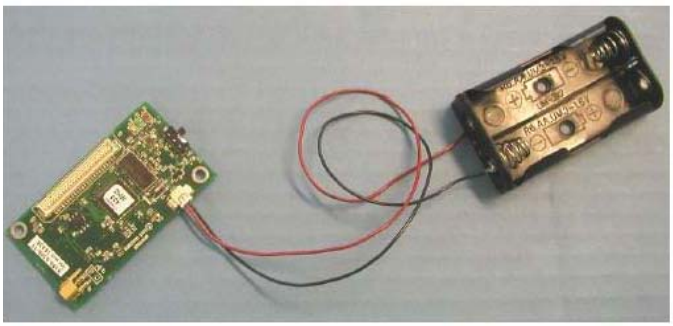
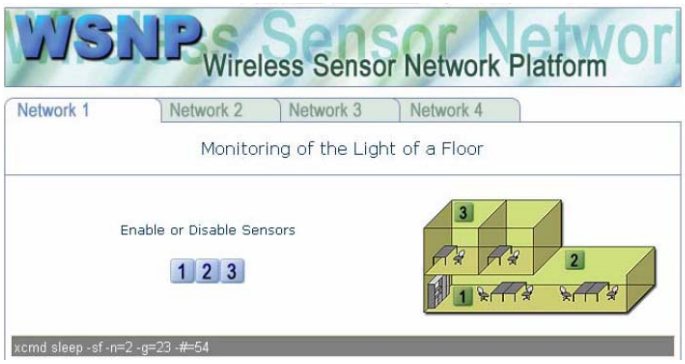
CENTRO INTERNACIONAL DE MÉTODOS NUMÉRICOS EN INGENIERÍA

Programación y Gestión de Redes de Motas Via Internet I

J. Jiménez
A. Priegue
F. Guzmán
E. Oñate





Publicación CIMNE Nº-279, Diciembre 2005

Programación y Gestión de Redes de Motas Via Internet I

J. Jiménez
A. Priegue
F. Guzmán
E. Oñate

Publicación CIMNE Nº-279, Diciembre 2005

Centro Internacional de Métodos Numéricos en Ingeniería
Gran Capitán s/n, 08034 Barcelona, España

ÍNDICE

0. Introducción

Este trabajo presenta los desarrollos llevados a cabo en CIMNE en la programación y gestión de redes de motas utilizando Internet. El trabajo se enmarca dentro de la línea de I+D de CIMNE en el estudio, desarrollo y aplicación de redes de sensores inteligentes sin hilos en diversos ámbitos de la ingeniería. En las referencias [(1-3)] se describen los resultados de investigaciones anteriores del equipo de CIMNE sobre el tema.

El trabajo se ha llevado a cabo en el marco de las actividades del proyecto "*Desarrollo de tecnologías para investigación y gestión de procesos constructivos utilizando redes de sensores sin hilos.(PROSENSOR)"* financiado por el programa PROFIT del Ministerio de Educación y Ciencia. El proyecto se inició en Noviembre de 2005 y su finalización está prevista en Diciembre, 2007.

Los autores agradecen a la empresa Sensoria S.A., el asesoramiento y colaboración en la realización del trabajo.

Referencias

[(1)] J. Jiménez y E. Oñate, *"Plataforma WSN/NIMS - I"*, Publicación CIMNE PI-268, 20pp., 2005

[(2)] J. Jiménez, A. Priegue, J. Piazzese y E. Oñate, *"Plataforma WSN/NIMS II. Primeras experiencias en CIMNE"*, Publicación CIMNE PI-272, 28pp., 2005

[(3)] J. Jiménez, A. Priegue, J. Piazzese y E. Oñate, *"Plataforma WSN/NIMS III. Programación básica de motas"*, Publicación CIMNE PI-277, 20 pp., 2005

1. Desarrollo de una aplicación de programación sobre el aire

En este apartado se explicará el procedimiento para realizar la programación de las motas a través del envío de los paquetes por el aire. Dicha aplicación tiene la ventaja de poder reprogramar las motas sin la necesidad de traer a éstas a la tarjeta de programación Mib510.

A lo largo de esta sección se describirán los siguientes pasos:

- Conceptos de la programación sobre el aire.
- Adaptación de una aplicación.
- Validación de la aplicación Blink.

1.1 Conceptos de la programación sobre el aire

Para la programación sobre el aire, las motas utilizan un servicio llamado "Deluge", el cual se rige por un modelo epidémico para contagiar a todas las motas de una red. Este programa es muy compacto y utiliza 159 bytes de la memoria RAM, y 3.5 kB extra para el programa.

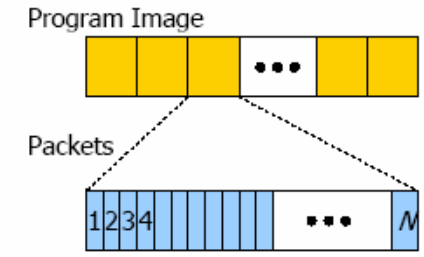
El programa "Deluge" sigue una estrategia de trabajo la cual considera una serie de pre-condicionantes.

- Los nodos están habilitados con el servicio Deluge.
- La librería Deluge está conectada
- "Bootloader" instalado.

Existen dos pasos en la aplicación del servicio "Deluge".

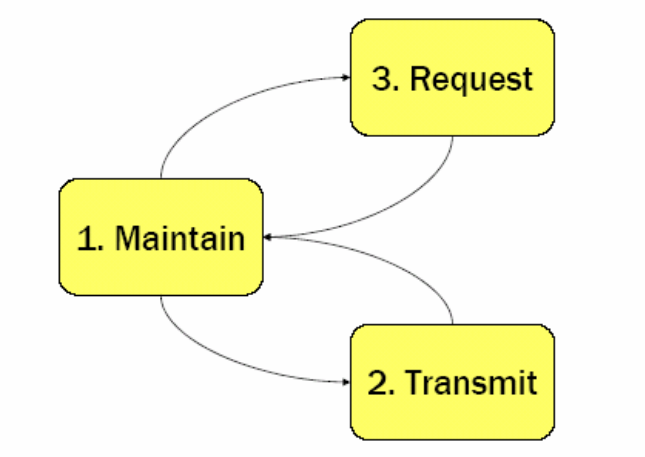
- Infección - Descarga y propagación.
- Reprogramación y reinicio del nodo.

El programa a enviar por el aire a las motas recibe el nombre de "imagen de programa", y dicho programa es dividido en páginas cada una conteniendo N paquetes.



Las ventajas que ofrece la estructura de páginas es que se puede trabajar con memoria RAM limitada, para sólo mantener el estado sobre cuales son los paquetes que se requieren. Así también, la estructura de páginas permite trabajar con multiplexado espacial. Cada imagen de programa está representada por un número de versión único.

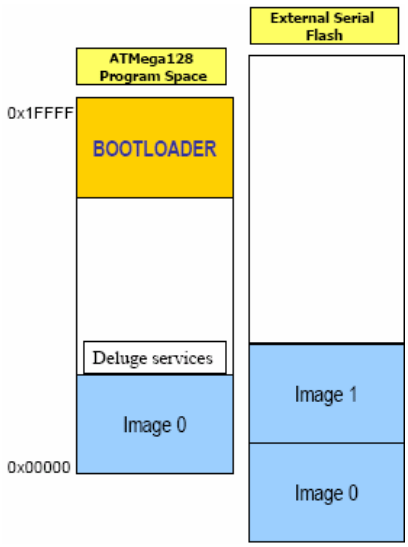
La operación de un nodo bajo los efectos de programación sobre el aire se representa en la figura siguiente. Al principio, el nodo es infectado con una nueva versión de programa y posteriormente éste se encarga de infectar al vecino.



