv.uoc.es/app/mediawiki14/wiki/Inici24, o el de la Universidad Técnica Federico Santamaría en Chile, http://profesores.elo.utfsm.cl/~agv/elo323/2s10/Assignment/TinyOSLinux/Instalar%20TinyOS%20sobre%20Linux.html les ha funcionado a la primera (de hecho a mi el semestre pasado me pasó) hay un número muy elevado de usuarios que se decepcionan al no lograr montar un entorno de trabajo sólido y estable.

Tenemos otros sistemas operativos que pueden ser una buena alternativa como:

- LiteOS - http://www.liteos.net/
- Contiki - http://www.contiki-os.org/
- ecOS - http://ecos.sourceware.org/
- MicroC/OS-II - http://micrium.com/page/products/rtos/os-ii

Creo que el objetivo básico de aglutinar todos los conocimientos recibidos durante estos años de estudio queda cumplido, aunque no haya cumplido con todos los objetivos iniciales que me había propuesto en el proyecto. Pero sirve como base para desarrollar un sistema mucho más completo que permita obtener más parámetros de las motas, así como su representación.
Se pueden realizar multitud de opciones, desde históricos de datos, gráficos, conexiones con relés a dispositivos, envíos de mensajes vía internet, alarmas, etc

Conclusión final

He integrado conocimientos de hardware, de gestión de tiempos, de programación en varios lenguajes, de administración de un servidor Linux, de administración de un servidor web apache, y he descubierto que todavía me falta mucho por aprender, que tengo las bases, pero ahora tocaría especializarse y centrarse en algo específico.

9.- BIBLIOGRAFÍA

Webs:

http://docs.tinyos.net/tinywiki/index.php/TinyOS_Tutorials

http://www.tinyos.net/nest/doc/tutorial/

http://docs.tinyos.net/tinywiki/index.php/Main_Page

http://crysol.org/node/405

http://cv.uoc.es/app/mediawiki14/wiki/Pàgina_principal

http://sourceforge.net/projects/tinyos/

http://vimeo.com/channels/tinyos

http://tinyos.ethz.ch/phpBB3/

http://www.keally.org/tinyos.php

http://script.aculo.us/

http://api.prototypejs.org/

Libros:

- *TinyOS Programming*, Philip Levis –David Gay, Cambridge University Press, ISBN: 9780521896061.
- HTML5, CSS3 y JavaScript, Julie C. Meloni, Anaya-Sams, serie programación, ISBN: 9788441531932.

10.- Compilación y ejecución

Se facilita un fichero comprimido llamado “workspace.zip” que al descomprimirlo genera la siguiente estructura de carpetas

Figura 31 - Estructura de carpetas

En la primera carpeta está la aplicación java, y en la segunda está la parte correspondiente a la programación de las motas.
Vamos a realizar la compilación de la parte correspondiente, suponemos que tenemos los conocimientos básicos para manejarnos dentro de la estructura de directorios y sabemos cambiar de directorio

Preparando las motas

A una la llamaremos BASE y estará conectada al puerto del PC, en este caso vía USB, y la otra será autónoma y habrá que ponerle pilas para que funcione, y que llamaremos EMISOR.
Primero tenemos que dar permisos al puerto para que funcione correctamente

```
root@debian:/home/luis# chmod 777 /dev/ttyUSB0
root@debian:/home/luis#
```

Figura 32 - Dando permisos al puerto

Ahora accedemos a la carpeta donde está el código fuente y procedemos a su compilación

```
luis@debian:~$ cd /home/luis/workspace/TFC/src/
luis@debian:~/workspace/TFC/src$ make cou24
mkdir -p build/cou24
    compiling EmisorC to a cou24 binary
ncc -o build/cou24/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all
 -target=cou24 -fnesc-cfile=build/cou24/app.c -board= -DDEFINED_TOS_AM_GROUP=0
x22 --param max-inline-insns-single=100000 -DIDENT_APPNAME=\"EmisorC\" -DIDENT
_USERNAME=\"luis\" -DIDENT_HOSTNAME=\"debian\" -DIDENT_USERHASH=0x03f7dcf9L -D
IDENT_TIMESTAMP=0x50e8791fL -DIDENT_UIDHASH=0x059fc602L -fnesc-dump=wiring -fn
esc-dump='interfaces(!abstract())' -fnesc-dump='referenced(interfacedefs, comp
onents)' -fnesc-dumpfile=build/cou24/wiring-check.xml EmisorC.nc -lm
    compiled EmisorC to build/cou24/main.exe
          13048 bytes in ROM
            507 bytes in RAM
avr-objcopy --output-target=srec build/cou24/main.exe build/cou24/main.srec
avr-objcopy --output-target=ihex build/cou24/main.exe build/cou24/main.ihex
    writing TOS image
luis@debian:~/workspace/TFC/src$
```

