ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Desarrollo De Una Herramienta Para La Generación De Interfaces Gráficas Con Redes De Sensores Inalámbricas

AUTOR: Antonio Rosa Rodríguez.
DIRECTOR(ES): Fernando Losilla lópez.

Septiembre / 2008

Autor	Antonio Rosa Rodríguez.
E-mail del Autor	blankiroji@hotmail.com.
Director(es)	Fernando Losilla López.
E-mail del Director	flosilla@hotmail.com
Codirector(es)	
Título del PFC	**Desarrollo De Una Herramienta Para La Generación De Interfaces Gráficas Con Redes De Sensores Inalámbricas**
Descriptores	Redes de sensores.

Resúmen

El objetivo fundamental del proyecto es el control y monotorización de todos los nodos de la red desde un PC, los cuales se conectan con la red a través de un nodo cuya función es de hacer de conexión entre el PC y la red de sensores. La única función del resto de sensores es ejecutar los comandos que son enviados desde la aplicación principal y enviar la respuesta al PC. Lo que se persigue es poder crear el GUI de forma sencilla a partir de una serie de clases, ahorrando la mayor cantidad de trabajo posible. De la misma manera se proveerá un GUI genérico que permita controlar la red de sensores sin tener que modificar el código que se ejecuta en el PC ni, por tanto, aprender a usar los lenguajes de programación y librerías con los que se ha desarrollado.

Titulación	ITTET
Intensificación	Telemática.

Departamento	TIC
Fecha de Presentación	Septiembre-2008

A mis padres

A mi hermanos

A mis compañeros y amigos

A mi director de proyecto

Por vuestro apoyo, paciencia y confianza, ***gracias***

Índice general

Índice general 3

Índice de Figuras 6

INTRODUCCIÓN Y OBJETIVOS 8

CAPÍTULO 1: Redes de Sensores Inalámbricas(WSN, Wireless Sensor Network)

1.1 Introducción……...12

1.2 Caracteristicas de las WSN…….....14

1.3 Requisitos para una WSN……..16

1.4 Arquitectura de las WSN……...18

1.4.1 Arquitectura Centralizada……...18

1.4.2 Arquitectura Distribuida…….19

1.5 Protocolos de las WSN……..19

1.6 ZigBee, estandar WSN……..22

1.7 Problemas de als WSN……..26

1.8 Aplicaciones de las WSN……..27

1.9 Plataformas Hardware……...32

1.9.1 Micaz……..32

1.9.2 Micas2……33

1.9.3 Micad2dot……...34

1.9.4 Telos…….35
1.10 Resumen Comparativo……..37

CAPÍTULO 2: TinyOS

2.1 Introducción……..39
2.2 Introducción a TinyOS…….39
2.3 NesC…….40
2.4 TinyOS……..42
2.4.1 Tipos de Componentes……...44
2.4.2 Ejemplo de un Componente…….45
2.4.3 Implementación de un Componente……47
2.4.4 Ejemplo de una aplicación completa……...47
2.4.5 Modelo de comunicación……48
2.5 Herramientas de TinyOS……..48
2.5.1 Listen……48
2.5.2 SerialForwarder……48
2.5.3 NesDoc…….49
2.5.4 Motelist……50
2.5.5 MIG(Message Interfaz Generator)……...50
2.5.6 TOSSIM/TinViz……..50
2.5.7 TinyDB…….51
2.5.8 Surge View……...52
2.5.9 Oscope……..52
2.6 Deluge……...53
2.6.1 Conjunto de Herramientas para Deluge……54
2.6.2 Detalles de la programación de red……..55
2.7 TinySchema……..56

2.7.1 Atributos, Comandos y Eventos……56

2.7.2 Componentes Attr, Command, Event, Tipos y estructuras definidas en TinySchema……...57

2.7.3 Ejemplos Practicos de Atributos y Comandos…….61

2.8 Componentes utilizados en la creación de los Atributos y Comandos……66

2.8.1 Atributos……66

2.8.2 Comandos……..70

2.9 Compilar y ejecutar en TinyOS……...71

2.10 Conclusiones…….71

CAPÍTULO 3: Aplicación Desarrollada.