Cada nodo tiene la capacidad de comunicar al nuevo paquete sobre el aire el número de versión de programa con el que cuenta. Si el nodo requiere actualización solicita la descarga del paquete al vecino más próximo que cuente con la información, una vez realizado esto, el nodo actualizado se comunica con sus nodos vecinos que están a su alcance para repetir el proceso en caso de ser necesario.

El Bootloader, es un programa especial para las motas que les permite realizar las siguientes operaciones:

- Toma la imagen de programa de la memoria externa flash.
- Escribe la imagen de programa directamente en la memoria del procesador.
- Se repite hasta que una nueva imagen de código ha sido cargada completamente.
- Reinicia el procesador y el nuevo programa comienza su operación.



1.2 Adaptación de una aplicación

En este apartado utilizaremos la aplicación “Blink”, la cual adaptaremos al servicio “Deluge” con la finalidad de poder enviar este programa a las motas de la red vía aire.

El primer paso será realizar la conexión de la componente “DelugeC” a la aplicación Blink, de manera que se pueda acceder a las interfaces ofrecidas por dicha componente.

```
configuration Blink {  }

implementation
{
  components Main, BlinkM, SingleTimer, LedsC, DelugeC;
  Main.StdControl ->DelugeC;
  Main.StdControl -> SingleTimer.StdControl;
  Main.StdControl -> BlinkM.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

La componente DelugeC, utiliza la interfaz StdControl para inicializarse, comenzar y parar su funcionamiento.

El segundo paso es incluir la librería Deluge en el archivo Makefile de la aplicación.

```
COMPONENT=Blink

TINYOS_NP=BNP

XBOWROOT=%T/../contrib/xbow/tos
PFLAGS= -I$(XBOWROOT)/platform/mica2
include ../MakeXbowlocal
include $(TOSROOT)/tools/make/Makerules
```

Una vez realizados los dos pasos mencionados anteriormente, la aplicación Blink está lista para ser compilada para posteriormente ser enviada a las motas de una red vía aire.

1.3 Validación de la aplicación Blink

El objetivo será programar las motas de una red a través del envío de la información vía aire.

- Realizaremos la compilación y descarga en cada una de las motas del programa “GoldenImage” (El cual representa nuestro servicio Deluge).

  Dicho programa se encuentra en el directorio:
  contrib/xbow/apps/TestDeluge/GoldenImage/

Una vez compilado e instalado en cada una de las motas, contamos con el servicio Deluge en cada una de éstas.

- A continuación se definirá el puerto COM para hacer uso de las herramientas de Java que ofrece Tinyos.

Para la tarjeta Mib510, desde la ventana de Cygwin se ejecutará el siguiente comando:

export MOTECOM=serial@COM1:57600

Para este caso suponemos que el puerto de comunicación es el 1 y que el número de bits por segundo es de 57600.

- Estado de las motas.

Para verificar el ID del nodo y que el programa GoldenImage ha sido correctamente instalado, se ejecuta el siguiente comando en el directorio Java de Tinyos:

tinyos-1.x/tools/java

java net.tinyos.tools.deluge - -ping

El resultado de dicha instrucción se representa en el recuadro que se muestra a continuación.

```
mbaleri@giri-d505 /opt/tinyos-1.x/tools/java
$ java net.tinyos.tools.Deluge --ping
------------------------------------------------------
Openning connection to node ...
Platform COM1:57600 decoded into 1
Connection to node established ...
------------------------------------------------------
Pinging node ...

Reply from node 1:
  Executing image: Golden Image
  Image states:
        Image   Version Pages   Complete
        -----   ------- -----   --------
        0       ------- No Image -------
        1       ------- No Image -------
------------------------------------------------------
```

El recuadro muestra que se está utilizando el puesto COM1 a una velocidad de datos de 57600 bits por segundo. Así también se muestra el ID del nodo y la ejecución del programa “GoldenImage” en este.

Para este caso, el recuadro también indica que la mota no tiene ninguna aplicación sobre el aire descargada. Esto se puede interpretar por medio del parámetro ***“No Image”*** en la columna de Páginas.

Cada una de las motas deberá contar con el programa GoldenImage instalado, tanto la mota que se encontrará en la mib510 como base, así como las motas que se encontrarán desplegadas como miembros de la red de motas.

Una vez instalado GoldenImage, centraremos la atención en la aplicación ***"Blink"***, la cual será inyectada a la red de motas, de forma que cada mota integrante de la red pueda descargar la "imagen de programa" en su memoria del procesador.

Una vez compilada la aplicación ***"Blink"***, el archivo ***main.ihex***, creado en la carpeta buid de la aplicación, tendrá que ser copiado y pegado en la carpeta ***tinyos/tools/java***.

El siguiente paso será inyectar el programa en la red de motas. Para dicho objetivo, se ejecutará un comando especial en el siguiente directorio:

opt/tinyos-1.x/tools/java

java net.tinyos.tools.Deluge - -inject - -ihexfile main.ihex - -imgnum 0

Una vez ejecutada dicha instrucción, aparece el siguiente recuadro que nos indica el progreso de la descarga. El red rojo parpadeando significa que la mota está recibiendo páginas de código.

```
mbaleri@giri-d505 /opt/tinyos-1.x/tools/java
$ java net.tinyos.tools.Deluge --inject --ihexfile main.ihex -
-imgnum 0
------------------------------------------------------
Reading file: main.ihex
Read complete: (Bytes=22252,Sections=1)
Deluge image stats: (Size=23184,Pages=21,Pkts=1008)
------------------------------------------------------
Openning connection to node ...
Platform COM1:57600 decoded into 1
Connection to node established ...
------------------------------------------------------
Upgrading image [0] from version [NONE] to [0] ...
Injecting page [21] of [21] ...
INJECTION COMPLETE!
------------------------------------------------------
```

Conforme la instrucción se ejecuta, se puede observar el comportamiento de los Leds en las motas:

- Led verde, cuando la mota busca por la versión de la información.
- Led rojo parpadeando, cuando la mota recibe páginas de información.

Si una nueva mota es encendida o puesta dentro del rango de recepción, ésta se actualizará automáticamente, siempre y cuando contenga la aplicación GoldenImage, previamente en su memoria.

Si vemos el estado de la mota, por medio del comando:

java net.tinyos.tools.Deluge - -ping

```
mbaleri@giri-d505 /opt/tinyos-1.x/tools/java
$ java net.tinyos.tools.Deluge --ping
------------------------------------------------------
Openning connection to node ...
Platform COM1:57600 decoded into 1
Connection to node established ...
------------------------------------------------------
Pinging node ...

Reply from node 1:
  Executing image: Golden Image
  Image states:
        Image   Version Pages   Complete
        -----   ------- -----   --------
        0       0       21      21
        1       ------- No Image -------
------------------------------------------------------
```

Lo anterior índica que el número de páginas de la aplicación ha sido completado y que la mota tiene el programa listo para ser ejecutado.

Por último para indicar a la mota que empiece a ejecutar el programa nuevo, es necesario indicarlo por medio del siguiente comando desde Cygwin.

java net.tinyos.tools.Deluge - -reboot - -imgnum 0

Este comando repercute en todos los nodos, haciendo la ejecución de la imagen 0, tal y como se indica en la instrucción.