Figura 33 - Compilando código de las motas

Ahora procederemos a la programación de las motas, recordemos que una vez lanzado el comando de programar hay que pulsar el botón de RESET. Considero que ya tenemos programada la BASE con el código que incluye el tinyOS de BaseStation. Solo voy a programar la mota EMISOR.

```
luis@debian:~/workspace/TFC/src$ cd build/cou24/
luis@debian:~/workspace/TFC/src/build/cou24$ meshprog -t /dev/ttyUSB0 -f main.srec
using tty /dev/ttyUSB0
reading from file main.srec
sending main.srec -> /dev/ttyUSB0

Opened file main.srec
Opened device /dev/ttyUSB0
.............,[i@@&]
Starting transmission.
finished!
luis@debian:~/workspace/TFC/src/build/cou24$
```

Figura 34 - Programación de la mota

En esta parte podría producirse un error a veces y no sincronizar correctamente, volveríamos a intentarlo, suele pasar en las máquinas virtuales.

Ahora conectamos la mota BASE al ordenador al puerto USB y ejecutamos la aplicación SerialForwarder que nos facilita el sistema TinyOS y que nos permitirá capturar con ella los paquetes que recibe la mota BASE.

```
luis@debian:~$ java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB0:19200 &
[1] 9194
```

Figura 35 - Inicio del SerialForwarder

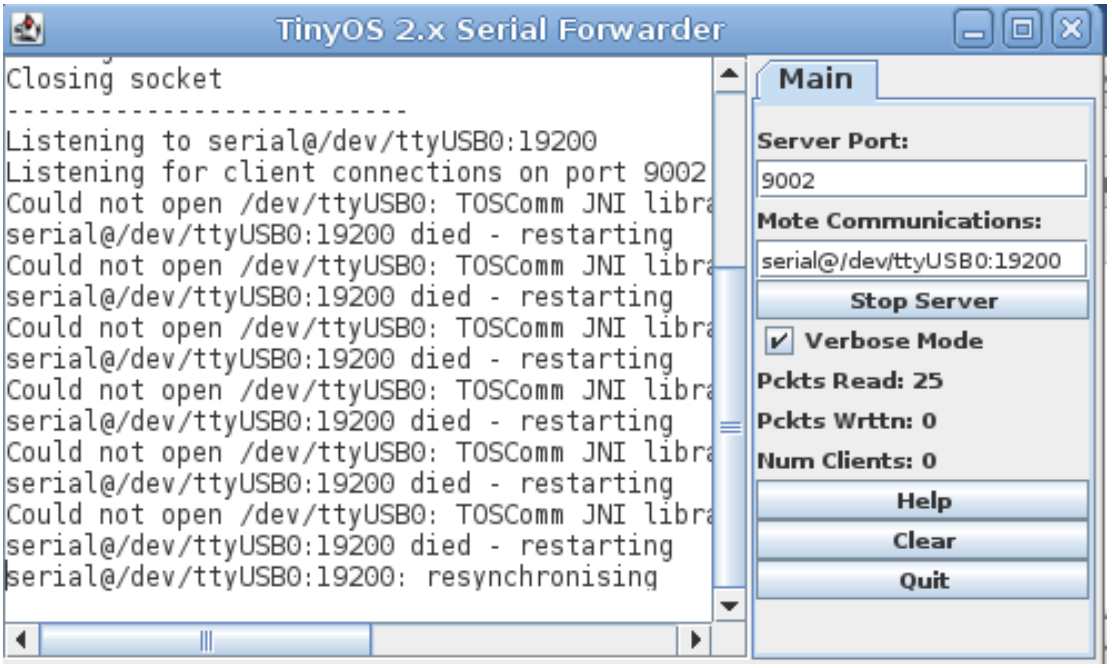


Figura 36 - Serial Forwarder funcionando

Tal y como podemos ver en la figura 36, no siempre sincroniza el puerto a la primera, por lo que en ese caso lo mejor es parar el servidor, esperar un ratillo y volverlo a arrancar, en el botón “Stop Server” o salir con el “Quit” y volver a lanzarlo.