3.1 Introducción…….73

3.2 Programación de los Nodos Sensores de la Red……..74

3.2.1 Estructura de los Mensajes……...74

3.2.2 Estructura de la Aplicación Telos.NC……..78

3.3 Programación de la GUI “WSNMonitor”……80

3.3.1 Clases Java que Controlan las Comunicaciones Con la Red De Sensores Inalámbricas…….80

3.3.2 Librerias para la creacion de la Interfaz Gráfica de Usuario……...84

3.3.3 Deluge……..86

3.4 Manual de Uusario de de la GUI “WSNMonitor”……...86

3.5 Ejemplos prácticos del uso de la GUI “WSNMonitor”……...89

CAPÍTULO 4: Conclusiones y Líneas Futuras

4.1 Conclusiones…….....101

4.2 Líneas Futuras……102

BIBLIOGRAFÍA……..103

19.1.0

Índice de Figuras

Figura 1. Red de sensores inalámbrica……12
Figura 2. Elementos de una WSN……15
Figura 3: Arquitectura WSN centralizada……18
Figura 4: Arquitectura WSN distribuida……..19
Figura 5: Niveles físico, red y aplicación……22
Figura 6: Arquitectura ZigBee……..23
Figura 7: Tramas……...24
Figura 8: Reconocimiento del enemigo……...29
Figura 9: MICAz……..32
Figura 10: Diagrama de Bloques……..33
Figura 11: Placa de desarrollo MIB510……33
Figura 12: Mica2……..34
Figura 13: Diagrama de Bloques. ……34
Figura 14: Mica2dot……35
Figura 15: Diagrama de Bloques……..35
Figura 16: TelosB……36
Figura 17: Diagrama de Bloques……..36
Figura 18: Evolución de los motes……...37
Figura 19: Comparativa de tiempos y consumos……38
Figura 20.Aplicación. ……..41
Figura 21.Compilación de una aplicación TinyOS……..42
Figura 22. Modelo de componentes de TinyOS y su interacción……43
Figura 23. Grafo de componentes para un sistema MultiHop……..45
Figura 24. Representación gráfica de la componente “Active Messages” (AM)…….46
Figura 25. Archivo que describe la componente……..46
Figura 26: Aplicación Surge……47
Figura 27. SerialForwarder……...49
Figura 28.TinyViz…….....51
Figura 29. Osciloscopio……52
Figura 30. Estructura AttrDesC……59
Figura 31. Estructura AttrDescs……...59
Figura 32. Estructura CommandDesc……...59
Figura 33. Estructura CommandDescs…….....60
Figura 34. Estructura EventDesc……..60
Figura 35. Estructuras EvenDescs y EventDescsPtr……60
Figura 36. Módulo AttrGlobalM.NC……62
Figura 37. Continuacion Módulo AttrGlobalM.NC……63
Figura 38. Configuración AttrGlobal……...63
Figura 39. Módulo CommandPotM.NC……..65
Figura 40. Configuración CommandPot.NC……65
Figura 41. Configuración Atributo HamaTelos……67
Figura 42. Componentes de la aplicación HamaTelos……67
Figura 43. Configuración Atributo Humidity……..68
Figura 44. Componentes del Atributo TempTelos……..68
Figura 45. Configuración Atributo VoltajeInterno……..69
Figura 46. Componentes del Atributo VoltIntM……69
Figura 47. Configuración Atributo Temperatura interna……69

Figura 48. Componente Atributo TempeInt…….70
Figura 49. Atributo ApTelosM…….70
Figura 50. Comando CommandSolar……...71
Figura 51. Componentes del Comando ComandSolar…….71
Figura 52. Estructura de los Mensajes……..75
Figura 53. Estructura Mensaje ListaAttr……..75
Figura 54. Nombres de Comandos……...76
Figura 55.CommandMsg……..77
Figura 56. Envio de Mensajes CommandMsg…….77
Figura 57. Estructura Mensaje ResetMsg……78
Figura 58. AM_TYPES……78
Figura 59. Estructura Telos.NC ……...79
Figura 60.MoteIF……..82
Figura 61.Message Listener……..82
Figura 62. Manejador de Mensajes Recibidos…….....83
Figura 63. Clase Lista Grupos……..84
Figura 64. WSNMonitor……...90
Figura 65. Nodos activos en la red de sensore…….91
Figura 66. Comandos ……...92
Figura 67. Seleccionar Comando…….93
Figura 68.Valor de la Temperatura……..93
Figura 69. Modificar Grupo de red……...94
Figura 70.Grupo de Red modificado……95
Figura 71.Seleccionar Comando……..95
Figura 72. Abrir Deluge……...96
Figura 73. Ventana Deluge……...97
Figura 74. Ping Deluge……98
Figura 75. Selecciónar número de imagen……...99
Figura 76. Seleccionar nombre de imagen……...99
Figura 77. Descarga de imagen……..100

INTRODUCCIÓN Y OBJETIVOS

Las redes de sensores inalámbricas se perfilan como una de las tecnologías más prometedoras en los próximos años en al ámbito del control y de la toma de medidas de parámetros físicos del entorno de forma masiva. Para ello se han de diseñar nodos autónomos capaces de medir/actuar, procesar información y comunicarla de forma inalámbrica a través de los otros nodos de la red, hasta una estación base de recogida y análisis de datos donde se provoca una actuación en consecuencia. Existen múltiples aplicaciones en todos aquellos ambientes en donde se pretenda observar y/o controlar el estado de dispositivos, seres vivos o parámetros ambientales de manera constante, autónoma, no intrusiva y durante largos periodos de tiempo.

Las redes de sensores inalámbricas (Wireless Sensor Network, WSN) están formadas por un conjunto de dispositivos que permiten una comunicación sin cables, interconectados entre sí a través de una red inalámbrica y a su vez conectados a un sistema central en el que se recopilará la información recogida por cada uno de los sensores.

Este proyecto persigue desarrollar una aplicación que permita visualizar mediante el uso de una interfaz gráfica en Java, los sensores que tenemos desplegados por la red, realizar consultas de valores de sus sensores, modificar parámetros que controlan el funcionamiento de los sensores, permitir mostrar graficas de la evolución de los valores a medir por los sensores, y por último reprogramar los sensores de forma inalámbrica.

El título de este Proyecto Final de Carrera es “Desarrollo de un herramienta para la generación de interfaces gráficas con redes de sensores inalámbricas”. El objetivo fundamental del proyecto es el control y monitorización de todos los nodos de la red desde un PC, los cuales se conectan con la red a través de un nodo cuya función es de hacer de conexión entre el PC y la red de sensores. La única función del resto de sensores es ejecutar los comandos que son enviados desde la aplicación principal y enviar la respuesta al PC. Si bien ésta es la forma habitual en que trabajan las redes de sensores, en este proyecto se pretende facilitar el desarrollo de interfaces gráficas de usuario (GUI). El desarrollo de estas, normalmente se realiza a medida de la aplicación que se desarrolla, teniendo que emplear un esfuerzo importante. Lo que se persigue es poder crear el GUI de forma sencilla a partir de una serie de clases, ahorrando la mayor cantidad de trabajo posible. De la misma manera se proveerá un GUI genérico que permita controlar la red de sensores sin tener que modificar el código que se ejecuta en el PC ni, por tanto, aprender a usar los lenguajes de programación y librerías con los que se ha desarrollado.

La ejecución de las distintas tareas en los nodos sensores es llevaba a cabo por un conjunto de componentes previamente definidos siguiendo unas normas predefinidas que permiten la comunicación con el PC y bajo las cuales pueden ser desarrollados nuevos componentes que realicen tareas distintas a las que se presentan este proyecto.