```
mbaleri@giri-d505 /opt/tinyos-1.x/tools/java
$ java net.tinyos.tools.Deluge --reboot --imgnum 0
------------------------------------------------------
-
Openning connection to node ...
Platform COM1:57600 decoded into 1
Connection to node established ...
------------------------------------------------------
-
Reboot message sent.
```

Una vez realizado este paso, podemos ver como la aplicación Blink se empieza a ejecutar sobre cada una de las motas.

2. Desarrollo de una Aplicación de Sensor de Luz

El siguiente apartado tiene el objetivo de analizar y desarrollar una aplicación de sensor de luz basada en los siguientes puntos:

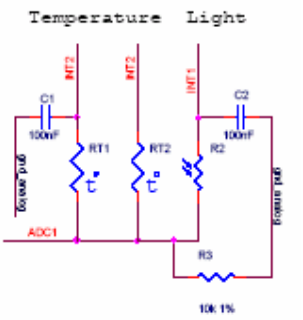
- Arquitectura del Sensor.
- Desarrollo del driver de Sensor
- Interfaz de comunicación UART
- Interfaz de comunicación por RF

Los datos y experimentos contenidos, se han desarrollado bajo la plataforma "***mica2"*** y la tarjeta de sensores "***MTS310"***.

2.1 Arquitectura del Sensor

La tablilla de sensores ***MTS310*** es un dispositivo que contiene 6 sensores distintos, de los cuales ocuparemos solamente el sensor de luz para el desarrollo de nuestra aplicación.

Con la finalidad de poder usar el sensor de luz, la señal digital ***PW1***, debe estar encendida. Así también, la salida del sensor debe estar conectada al puerto ***ADC1***, del convertidor análogo digital. La potencia del sensor de luz es controlada ajustando la señal ***INT1***.



Circuito del sensor de luz (alimentación y salida)

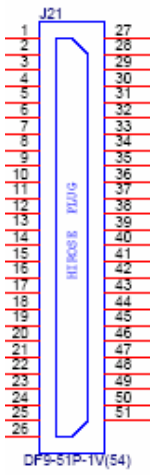
En Tinyos existe una instrucción que permite a los usuarios asignar un nombre a los ***pins*** de conexión entre la tablilla de sensores y la ***mica2***.

- TOSH_ASSIGN_PIN(name, port, bit)
- TOSH_ALIAS_PIN(name, other_name)

Éstos a su vez generan un total de 5 nuevas instrucciones para manipular los ***pins*** de la tablilla de sensors ***MTS310***.

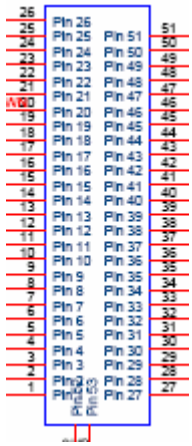
TOSH_SET##name##_PIN()	Pone el pin seleccionado en alto
TOSH_CLR##name##_PIN()	Pone el pin seleccionado en bajo
TOSH_READ##name##_PIN()	Lee el pin seleccionado
TOSH_MAKE##name##_PIN()	Define el pin como una salida
TOSH_MAKE##name##_INPUT()	Define el pin como una entrada

Se presenta a continuación la interfaz física de conexión entre el ***mica2*** y el ***MTS310***. Dicho dispositivo es un "conector de 51 pins", el cual permite que se puedan asignar todas las señales de control y de adquisición de datos.



PIN	NAME
1	GND
2	VSNSR
3	INT3
4	INT2
5	INT1
6	INT0
7	BAT_MON
8	LED3
9	LED2
10	LED1
11	RD
12	WR
13	ALE
14	PW7
15	USART1_CLK
16	PROG_MOSI
17	PROG_MISO
18	SPI_SCK
19	USART1_RXD
20	USART1_TXD
21	I2C_CLK
22	I2C_DATA
23	PWM0
24	PWM1A
25	AC+
26	AC-

PIN	NAME
27	UART_RXD0
28	UART_TXD0
29	PW0
30	PW1
31	PW2
32	PW3
33	PW4
34	PW5
35	PW6
36	ADC7
37	ADC6
38	ADC5
39	ADC4
40	ADC3
41	ADC2
42	ADC1
43	ADC0
44	THERM_PWR
45	THRU1
46	THRU2
47	THRU3
48	RSTN
49	PWM1B
50	VCC
51	GND



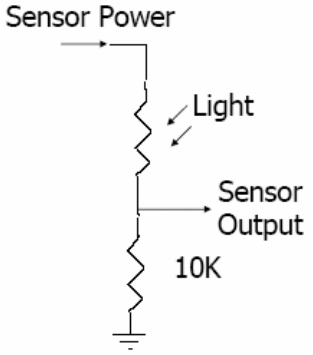
MICA2	Configuración de pins	MTS310

En nuestra aplicación, pondremos especial interés en los pins 5 (INT1) y el pin 42 (ADC1). Los cuales habilitarán al sensor de luz y harán la medición respectivamente.

2.2 Desarrollo del driver de Sensor

Los pasos básicos para el desarrollo de ésta aplicación se desglosan a continuación.

2.2.1. Determinar las interfaces del hardware con el sensor



Energía del sensor = conector INT1 (pin5) en el mica2.

Salida del sensor = conector ADC1 (pin42) en el mica2.

2.2.2. Definir las líneas de entrada / salida con Tinyos

Para el desarrollo de este punto es necesario crear un archivo **.h** dentro de la carpeta de nuestra aplicación, al cual llamaremos "***sensorboard.h***". En este archivo se definirán los nombres de los pins de entrada (energía) y salida (medidas) del sensor.

Para definir el nombre de la línea de entrada que usamos:

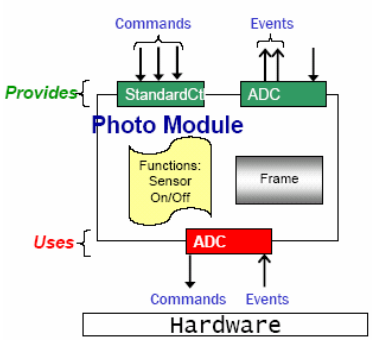
- TOSH_ALIAS_OUTPUT_ONLY_PIN(PHOTO_CTL, INT1);

Para definir el nombre del canal de salida al convertidor AD usamos:

- enum { TOS_ADC_PHOTO_PORT = 1} ;

2.2.3. Crear la componente de Tinyos para controlar el sensor

Una vez asignados los pins que configuran las entradas y salidas del ***MTS310***, es necesario desarrollar la ***componente*** que se encargue de apagar y encender el sensor y así también que se encargue de activar la toma de datos.



La estructura de la componente queda como se muestra en la figura anterior. Ésta provee las interfaces StdControl y ADC. Así también hace uso de la interfaz ADC.

PhotoDriverM.nc

```
includes sensorboard;
module PhotoDriverM {
  provides interface StdControl;
  uses {
    interface ADCControl;
  }
}
implementation {
  command result_t StdControl.init() {
    TOSH_MAKE_PHOTO_CTL_OUTPUT();
    TOSH_SET_PHOTO_CTL_PIN();
    return call ADCControl.init();
  }
  command result_t StdControl.start() {
    TOSH_MAKE_PHOTO_CTL_OUTPUT();
    TOSH_SET_PHOTO_CTL_PIN();
    return SUCCESS;
  }
  command result_t StdControl.stop() {
    TOSH_CLR_PHOTO_CTL_PIN();
    return SUCCESS;
  }
}
```

El ***modulo*** PhotoDriverM.nc, crea una componente que controla el estado de los pins de la tablilla de sensores MTS310.

Por medio de la interfaz StdControl, este programa inicia, arranca y para la actividad del sensor de luz en la tablilla MTS310.

La interfaz ADCControl inicializa las estructuras de trabajo del convertidor AD.

La ***configuración*** PhotoDriver.nc, se encarga de realizar las conexiones entre las interfaces de las componentes a usar.

La energía del sensor se gestiona desde el modulo PhotoDriverM, mientras que la salida del sensor se considera una aplicación externa de la componente DAC.

PhotoDriver.nc

```
includes sensorboard;
configuration PhotoDriver
{
  provides interface ADC as SensorData;
  provides interface StdControl;
}
implementation
{
  components PhotoDriverM, ADCC;