La aplicación java

Ya tenemos casi todo listo. Ahora considero que no tengo compilada la aplicación en java, así que abrimos nuestro entorno de programación Eclipse por comodidad (podría usarse el compilador en línea) y abrimos el proyecto, seleccionamos “ControlMotas”, pinchamos el botón derecho RUN AS Java Application, luego (default package) tal como en la figura 37

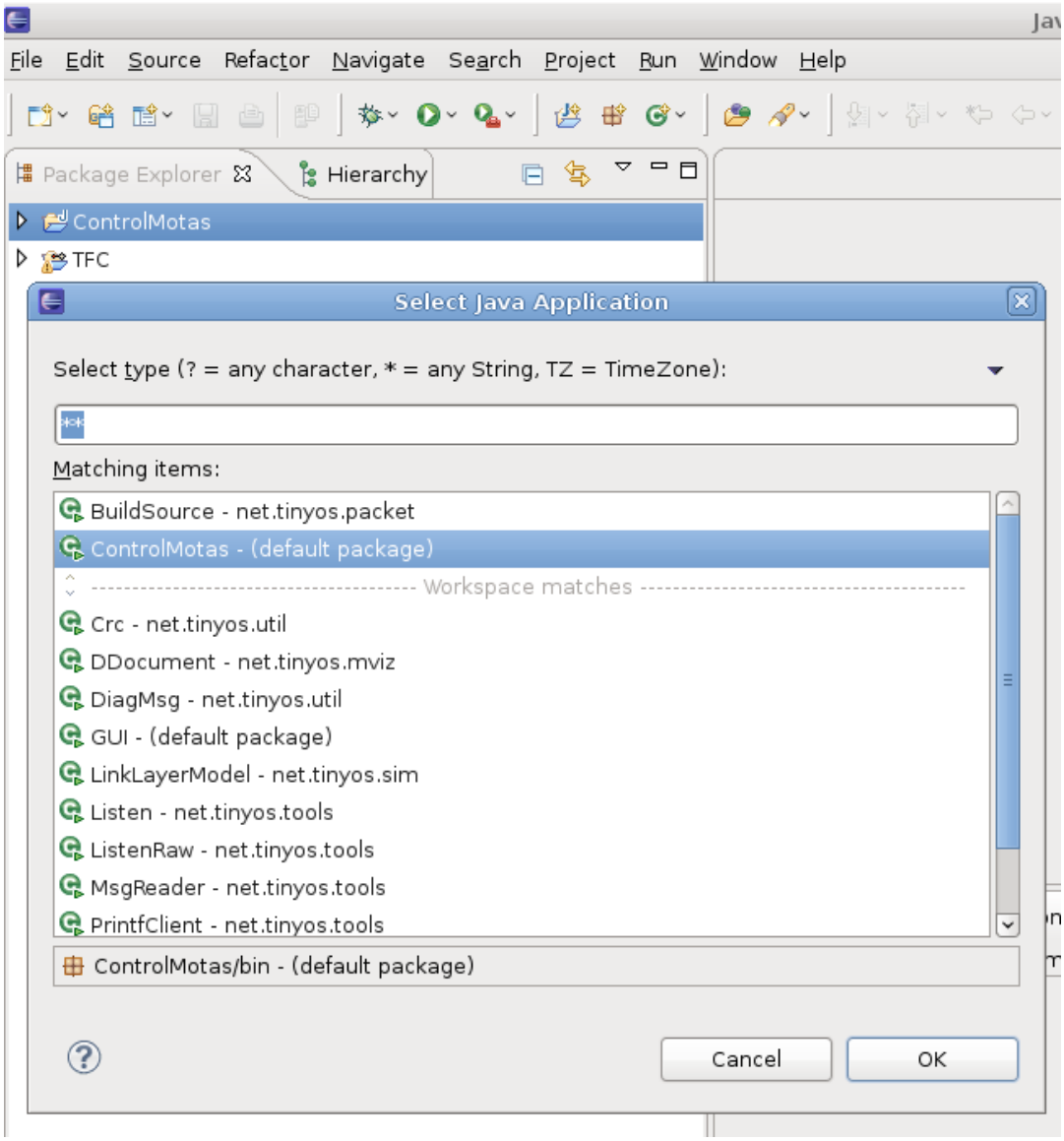


Figura 37 - Ejecución y compilación de la aplicación JAVA

Y ya tenemos nuestro programa funcionando, por defecto se le han pasado unos valores, que son los que vemos en la pantalla. Solo basta cambiarlos para ver como funciona. Por ejemplo ahora mismo marca 24 grados, y la alarma de temperatura 20, por lo que saldrá el mensaje

correspondiente en la zona de mensajes y se pondrá en rojo el fondo del texto de la temperatura.

Figura 38 - Aplicación funcionando con alarma de temperatura activa

Si modifico y pongo 25 como temperatura de alarma instantáneamente desaparece la alarma.

Figura 39 - Aplicación sin alarma de temperatura

Como truco para ver la variación de temperaturas, basta con pegar el dedo al sensor de temperatura y veremos como sube hasta nuestra temperatura corporal, como si fuera un termómetro, podemos ver su situación en la figura 22.

Opción web-xml

Como no me ha sido posible preparar un pequeño widget, he añadido una pequeña carpeta al proyecto que se llama web, donde van incluidos unos archivos que permiten ver los resultados también a través de la web, con la dirección http://www.probando.com:81

Lo hago a través del puerto 81 ya que no lo he podido montar a través del servidor de producción, pero sí he redireccionado dicho puerto en el router hacia el servidor web que he montado en la máquina virtual.