El desarrollo de este proyecto ha implicado el uso de numerosas tecnologías relacionadas con las redes de sensores y la programación. Los nodos empleados son los ampliamente extendidos TelosB, que se han programado haciendo uso del sistema operativo para redes de sensores TinyOS y su lenguaje de programación nesC, una

ampliación del lenguaje C estándar orientada a componentes. Asimismo se han empleado otras subsistemas de este sistema operativo que han permitido realizar tareas como la reprogramación inalámbrica de la red (Deluge) o el aislamiento de tareas en componentes que las ejecutan (TinySchema) creando atributos para la lectura de datos (valor medido por los sensores y parámetros de funcionamiento de los nodos) y comandos para su modificación o la ejecución de acciones por parte de los nodos. Para la GUI ejecutada en el PC se ha empleado java junto con la librería gráfica Swing, la librería de comunicaciones proporcionada por TinyOS y otras herramientas del sistema operativo que se describirán a lo largo de este documento.

Los objetivos que se han perseguido durante la realización de este proyecto han sido los siguientes:

- Fácil adición de funcionalidades a la aplicación, las cuales se encapsularán en componentes que se ejecutan en los nodos de la red de sensores. Estas funcionalidades deben implementarse en componentes aislados de forma que puedan ser fácilmente añadidas o eliminadas de la aplicación que se ejecuta.

- Implementación de las funcionalidades básicas para las redes de sensores, como la lectura de los distintos tipos de sensores que integran los nodos y la lectura y escritura de parámetros de funcionamiento de una red de sensores como por ejemplo la potencia de transmisión.

- Desarrollo de un interfaz de usuario genérico así como de clases java básicas que permitan la elaboración más adelante de interfaces de usuario con mayor nivel de personalización.

- Auto-descubrimiento por parte del GUI tanto de los nodos que forman parte de una red como de las funcionalidades que estos ofrecen.

- Elaboración de facilidades en el GUI que permitan simplificar el manejo de una red de sensores, como la reprogramación inalámbrica de la red, el manejo simultáneo de varias redes de sensores o la representación gráfica de datos y su almacenamiento.

Por último explicar de forma breve el contenido de los capítulos que contiene esta memoria.

En el Primer capítulo se describen las redes de sensores inalámbricas, explicando sus características principales : requisitos, tipos de arquitecturas, protocolos que se usan en las redes de sensores inalámbricas, problemas que se presentan en las redes de sensores inalámbrica, distintos ámbitos de aplicación, y para terminar este capítulo las diversas plataformas hardware que podemos encontrar de nodos sensores.

En Segundo capítulo se describe el sistema operativo empleado, TinyOS, así como los distintos subsistemas que lo forman y que han sido usados en este proyecto. Además se da una serie de indicaciones para su uso en una red de sensores. Destacar que se hace un

especial hincapié en las herramientas TinySchema y Deluge ya que han sido parte fundamental para la elaboración de este proyecto.

El Tercer capítulo trata de explicar de forma especifica la herramienta desarrollada, desde dos puntos de vista, que son: por un lado lo que confiere a los sensores, y por otro lado a lo que el la aplicación que se ejecuta en el PC. Se comienza explicando la parte referente a las comunicaciones desde la red de sensores hacia el PC, continuando posteriormente con la aplicación ejecutada en el PC. También se explica su estructura, librerías empleadas y clases y componentes desarrollados. Se explica como se usa la aplicación, sus funcionalidades y se muestran capturas de pantalla para describir un ejemplo real.

El Cuarto capítulo es donde una vez explicado todo el proyecto, se da paso a una reflexión con unas conclusiones y, se abre una línea de trabajo para continuar con este proyecto con un apartado de líneas futuras.

REDES DE SENSORES INALÁMBRICAS (WSN, WIRELESS SENSOR NETWORK)

1.1. Introducción

Las redes de sensores están formadas por un grupo de sensores con ciertas capacidades sensitivas y de comunicación inalámbrica los cuales permiten formar redes *ad hoc* sin infraestructura física preestablecida ni administración central.

Las redes de sensores es un concepto relativamente nuevo en adquisición y tratamiento de datos con múltiples aplicaciones en distintos campos tales como entornos industriales, domótica, entornos militares, detección ambiental.

Esta clase de redes se caracterizan por su facilidad de despliegue y por ser autoconfigurables, pudiendo convertirse en todo momento en emisor, receptor, ofrecer servicios de encaminamiento entre nodos sin visión directa, así como registrar datos referentes a los sensores locales de cada nodo. Otra de sus características es su gestión eficiente de la energía, que les permite obtener una alta tasa de autonomía que las hacen plenamente operativas.

La miniaturización de ordenadores creciente dio a luz la idea de desarrollar computadoras extremadamente pequeñas y baratas que se comunican de forma inalámbrica y se organizan autónomamente. La idea de estas redes es repartir aleatoriamente estos nodos en un territorio grande, el cual los nodos observan hasta que sus recursos energéticos se agoten