  StdControl = PhotoDriverM;
  SensorData =
ADCC.ADC[TOS_ADC_PHOTO_PORT];
  PhotoDriverM.ADCControl -> ADCC;
}
```

2.2.4. Crear una aplicación simple

En este apartado, se presenta el código para poder utilizar el driver del sensor de luz que se ha desarrollado previamente. El programa ***TestSensorM***, se encarga de iniciar el sensor y tomar la muestra.

module TestSensorM {
provides {
interface StdControl;
}
uses {
interface StdControl as SensorControl;
interface ADC as SensorData;
interface Timer;
interface Leds;
}
}
implementation {
/* Initialize the component */
command result_t StdControl.init()
	Se declaran las interfaces que la componente provee, y también las interfaces que la componente utiliza.
	
	Las interfaces ***StdControl*** y ***ADC*** reciben otro nombre con el comando ***"as"***, esto debido a que la aplicación utiliza las mismas interfaces.
	
	Se inicializan los leds (todos puestos en estado apagado) y la tablilla de sensor "PhotoDriverM"

{
call Leds.init();
call SensorControl.init();
return SUCCESS;
}
/* Start the component. Start the clock */
command result_t StdControl.start()
{
call Leds.redOn();
call SensorControl.start();
call Timer.start(TIMER_REPEAT, 1000);
return SUCCESS;
}
/* Stop the component */
command result_t StdControl.stop()
{
call SensorControl.stop();
return SUCCESS;
}
/* Measure Sensor */
event result_t Timer.fired()
{
call SensorData.getData();
call Leds.redToggle();
return SUCCESS;
}
/* Sensor ADC data ready */
async event result_t
SensorData.dataReady(uint16_t data)
{
call Leds.greenToggle();
return SUCCESS;
}
}
	Se ejecuta el comando ***Start***, el cual provoca el encendido del led rojo, y el arranque de la tablilla de sensores. También se ejecuta el ***Timer***, el cual se repite por intervalos de 1 segundo.
	
	Este código permite que la actividad del sensor se pueda detener.
	
	Una vez ejecutado el ***Timer***, se provoca el evento ***fired***, el cual llama la interfaz ***getData*** de la componente ADC. Una vez tomado el dato, el red rojo se conmuta.
	
	Cuando se dispone del paquete de datos del sensor, el red verde se conmuta y se vuelve a repetir el ciclo ***Timer*** de 1 seg hasta que el programa sea detenido.

Para completar la aplicación es necesario contar con el programa de conexiones ***TestSensor.nc***.

configuration TestSensor
{
// this module does not provide any interface
}
	El programa ***TestSensor*** se encarga de hacer las conexiones entre las componentes primitivas y complejas de la aplicación.

implementation
{
components Main, TestSensorM, LedsC,TimerC,
PhotoDriver as Sensor;
 	
Main.StdControl -> TestSensorM;
Main.StdControl -> TimerC;
 	
// Wiring for New Sensor
 	
TestSensorM.SensorControl->Sensor;
TestSensorM.SensorData -> Sensor;
 	
TestSensorM.Leds -> LedsC;
TestSensorM.Timer -> TimerC.Timer[unique("Timer")];
}
	Se definen las componentes a utilizar dentro de nuestra aplicación, y se asigna un alias a ***PhotoSensor***.
	
	Se define el uso de la interfaz StdControl en las componentes ***TestSensorM*** y ***TimerC***.
	
	Se hace la conexión entre el programa ***TestSensorM*** y ***PhotoDriver***.
	
	Conexión de la componente ***LedsC*** y ***TimerC***. La palabra ***unique***, permite identificar nuestro temporizador en las componentes de la aplicación.

Finalmente, destacamos que en la carpeta de trabajo de nuestra aplicación, se contendrán los siguientes archivos:

Makefile. Define las direcciones de archivos complementarios para la compilación de la aplicación.

Sensorboard.h. Archivo definido por el usuario. Contiene enumeraciones, registros o tipos de datos creados por el usuario.

PhotoDriver. Es el archivo de conexiones entre las componentes del sistema y el programa PhotoDriverM.

PhotoDriverM. Es el archivo que implementa el código que inicia, arranca y detiene el sensor de luz.

TestSensor. Es el archivo de conexiones entre las componentes del driver del sensor de luz y el programa TestSensorM.

TestSensorM. Contiene el código para iniciar el sensor, el temporizador y tomar los datos. Así también contiene el código para el control de los leds.

2.3 Interfaz de comunicación UART

A través de este apartado, se analizará como hacer uso de la información obtenida por medio del sensor de luz, basándonos en el la comunicación a través del puerto ***UART*** del sistema y el comando ***Xlisten***.

Empezaremos definiendo dos conceptos fundamentales:

- ***UART***. Es un circuito que utiliza la ***mica2***, para convertir los datos en paralelo, que manda la UCP, en serie, con el fin de comunicarse con otro sistema externo.

- ***Xlisten***. Es una herramienta diseñada para desplegar las lecturas tomadas por las tarjetas de sensores, a través del puerto serie o por RF.

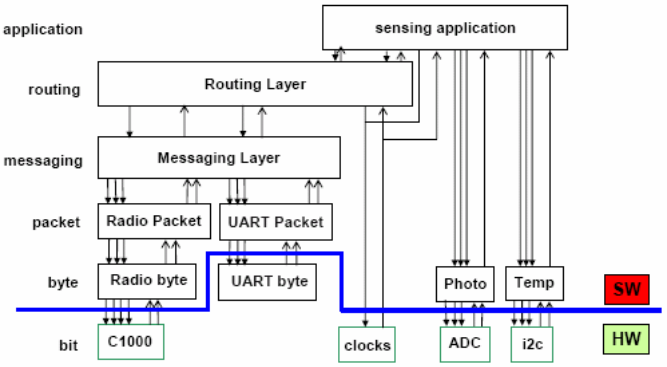
Los datos medidos por el sensor de luz, pueden ser enviados a un ordenador por medio del puerto serie del mismo. La velocidad a la cual se envían estos datos, entre la mica2 y el ordenador, es de 57600 bits por segundo.

En el apartado 2, se han creado los programas para el driver del sensor (PhotoDriver) y los programas de validación del mismo (TestSensor). Para realizar las pruebas del apartado 3 -enviar los datos medidos por el puerto serie y desplegarlos en el ordenador- añadiremos algunas líneas de código al programa ***TestSensorM*** y a las conexiones de ***TestSensor***.

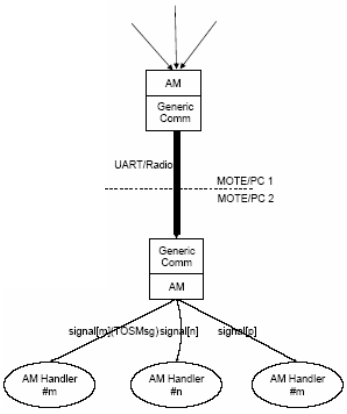
2.3.1 Nueva componente GenericComm

La componente ***GenericComm*** será integrada a las conexiones definidas en el archivo TestSensor.nc, posteriormente el programa TestSensor.M.nc, hará uso de las interfaces que proporciona dicha componente.

GenericComm es una componente que dirige toda la información de los Mensajes Activos (AM) en las motas hacia el UART o el medio de Radio Frecuencia (RF).



Organización de Tinyos

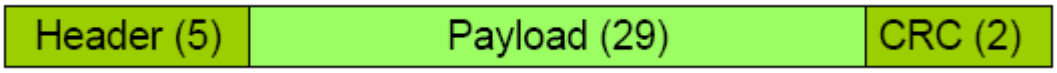


GenericComm

Los diferentes tipos de Mensajes Activos se identifican por medio de los Canales de Datos, de tal manera que la información irá directamente a los componentes que la necesitan.

2.3.2 Estructura de mensajes en Tinyos

Los mensajes que Tinyos envía por el puerto serie o el medio de RF siguen la estructura que se presenta a continuación.



- Header: Esta parte está integrada por 5 bytes, los cuales se utilizan de la siguiente manera.
  - Dirección de destino: 2 bytes.
  - Tipo de Mensaje Activo (AM): 1 byte.
  - ID del grupo: 1 byte.
  - Longitud de datos (Payload): 1 byte.
- Payload: Se pueden enviar hasta 29 bytes de datos de los resultados de las actividades de la aplicación.
- CRC: Se utilizan 2 bytes, y pueden ser empleados para monitorizar la media de error de bit.

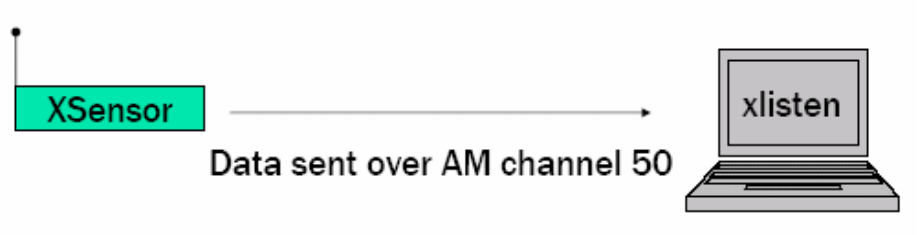
Existen un par de protocolos que permiten aumentar la fiabilidad de transmisión de los datos dentro del sistema Tinyos.

- Serial Framer Protocol. Apoya en asegurar la fiabilidad del grupo de bytes transmitido sobre el puerto UART.

- Serial Forwarder Protocol. Apoya en el re-envío de comunicación del puerto serie sobre un puerto de Internet.

2.3.3 Código para el puerto UART

Bajo esta sección, se agregará el código necesario para habilitar el envío de datos por el puerto UART.



- Conexiones de la componente GenericComm en TestSensor.nc.

En este apartado se definen las interfaces para el envío de mensajes, así como la estructura de los mensajes.

#define AM_TYPE 50	Se define el tipo de Mensaje Activo.
Components GenericComm as Comm;	Se agrega la componente asignándole un alias.
TestSensorM.CommControl -> Comm; TestSensorM.SendMsg -> Comm.SendMsg[AM_TYPE];	Se hacen las conexiones para facilitar las interfaces ***CommControl*** y ***SendMessage***.

- A continuación se estructura la salida del paquete ***XSensor*** de la aplicación.

En las especificaciones del archivo TestSensorM.nc agregamos las interfaces de mensajes:

```
interface StdControl as CommControl;
interface SendMsg;
```

En la sección de implementación del archivo TestSensorM.nc declaramos las variables del mensaje TOS:

```
typedef struct DataMsg {
        uint8_t  board_type;
        uint8_t  packet_id;
        uint8_t  node_id;
        uint8_t  parent;
        uint16_t batt;
        uint16_t temp;
        uint16_t light;
} __attribute__ ((packed)) DataMsg;
```

Finalmente se define el espacio de memoria y variables globales.

```
TOS_Msg         msg_buff;
TOS_MsgPtr      msg_ptr;
norace uint16_t sens;
```

- Envío del TOSmsg sobre el puerto UART.

`#define MSG_LEN 29`
 	
`task void send_msg(){`
 	
`DataMsg* data = (DataMsg*)msg_ptr->data;`
`  call Leds.yellowToggle();`
 	
`  data->batt = bat;`
`  data->light = sens;`
`  data->board_type = SENSOR_BOARD_ID;`
`  data->packet_id  = 1;`
`  data->node_id = TOS_LOCAL_ADDRESS;`
`  data->parent = 0;`
 	
`call SendMsg.send(TOS_UART_ADDR, MSG_LEN, msg_ptr);  }`
	Longitud del mensaje en bytes.
	
	Función send_msg de la componente GenericComm.
	
	Asignación de los parámetros de identificación y de medición del sensor.
	
	Definición del puerto UART como destino de la información.

Con el uso de la interfaz StdControl, se inicializa la componente GenericComm.

```
command result_t StdControl.init()
{
    atomic msg_ptr = &msg_buff;
    call Leds.init();
    call SensorControl.init();
    call CommControl.init();
    call BatteryControl.init();
    return SUCCESS;
 }

 command result_t StdControl.start()
 {
    call Leds.redOn();
    call SensorControl.start();
    call CommControl.start();
    call BatteryControl.start();
    call Timer.start(TIMER_REPEAT, 1000);
    return SUCCESS;
 }
```

La información del sensor se coloca en un TOSMsg una vez llegado el evento dataReady. Posteriormente se llama una tarea para enviar el paquete.

```
 async event result_t SensorData.dataReady (uint16_t data)
{
    call Leds.greenToggle();
    sens = data;
    post send_msg();
    return SUCCESS;
 }
```

Finalmente se define la interfaz SendDone.

```
event result_t SendMsg.sendDone(TOS_MsgPtr msg, result_t success)
{
   call Leds.yellowToggle();
   return SUCCESS;
}
```

- Validación del código implementado.

Para comprobar que el código funciona correctamente, es necesario realizar los siguientes pasos:

- Conectar la Mib510 al puerto serie del ordenador.
- Instalar un Mica2 sobre la Mib510.
- Desde el Cygwin, ir a la carpeta donde se encuentra la aplicación.
- Compilar la aplicación “make mica2”
- Descargar la aplicación “make mica2 reinstall.0”

Para poder ver la lectura del sensor de luz, es necesario que se llame al comando ***Xlisten*** desde la interfaz ***Cygwin***.

2.4 Interfaz de comunicación por RF

En este apartado se realizarán los cambios necesarios al código de comunicación por el puerto ***UART***, de tal manera que éste pueda transmitir por Radio Frecuencia. El objetivo es tener comunicación inalámbrica entre una mota y la estación base.

El programa ***TOSBase*** es una aplicación que permite identificar mensajes de las motas enviados por radio frecuencia. Por tal motivo, este programa será instalado en la mota base para que ésta pueda actuar como un rastreador de señales.

Cualquier paquete en el medio, que sea identificado por la mota base, será enviado por el puerto UART al ordenador.

- Envío de mensaje por radio a través de la componente GenericComm.

Es necesario realizar una pequeña modificación para activar la comunicación vía radio frecuencia. Se cambia el parámetro ***TOS_UART_ADDR*** por ***TOS_BCAST_ADDR***.