La última versión del proyecto incluye una opción que permite generar un fichero XML. Como está diseñado con una librería de javascript, sería necesario instalar en el servidor web, en la carpeta correspondiente al dominio, la librería y el resto de ficheros tal y como se adjuntan.

Como podría haber problemas de seguridad por los permisos de los ficheros, recomiendo crear un enlace simbólico del fichero xml en la carpeta del servidor web que esté redireccionado al fichero en sí, en la carpeta de trabajo que será la del usuario.

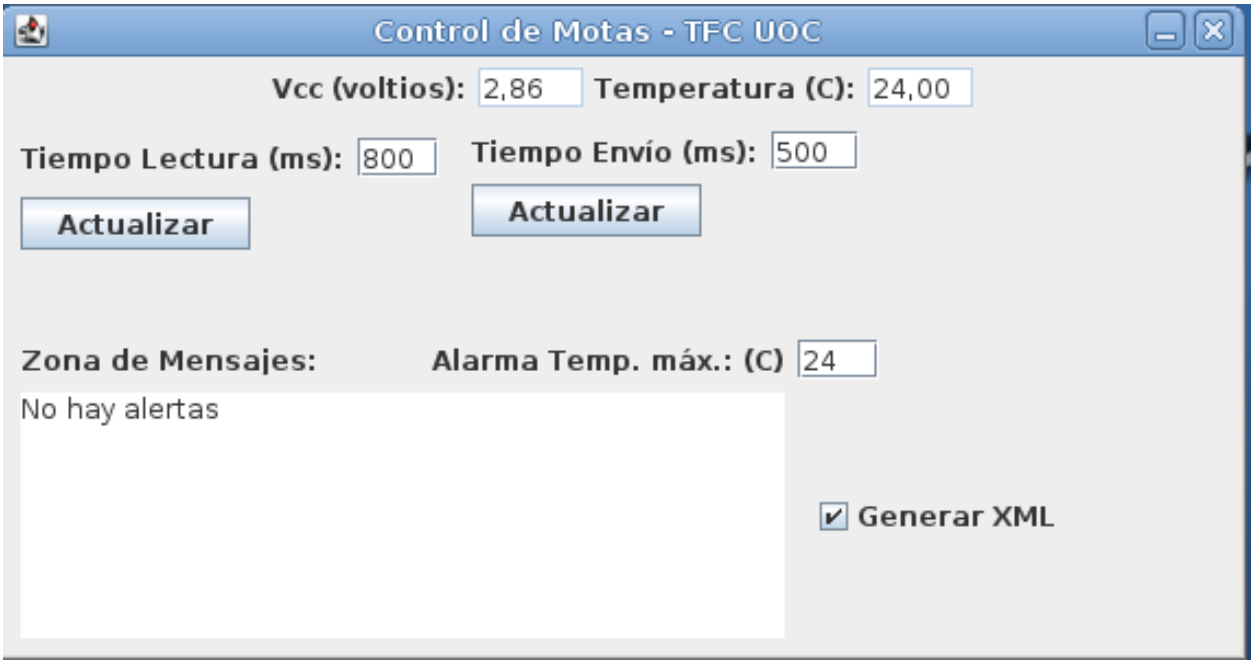


Figura 40 - Aplicación final con XML

Agradecimientos

En esta sección no cabrían todas las personas a las que quisiera agradecer su ayuda tanto técnica, como personal y humana. Por lo que si no ves tu nombre aquí no es que me haya olvidado de ti.

- Primero a l@s que no están y me han dado tan buenos momentos.
- A mis padres, hermanas, cuñado y sobrinos por soportarme con mis locuras y manías y fueron quienes pusieron mi primer ordenador (un Commodore 64) en mis manos.
- A Jennifer por hacerme sonreír y despertarme cuando me quedo dormido.
- A Iaio y a Diego por esos largos paseos para eliminar tensiones.
- A Fernando Lozano por aclararme conceptos técnicos que ya tenía olvidados.
- A Erick Decker, András Biró y Sean Dekker por contestar mis dudas de TinyOS.
- A mis fisios Fer y María por desbloquear mis hernias después de tantas horas sentado.
- Y como no, a mi tutor Sebastià por darme el empujón final y animarme cuando ya estaba rendido.

Gracias de verdad a todos, y aunque mi vida profesional ahora se centrará en la salud, han sido 30 años, apasionados por la informática. Pienso dedicarle más pasión a mis pacientes, si es que es posible. Y la informática pasa a ser a partir de ahora una herramienta para mi.

Con esto queda concluida una fase de mi vida y comienza otra más humana.

ÍNDICE DE FIGURAS

ÍNDICE