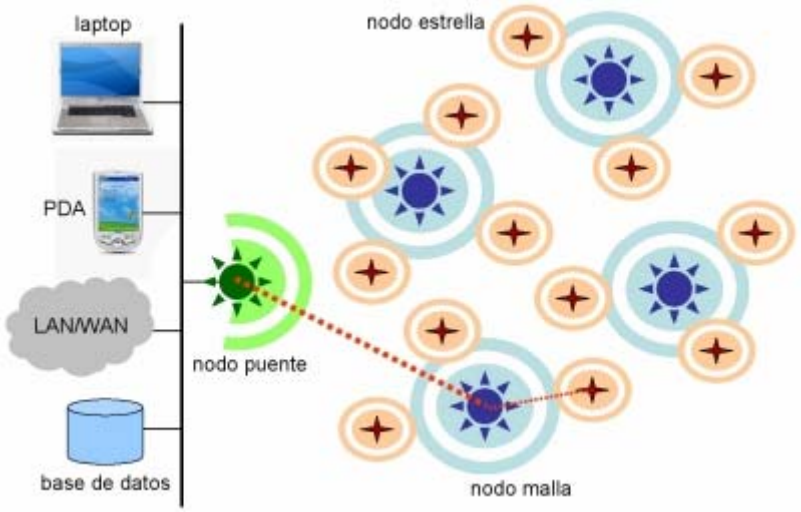


Figura 1. Red de sensores inalámbrica

Las redes de sensores con cables no son nuevas y sus funciones incluyen medir niveles de temperatura, líquido, humedad etc. Muchos sensores en fábricas o coches por ejemplo, tienen su propia red que se conecta con un ordenador o una caja de controles a través de un cable y, al detectar una anomalía, envían un aviso a la caja de controles.

La diferencia entre los sensores que todos conocemos y la nueva generación de redes de sensores inalámbricas es que estos últimos son inteligentes, es decir, capaces de poner en marcha una acción según la información que vayan acumulando, y no están limitados geográficamente por un cable fijo.

Los nuevos avances en la fabricación de microchips de radio, nuevas formas de rotures y nuevos programas informáticos relacionados con redes están logrando eliminar los cables de las redes de sensores, multiplicando así su potencial.

El ámbito de aplicación de este tipo de sistemas, como veremos, es muy amplio: monitorización de entornos naturales, aplicaciones para defensa, aplicaciones médicas en observación de pacientes, etc. El motivo del éxito de este tipo de redes de sensores se debe a sus especiales características físicas.

A los nodos de las redes se les imponen unas restricciones de consumo severas. El motivo de la imposición de estas restricciones es la necesidad de que los nodos sean capaces de operar, por sí mismos, durante periodos largos de tiempo, en lugares donde las fuentes de alimentación son si no inexistentes, de baja potencia. El tamaño es otra restricción que cada vez se hace más necesaria para la mayoría de las aplicaciones, de manera que las tarjetas o nodos que forman las redes de sensores sean cada vez de menor tamaño.

Desde el punto de vista del software, para la realización de las aplicaciones para redes de sensores, la Universidad de Berkeley e Intel han desarrollado una plataforma específica para este tipo de sistemas, que tiene en cuenta las restricciones de los nodos. En particular se ha desarrollado un sistema operativo, llamado TinyOS, cuya característica principal reside en que al ser modular resulta ideal para instalarse en sistemas con restricciones de memoria.

Se desarrolló también un lenguaje de programación, llamado nesC, de sintaxis muy parecida a C, basado en componentes, y a partir del cual se rediseño una primera versión de TinyOS de modo que actualmente está íntegramente implementado sobre nesC. Tanto nesC como TinyOS están basados en componentes e interfaces bidireccionales.

Además, actualmente, Berkeley e Intel han desarrollado diversas aplicaciones a modo de ejemplo, simuladores de ejecución, y varias universidades internacionales están dedicando esfuerzos al desarrollo de aplicaciones usando esta emergente tecnología.

Existen otras empresas que son proveedores de esta tecnología, el mayor de estos es Crossbow Technology, que ha desarrollado redes de sensores a gran escala para su uso comercial. Las últimas investigaciones apuntan hacia una eventual proliferación de redes de sensores inteligentes, redes que recogerán enormes cantidades de información hasta ahora no registrada que contribuirá de forma favorable al buen funcionamiento de fabricas, al cuidado de cultivos, a tareas domesticas, a la organización del trabajo y a la predicción de desastres naturales como los terremotos. En este sentido, la computación que penetra en

todas las facetas de la vida diaria de los seres humanos esta a punto de convertirse en realidad.

Si los avances tecnológicos en este campo siguen a la misma velocidad que han hecho en los últimos años, las redes de sensores inalámbricas revolucionarían la capacidad de interacción de los seres humanos con el mundo.

1.2. Características de las WSN

Los recientes avances en microelectrónica, wireless y electrónica digital han permitido el desarrollo de nodos sensores de bajo coste, reducido tamaño, bajo consumo y que se comunican de forma inalámbrica.

El desarrollo de estos nodos sensores ha dado la posibilidad de crear redes basadas en cooperación de los nodos, con una notable mejora sobre redes de sensores tradicionales, que se suelen desplegar de dos modos:

Sensores que se encuentran lejos del fenómeno, grandes y muy complejos para distinguir el objetivo del ruido del entorno.

Muchos sensores con posición y topología cuidadosamente seleccionada. Transmiten datos de adquisición a nodos centrales que realizan la computación.

Como ya hemos comentado, las WSN se componen de miles de dispositivos pequeños, autónomos, distribuidos geográficamente, llamados nodos sensores con capacidad de cómputo, almacenamiento y comunicación en una red conectada sin cables, e instalados alrededor de un fenómeno objeto para monitorizarlo.

Una vez se produzcan eventos, toma de medidas o cualquier actividad programada con el fenómeno en cuestión los nodos enviaran información a través de la red, hasta llegar a un sistema central de control que recogerá los datos y los evaluara, ejecutando las acciones pertinentes en comunicación con otros sistemas o en la propia red de sensores.