```
task void send_msg()
    {
      DataMsg* data = (DataMsg*)msg_ptr->data;
      call Leds.yellowToggle();
      data->batt = bat;
      data->light          = sens;
      data->board_type = SENSOR_BOARD_ID;
      data->packet_id  = 1;
      data->node_id    = TOS_LOCAL_ADDRESS;
      data->parent     = 0;
      call SendMsg.send(TOS_BCAST_ADDR, MSG_LEN, msg_ptr);
     }
```

- Consideraciones del rango de radio frecuencia.

Es necesario definir la radio frecuencia en la cual trabajan las motas, 433Mhz, 916Mhz o 2400Mhz. Para hacer esto, se edita el archivo Makexbowlocal.

- Validación del código implementado.

Para comprobar que el código funciona correctamente, es necesario realizar los siguientes pasos:

- Conectar la Mib510 al puerto serie del ordenador.
- Colocar un Mica2 sobre la Mib510 e instalar la aplicación del sensor de luz.
- Colocar un Mica2 sobre la Mib510 e instalar la aplicación TOSBase.

Para poder ver la lectura del sensor de luz, es necesario que se llame al comando ***Xlisten*** desde la interfaz ***Cygwin***.

3. Consumo energético en motas

Todas las motas están diseñadas para ser alimentadas por baterías. Las formas del MICA2 y del MICAz están diseñados para funcionar con dos baterías AA, aún así cualquier combinación de baterías (AAA, C, D, etc.) puede ser utilizada, siempre considerando que la alimentación debe ser de 2.7 VDC a 3.6 VDC.

El MICA2DOT cuenta con un diseño para funcionar con una pequeña batería de forma de moneda, aunque cualquier combinación de baterías (AAA, C, D, etc.) puede ser utilizada bajo la condición de voltaje de 2.7VDC a 3.6VDC.

A continuación se presenta una tabla con las especificaciones de baterías para cada uno de los tipos de motas.

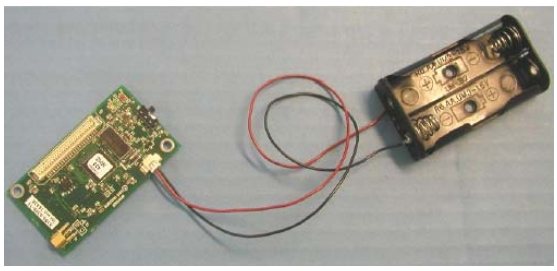
Mota	Batería	Capacidad (mA-hr)	Voltage
MICAz	2 AA	2000, alcalina	2.7 a 3.6
MICA2	2 AA	2000, alcalina	2.7 a 3.6
MICA2DOT	De moneda	560, Li-ion	2.7 a 3.6

Se debe tener especial cuidado al seleccionar la batería y su capacidad para poder cubrir los requerimientos de energía de las motas y los períodos requeridos de su funcionamiento. También se debe considerar que el rango de temperatura de trabajo y la degradación de capacidad sean adecuadas para los ciclos de trabajo de la mota. La siguiente tabla proporciona información sobre los consumos de corriente de los diferentes componentes del sistema.

Corriente de operación	MICAz	MICA2	MICA2DOT
ATMega 128L, operando	8 mA	8 mA	6 mA
ATMega 128L, durmiendo	8 uA	8 uA	8 uA
Radio, reciviendo	19.7 mA	8 mA	8 mA
Radio, transmitiendo (1mW)	17 mA	12 mA	12 mA
Radio, durmiendo	2 uA	2 uA	2 uA
Memoria, escritura	15 mA		
Memoria, lectura	4 mA		
Memoria, durmiendo	2 uA		

Los MICA2 y MICA2DOT también pueden ser alimentados externamente por medio de:

1. El conector de 51 pines se puede alimentar la corriente y el neutro a la unidad.
2. El conector de 2 pines “Molex”. (Mostrado a continuación)



En la siguiente tabla se muestra un ejemplo para predecir la vida útil de una batería en las motas. La hoja de cálculo utilizada para realizar el cálculo puede ser encontrada en el *‘Anexo’* de este documento.

ESPECIFICACIONES DEL SISTEMA		
DISPOSITIVO	CONSUMO	CICLO DE TRABAJO
• Procesador		
A toda operación	8 mA	1
Durmiendo	8 uA	99
• Radio		
Recibo	8 mA	0.75
Transmisión	12 mA	0.25
Durmiendo	2 uA	99
• Memoria		
Escritura	15 mA	0
Lectura	4 mA	0
Durmiendo	2 uA	100
• Sensor		
Operación	5 mA	1
Durmiendo	5 uA	99
mA-hr usados cada hora		
	Procesador	0.0879
	Radio	0.0920
	Memoria	0.0020
	Sensor	0.0550
	Total (mA-hr) usado	0.2369
Vida de batería vs. Tamaño de batería		Meses
	2000	11.73
	560	3.28

La tabla anterior muestra el consumo de corriente para una mota en una hora de trabajo. Cabe considerar que los ciclos de trabajo de la mota a toda operación pueden parecer muy cortos, sin embargo hay que recordar que las tareas que se ejecutan son del orden de micro - mili segundos.

En la mayoría de las aplicaciones de las motas, el procesador y la radio funcionan por un período muy breve de tiempo, seguidos por un ciclo de dormir. Mientras la mota duerme, el consumo de corriente es del orden de micro amperes, distinto al de mili amperes cuando la mota está operando. El ciclo de dormir ayuda a minimizar el consumo de corriente la mayor parte del tiempo, y cuando la mota despierta para procesar, transmitir o recibir información, sólo aparecen picos de consumo de corta duración. Este método de optimización de energía prolonga la vida de la batería, aún así, debido a las fluctuaciones de corriente en la batería, se reduce un mínimo la capacidad de la misma.

Para los detalles de la tabla anterior, se puede consultar el *'Anexo'* de este documento.

4. WSNP- Plataforma web para el monitoreo de redes sensoriales

Como ya se detalló en el informe precedente al actual, '*Plataforma WSN/ NIMS III :Programación básica de motas* ' WSNP es una plataforma Web que está siendo desarrollada en CIMNE, con el objetivo de tener un sistema distribuido capaz de monitorear redes sensoriales situadas en cualquier punto del mundo, así como poder interactuar con ellas a distancia, para lograr reprogramar, por ejemplo, la funcionalidad de los sensores, apagar y encender remotamente las motas, que funcionan como nodos transmisores de la red, etc.

4.1 Administración

Actualmente, se está ampliando la funcionalidad de la plataforma, para poder administrar diferentes niveles de usuarios, a quienes se les asignará permisos para poder visualizar y manipular las redes a las que tenga permiso acceder.

También se podrá dar de alta/baja y gestionar diferentes proyectos, los cuales tendrán asignado como mínimo una red sensorial, ubicada en una localización, que podrá ser un edificio, terreno, etc. Todo ello con la manipulación requerida de imágenes que representarán las plantas del edificio o el mapa geográfico del terreno, con el que el usuario interactuará para poder ubicar en ellos los nodos sensoriales, modificarlos de lugar, así como acceder de manera visual a cada mota, para trabajar con ellas, ya que podrá reprogramarlas, enviarle una serie de comandos para apagarlas, encenderlas, resetearlas, cambiar la velocidad de transmisión, etc.

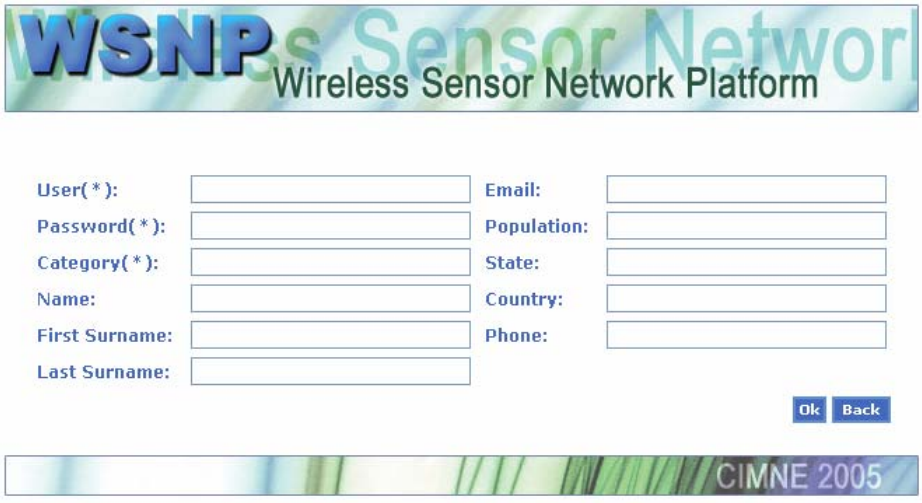
En primer lugar, para acceder a la plataforma, el usuario requiere de un *login* y *password* que deberá introducir en la pantalla de acceso, que se muestra en la siguiente pantalla.

Una vez el usuario se ha autenticado accederá, dependiendo de su categoría, a diferentes funcionalidades de la plataforma. Si el usuario tiene rol de super-administrador podrá gestionar los usuarios de la aplicación, así como todos los proyectos que contiene. Mientras que si el usuario es un usuario asignado a uno o varios proyectos, sólo podrá visualizar la parte de monitoreo de la red sensorial de éstos y dependiendo de sus privilegios, habrá funcionalidades que podrá usar y otras que no.

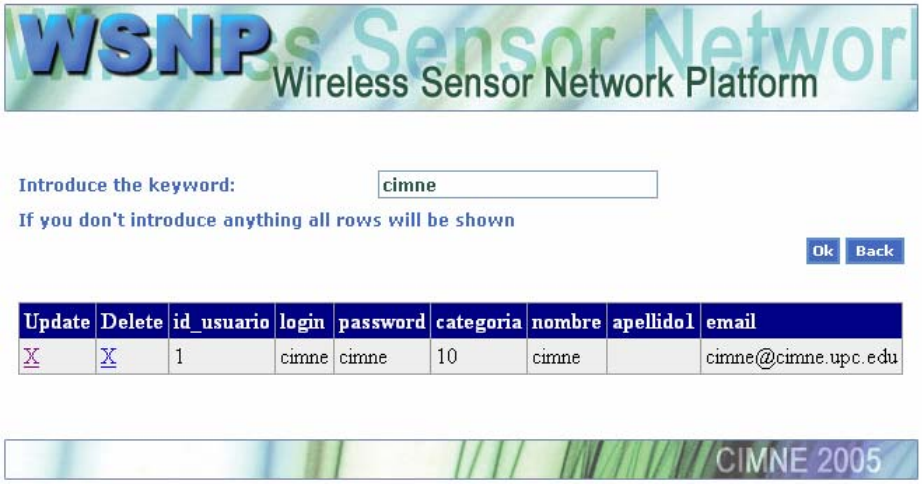
En la siguiente pantalla aparece el back-office de gestión total de la plataforma, donde el super-administrador puede acceder a la gestión de usuarios, proyectos de redes o directamente a la parte visual de monitorización de cualquier red insertada.

4.1.1 Administración de usuarios

En esta sección el super-administrador puede gestionar toda la información que se requiere de un usuario de Wireless Sensor Network Platform, pudiendo darle de alta, de baja y modificarlo. La siguiente pantalla muestra los datos a rellenar para añadir un nuevo usuario en el sistema.



Para poder modificar o borrar un usuario se deberá acceder a la pantalla de búsqueda, donde el super-administrador introduciendo una palabra clave localizará al usuario deseado.



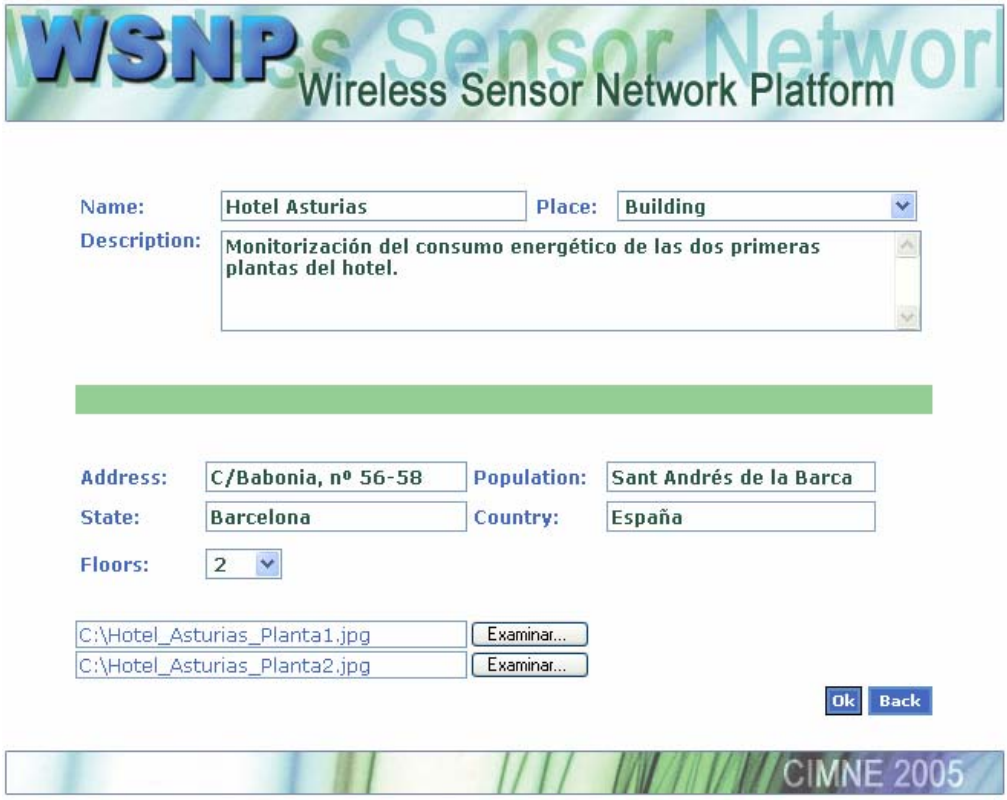
Una vez localizado el usuario, se puede acceder haciendo click en las casillas de *update* y *delete* a las correspondientes pantallas de modificación y borrado de usuarios, respectivamente.