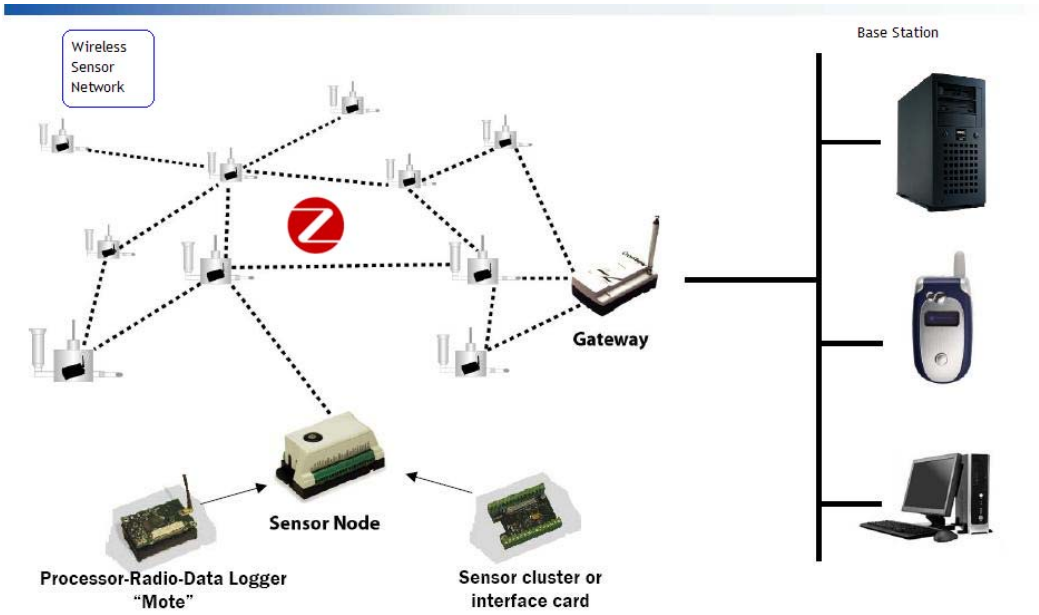


Figura 2. Elementos de una WSN

Tal y como vemos en esta figura podemos establecer, por tanto, una serie de elementos

que componen de forma general una WSN:

1. Sensores.

De distinta naturaleza y tecnología, toman del medio la información y la convierten en señales eléctricas.

2. Nodos sensores.

Son los procesadores de radio, que toman los datos del sensor a través de sus puertas de datos, y envían la información a la estación base.

3. Pasarelas o Gateways.

Elementos para la interconexión entre la red de sensores y una red TCP/IP.

4. Estaciones base.

Recolector de datos basado en un ordenador común o sistema integrado.

5. Red inalámbrica.

Típicamente basada en el estándar 802.15.4 - ZigBee.

Las redes de sensores tienen una serie de características propias y otras adaptadas de las redes Ad-Hoc:

• **Topología Dinámica**

En una red de sensores, la topología siempre es cambiante y éstos tienen que adaptarse para poder comunicar nuevos datos adquiridos.

• **Variabilidad del canal**

El canal radio es un canal muy variable en el que existen una serie de fenómenos como pueden ser la atenuación, desvanecimientos rápidos, desvanecimientos lentos e interferencias que puede producir errores en los datos.

• **No se utiliza infraestructura de red**

Una red de sensores no tiene necesidad alguna de infraestructura para poder operar, ya que sus nodos pueden actuar de emisores, receptores o enrutadores de la información. Sin embargo, hay que destacar en el concepto de red de sensores, la figura del nodo recolector (también denominados sink node), que es el nodo que recolecta la información y por el cual se recoge la información generada normalmente en tiempo discreto. Esta información generalmente es adquirida por un ordenador conectado a este nodo y es sobre el ordenador que recae la posibilidad de transmitir los datos por tecnologías inalámbricas o cableadas según sea el caso.

• **Tolerancia a errores**

Un dispositivo sensor dentro de una red de sensores tiene que ser capaz de seguir funcionando a pesar de tener errores en el sistema propio.

• **Comunicaciones multisalto o broadcast**

En aplicaciones de redes de sensores siempre es característico el uso de algún protocolo que permita comunicaciones multi-hop, léase AODV, DSDV, EWMA u otras, aunque también es muy común utilizar mensajería basada en broadcast.

• **Consumo energético**

Es uno de los factores más sensibles debido a que tienen que conjugar autonomía con capacidad de proceso, ya que actualmente cuentan con una unidad de energía limitada. Un nodo sensor tiene que contar con un procesador de consumo ultra bajo así como de un transceptor radio con la misma característica, a esto hay que agregar un software que también conjugue esta característica haciendo el consumo aún más restrictivo.

• **Limitaciones hardware**

Para poder conseguir un consumo ajustado, se hace indispensable que el hardware sea lo más sencillo posible, así como su transceptor radio, esto nos deja una capacidad de proceso limitada.

• **Costes de producción**

Dada que la naturaleza de una red de sensores tiene que ser en número muy elevada, para poder obtener datos con fiabilidad, los nodos sensores una vez definida su aplicación, son económicos de hacer si son fabricados en grandes cantidades

1.3. Requisitos para una WSN

Para que una red pueda funcionar de acuerdo con las anteriores características surgen una serie de retos que la aplicación debe resolver. Estos, que se detallan a continuación, son los requisitos no funcionales del sistema:

- **Eficiencia energética**

Es uno de los asuntos más importantes en redes de sensores. Cuanto más se consiga bajar el consumo de un nodo mayor será el tiempo durante el cual pueda operar y, por tanto, mayor tiempo de vida tendrá la red. La aplicación tiene la capacidad de bajar este consumo de potencia restringiendo el uso de la CPU y la radio FM. Esto se consigue desactivándolos cuando no se utilizan y, sobre todo, disminuyendo el número de mensajes que generan y retransmiten los nodos.

- **Autoorganización**

Los nodos desplegados deben formar una topología que permita establecer rutas por las que mandar los datos que han obtenido. Los nodos necesitan conocer su lugar en esta topología, pero resulta inviable asignarlo manualmente a cada uno, debido al gran número de estos. Es fundamental, por tanto, que los nodos sean capaces de formar la topología deseada sin ayuda del exterior de la red. Este proceso no sólo debe ejecutarse cuando la red comienza su funcionamiento, sino que debe permitir que en cada momento la red se adapte a los cambios que pueda haber en ella.

- **Escalabilidad**

Puesto que las aplicaciones van creciendo con el tiempo y el despliegue de la red es progresivo, es necesario que la solución elegida para la red permita su crecimiento. No sólo es necesario que la red funcione correctamente con el número de nodos con que inicialmente se contaba, sino que también debe permitir aumentar ese número sin que las prestaciones de la red caigan drásticamente.

- **Tolerancia a fallos**

Los sensores son dispositivos propensos a fallar. Los fallos pueden deberse a múltiples causas, pueden venir a raíz del estado de su batería, de un error de programación, de condiciones ambientales, del estado de la red, etc. Una de las razones de esta probabilidad de fallo radica precisamente en el bajo coste de los sensores. En cualquier caso, se deben minimizar las consecuencias de ese fallo. Por todos los medios se debe evitar que un fallo en un nodo individual provoque el mal funcionamiento del conjunto de la red.

- **Tiempo real**

Los datos llegan a su destino con cierto retraso. Pero algunos datos deben entregarse dentro de un intervalo de tiempo conocido. Pasado éste dejan de ser válidos, como puede pasar con datos que impliquen una reacción inmediata del sistema, o se pueden originar problemas serios como ocurriráıa si se ignora una alarma crítica. En caso de que una aplicación tenga estas restricciones debe tomar las medidas que garanticen la llegada a tiempo de los datos.

- **Seguridad**

Las comunicaciones wireless viajan por un medio fácilmente accesible a personas ajenas a la red de sensores. Esto implica un riesgo potencial para los datos recolectados y para el funcionamiento de la red. Se deben establecer mecanismos que permitan tanto proteger los datos de estos intrusos, como protegerse de los datos que estos puedan inyectar en la red.

Según la aplicación que se diseñe algunos de los anteriores requisitos cobran mayor importancia. Como ejemplo podemos considerar una aplicación que controle el comportamiento de animales salvajes dentro de un parque natural. Para determinar su localización, cada uno de estos animales llevaría sujeto un pequeño sensor. En esta situación la capacidad de autoorganización cobrarıa gran importancia. Sin embargo, si pensamos en una red con nodos inmóviles, como los usados en la red domótica de una oficina, este mismo atributo sería de menor importancia.

Es necesario encontrar el peso que cada uno de estos requisitos tiene en el diseño de la red, pues normalmente unos requisitos van en detrimento de otros. Por ejemplo, dotar a una red de propiedades de tiempo real podríıa implicar aumentar la frecuencia con la que se mandan mensajes con datos, lo cual repercutiráıa en un mayor consumo de potencia y un menor tiempo de vida de los nodos.

Esto lleva a buscar, para cada aplicación, un compromiso entre los requisitos anteriores que permita lograr un funcionamiento de la red adecuado para la misión que debe realizar.

1.4. Arquitecturas de las WSN

Tomando como elementos principales de la red a los nodos sensores, las pasarelas (gateway) y las estaciones base, podemos distinguir dos tipos principales de arquitecturas.

1.4.1 Arquitectura centralizada

En este tipo de arquitectura los nodos de una red que estudian un fenómeno enviaran sus datos directamente a la pasarela más cercana, que dirige el tráfico de esa red en concreto.

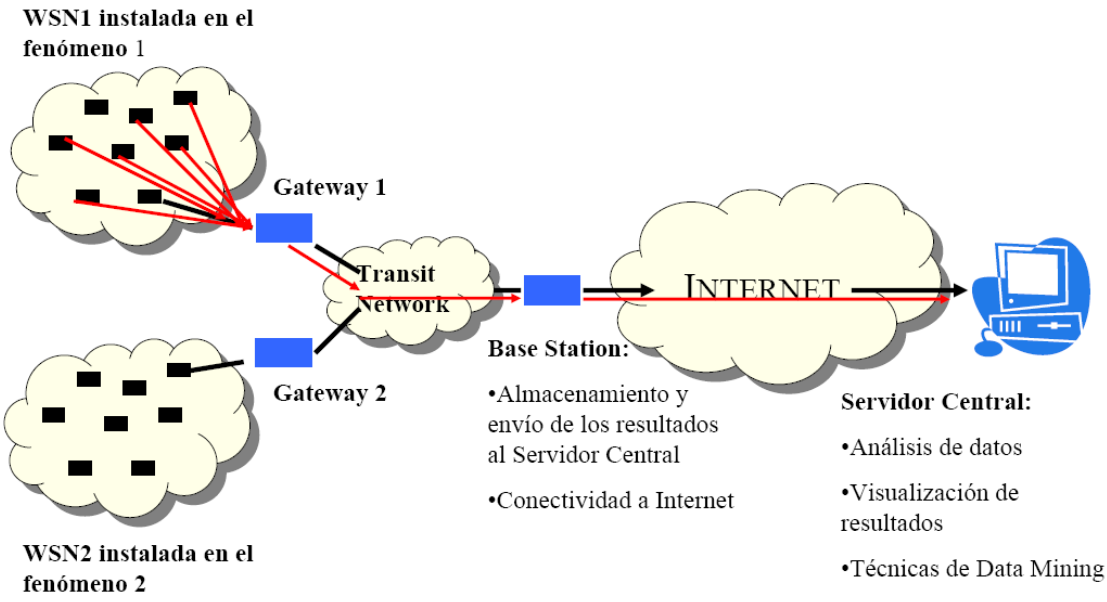


Figura 3: Arquitectura WSN centralizada

Si tenemos en cuenta que el ciclo de vida de un nodo consiste en despertarse, medir, transmitir y dormirse, y cada vez que transmita su mensaje ira a la pasarela, estaremos creando dos grandes problemas para la red:

1. Cuello de botella en las pasarelas.
2. Mayor consumo de energía por las comunicaciones.

Como resultado, el tiempo de vida de la red será relativamente corto.