4.1.2 Administración de proyectos de redes

En esta sección el super-administrador o el administrador del proyecto puede gestionar toda la información que se requiere de un proyecto de red sensorial.

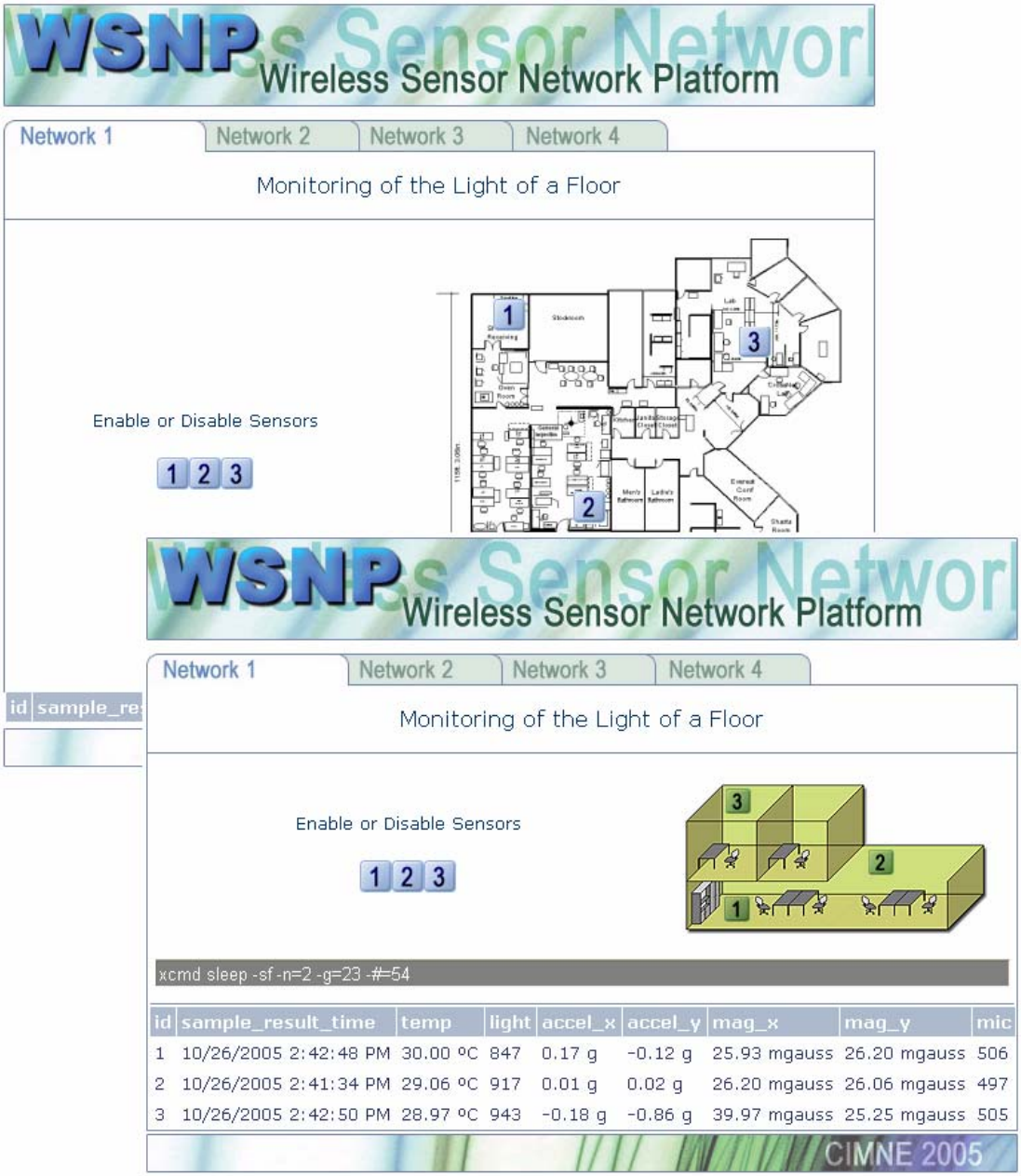
Siguiendo la misma filosofía que en la interfaz de gestión de usuarios, se podrá buscar un proyecto por palabra clave para poderlo modificar y/o borrar, así como acceder a la pantalla de inserción de proyectos nuevos.

En la pantalla de inserción de un nuevo proyecto, el usuario, dependiendo del tipo de localización dónde se ubicará la red, edificio o terreno, deberá cumplimentar diferentes campos de información, así como seleccionar las imágenes que corresponden a las plantas del edificio o a la zona geográfica del terreno.

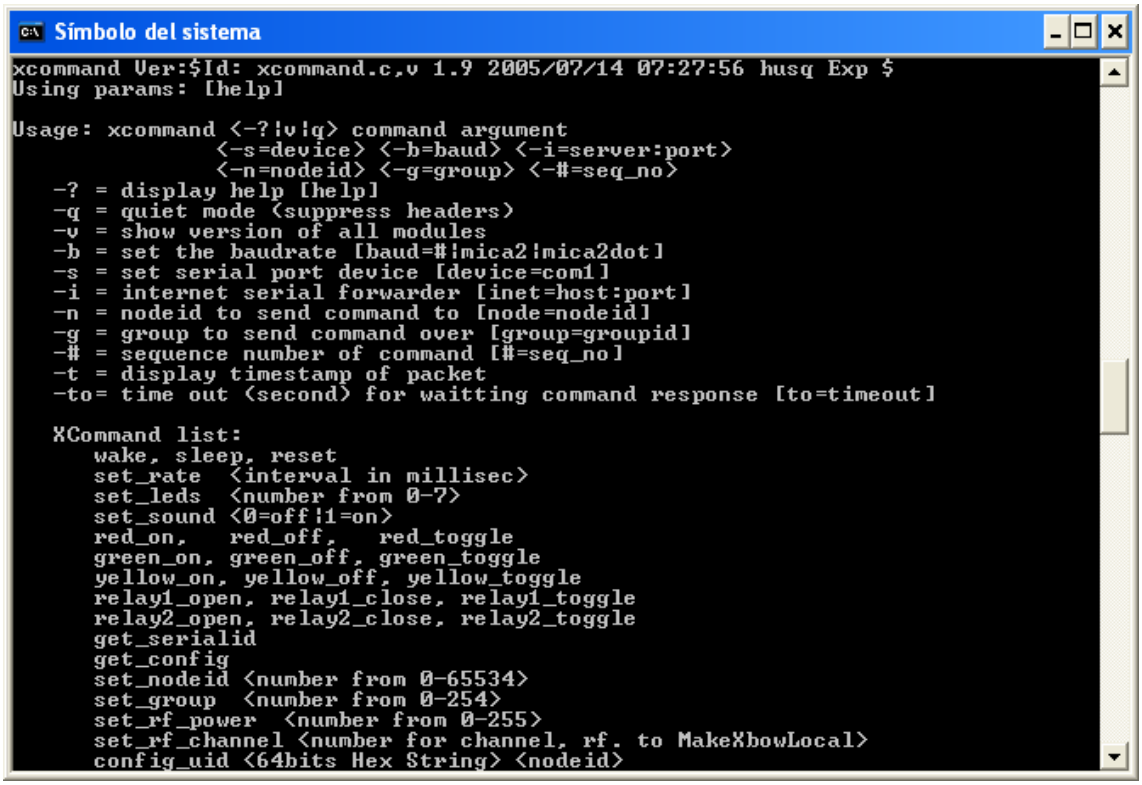


4.2 Monitoreo de una red

En esta parte de la plataforma el usuario, si tiene acceso y privilegios, podrá observar el estado de la red sensorial, los datos obtenidos por los sensores y podrá enviar comandos a ejecutar en las motas de la red.



Como se puede observar en las imágenes anteriores, en las pantallas de monitoreo de red, el usuario puede enviar comandos utilizando la instrucción *xcmd* por el puerto serie, siempre y cuando esté ejecutándose en *background* el *xserve*, que es el que habilita la disponibilidad del puerto, sin importar si la aplicación está recibiendo en el mismo instante datos de los sensores.



Actualmente, en la versión beta de la plataforma ya está implementada la funcionalidad de encender y apagar las motas desde la Web, que se traduce en llamar al comando *xcmd* con los argumentos correspondientes:

xcmd sleep/wake –sf –n=[id_nodo] –g=[grupo_nodo] -#=[secuencia]

En la pantalla anterior se puede observar el significado de cada uno de los argumentos utilizados en la sentencia. Cabe recalcar que la asignación del argumento # debe ser un número que se incremente secuencialmente en cada llamada pues si no malfunciona el comando.

Otros comandos que serán implementados en breve permitirán al usuario, desde la plataforma, modificar el tiempo de muestreo de los sensores de la red, activar/desactivar el zumbador, así como los leds, resetear las motas a distancia, cambiar los identificadores internos de las motas, modificar el número que identifica el grupo, esto último será muy útil para cuando se tenga que instalar varias redes cercanas que deban convivir sin interferir una con las otras y queramos que unos determinados nodos que forman parte de una de las redes, pasen a formar parte de la otra, sin tener que tocar físicamente las motas.

Por último, una vez finalizada la implementación del envío de comandos, se habilitará la plataforma para la reprogramación de las motas remotamente, implementando en la aplicación lo que ya se ha conseguido realizar localmente, detallado en el primer apartado de este mismo informe *'Desarrollo de una aplicación de programación sobre el aire'*.

5. Anexo

Estimate of baterry life operation for a MOTE						
	mA-hr	% trabajo-hr	mA-hr used each hr	Total	Time (min)	Time (Seg)
Processor						
Current full operation	8	1	0.08		0.6	36
Current sleep	0.008	99	0.00792	0.0879	59.4	3564
Radio						
Current to receive	8	0.75	0.06		0.45	27
Current to transmit	12	0.25	0.03		0.15	9
Current to sleep	0.002	99	0.00198	0.0920	59.4	3564
Logger Memory						
Current to write	15	0	0		0	0
Current to read	4	0	0		0	0
Current to sleep	0.002	100	0.002	0.0020	60	3600
Sensor Board						
Current full operation	5	1	0.05		0.6	36
Current sleep	0.005	99	0.00495	0.0550	59.4	3564

Total Current Used (mAmp-hr), each hour	0.2369

Daily consumption (24 hrs)		mA-hr / day
		5.684

Typical Battery Capacity		mA-hr	Voltage Range
AA (2) - Alkaline		2000	2.7 to 3.6
Coin, Li-ion		560	2.7 to 3.6

Battery life	Years	Months (30 days)	Weeks	Days	Hours
AA (2) - Alkaline	0.98	11.73	50.26	351.84	8444.16
Coin, Li-ion	0.27	3.28	14.07	98.5	2364.37