1.4.2 Arquitectura distribuida

Dada la naturaleza intrínseca de las redes de sensores, normalmente se tiende a este tipo de arquitectura con una computación distribuida. De hecho, como comentábamos en las carácterısticas principales de una WSN, son redes basadas en cooperación de los nodos.

Los nodos sensores se van a comunicar entre sus nodos vecinos y van a cooperar entre ellos, ejecutando algoritmos distribuidos para obtener una única respuesta global que un nodo (cluster head) se encargara de comunicar a la estación base a través de las pasarelas pertinentes.

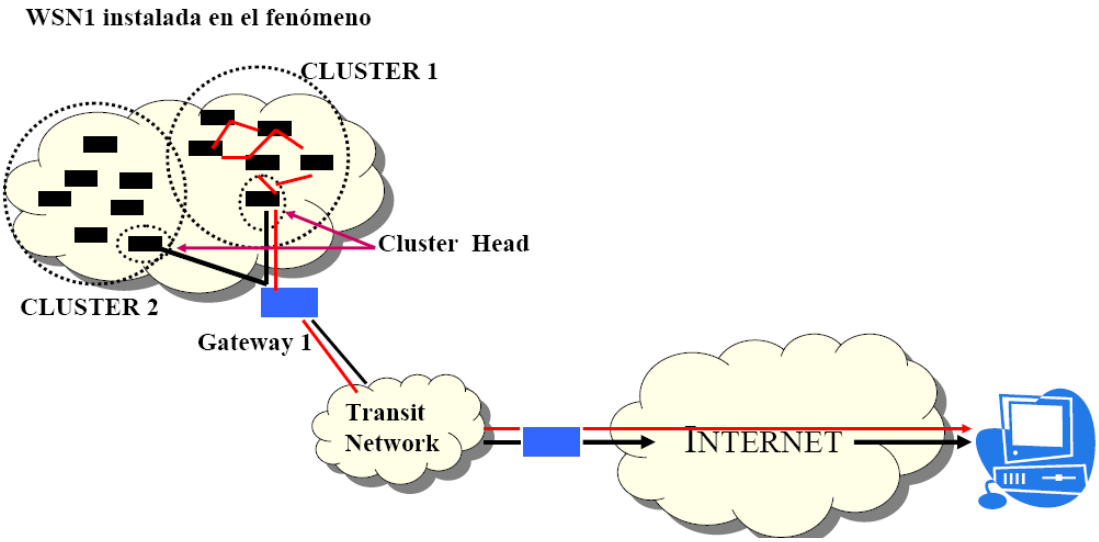


Figura 4: Arquitectura WSN distribuida

De esta forma se evitan los problemas que surgían en la arquitectura centralizada y, Además, se mantienen las características y ventajas que comentamos al comienzo de este capítulo.

1.5. Protocolos de las WSN

¿Por qué son diferentes las WSN?

A continuación vamos a ver por qué son diferentes las WSN de las redes tradicionales y por qué existen algoritmos y protocolos que no se ajustan a las redes de sensores, ya que se encuentran diferencias fundamentales en los principales objetivos de ambas redes.

Las WSN utilizan una comunicación inalámbrica y, obviamente, son diferentes de las Redes cableadas. También son diferentes de las tradicionales redes inalámbricas como las redes celulares, Bluetooth o las móviles ad hoc (MANETS). En estas redes, el objetivo es optimizar el rendimiento y el retardo. Aunque las MANETS comparten las características de desarrollo ad hoc y autoconfiguración de los nodos, el consumo de potencia no es una prioridad...Bluetooth también trata los mismos problemas de limitación de potencia pero el grado de bajo consumo de potencia que se requiere en las WSN es

mucho mayor. Además, los nodos sensores son frecuentemente expuestos a extremas condiciones ambientales, haciéndolos propensos a frecuentes fallos en los nodos. Esto conlleva unas restricciones estrictas en las WSN, no como en las otras redes.

- **Capa física**

Aunque la transmisión en las WSN puede ser por infrarrojos, radio o medio óptico, la banda industrial, científica y médica de los 915MHz (ISM) se ha hecho muy popular en las redes de sensores. La deficiencia de usar infrarrojos o medios ópticos para la trasmisión es que requieren que los nodos transmisor y receptor mantengan una línea de contacto visible. Sin embargo, algunas especificaciones inalámbricas como Bluetooth, HomeRF, y las redes LAN Wireless especificadas por el IEEE 802.11b operan todas en la frecuencia de los 2.4GHz.

- **Capa de enlace**

La responsabilidad de la capa de enlace es establecer un enlace fiable o una infraestructura de red sobre la que los datos puedan ser encaminados. Existen especificaciones como Bluetooth que utilizan la multiplexación por división en el tiempo (TDMA) con saltos de frecuencia mientras que las redes LAN inalámbricas especificadas por el 802.11b utilizan un método de acceso al medio con detección de portadora y evitando colisiones (CSMA/CA).

La necesidad de un nuevo protocolo de capa MAC para WSN radica en que las dificultades de las WSN son muy distintas de los problemas que tenían las especificaciones existentes. Por ejemplo, el alcance de un piconet, que es una colección de ocho nodos, en una red Bluetooth es de 32 pies, mientras que el alcance requerido es mucho más pequeño en una WSN. En las redes celulares, las estaciones base forman una columna cableada proporcionando una estructura parcial a la red. En las WSN, no hay estaciones base.

Un protocolo de capa MAC para las WSN es el llamado MAC autoorganizado para redes de sensores (SMACS), que configura la capa de enlace. El algoritmo de escucha y registro (EAR) permite a los nodos sensores móviles interconectar nodos estacionarios. SMACS actúa al crear la red detectando los nodos vecinos usando transferencia de mensajes. En SMACS, un canal se define con un par de intervalos de tiempo. La detección de vecinos y la asignación de canales se combinan en una fase, para que cuando los nodos vayan a escuchar a sus vecinos, ya hayan formado una red conectada. No hay jerarquías asumidas en SMACS y por esto se forma una topología llana. El algoritmo EAR tiene el problema del control de la movilidad cuando se introducen nodos móviles en la red.

- **Capa de red**

El encaminamiento en las WSN es bastante similar al de los protocolos ad hoc en las redes MANETS. La diferencia es que en los algoritmos de enrutamiento ad hoc, el consumo de potencia es secundario. En redes Bluetooth, las comunicaciones de un nodo master con siete esclavos definen un piconet. Cuando los piconets están interconectados para formar redes dispersas, las diferentes topologías limitan a los nodos que las forman. Para una WSN con la posibilidad de cientos de nodos, esto no será suficiente. Y no sólo eso, sino que también el coste proyectado de un dispositivo Bluetooth es menos de $4

mientras que el precio estimado de un nodo sensor es de menos de $1. Además, los requisitos de potencia de los nodos sensores son mucho menores que para Bluetooth.

Varios algoritmos se han propuesto para el encaminamiento de WSN. El principal objetivo de los algoritmos es el bajo consumo. Se pueden clasificar como aquellos que determinan:

1. Rutas con los nodos de mayor potencia a lo largo de la ruta.
2. Rutas que consuman la mínima energía.
3. Rutas con el mínimo número de saltos.
4. Rutas en las que la mínima potencia disponible es la máxima entre todos los demás caminos.

- **Capa de transporte**

La necesidad de una capa de transporte radica en que las WSN necesitan ser conectadas a una red más grande, como Internet. Las WSN se conectan a Internet por medio de pasarelas. El protocolo de la capa de transporte que conecta el usuario con la pasarela podríıa ser TCP o UDP, ya existentes.

Sin embargo, el protocolo que conecta la pasarela y los nodos sensores tendría que ser diferente ya que no hay un esquema de direccionamiento global en una red de sensores. La limitación de memoria en los nodos sensores conlleva una cuestión importante en tanto que se prefieren protocolos que requieran un bajo almacenamiento de información de estado antes que otros del tipo TCP.

- **Capa de aplicación**

Los nodos sensores tienen muchas aplicaciones distintas. Diseñar una capa de aplicación tiene el mérito de que las WSN pueden ser conectadas a grandes redes como Internet. El direccionamiento de nodos es una cuestión importante aquíı ya que, no como en otras redes, los nodos sensores no tienen un identificativo global.

Los protocolos de la capa de aplicación como el Protocolo de Administración de Sensores (SMP) y el Protocolo de Petición de Sensores y Entrega de Datos (SQDDP) están actualmente en investigación. El SQDDP introduce una interfaz de peticiones para emitir peticiones, responderlas y recopilar las respuestas de las peticiones.

Aunque las aplicaciones de las WSN son diversas, las últimas investigaciones indican que se van a tener que realizar más desarrollos en el direccionamiento de varios protocolos en todas las capas. además, se necesita una plataforma común donde se puedan probar los algoritmos propuestos. Las simulaciones de WSN son difíciles y normalmente están vinculadas a una aplicación en particular.

El área de las WSN es un área en expansión y en los próximos años veremos los resultados de los esfuerzos que se están realizando en estas aplicaciones.

A continuación podemos ver un esquema de los distintos niveles que podemos encontrar en cualquier red de sensores:

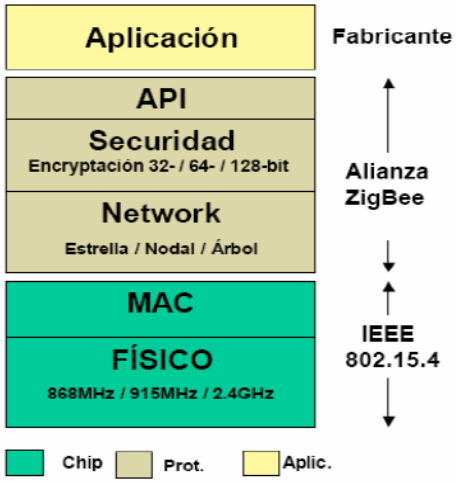


Figura 5: Niveles físico, red y aplicación

1.6. Zigbee, estándar WSN

ZigBee es un sistema ideal para redes domóticas, específicamente diseñado para reemplazar la proliferación de sensores/actuadores individuales. ZigBee fue creado para cubrir la necesidad del mercado de un sistema a bajo coste, un estándar para redes Wireless de pequeños paquetes de información, bajo consumo, seguro y fiable.

Para llevar a cabo este sistema, un grupo de trabajo formado por varias industrias (www.ZigBee.org), está desarrollando el estándar. La alianza de empresas está trabajando codo con codo con IEEE para asegurar una integración, completa y operativa. La alianza ZigBee también servirá para probar los dispositivos que se creen con esta tecnología. ZigBee sólo es el estándar basado en la tecnología necesaria para el control remoto de sensores/actuadores que se utilizan en domótica.

Figura 6: Arquitectura ZigBee

Siguiendo el estándar del modelo de referencia OSI (Open Systems Interconnection), en la figura 6, aparece la estructura de la arquitectura en capas. Las primeras dos capas, la física (PHY) y la de acceso al medio (MAC), son definidas por el estándar IEEE 802.15.4. Las capas superiores son definidas por la Alianza ZigBee. El grupo de trabajo de IEEE pasó el primer borrador de la capa física y la de acceso al medio en 2003. Una versión final de la capa de red (NWK) se acabó el año pasado, y en Junio del 2005 tenemos ya un Zigbee 1.0 publico.

Los productos ZigBee trabajan en una banda de frecuencias que incluye la 2.4 Ghz (mundial), de 902 a 928 Mhz (en Estados Unidos) y 866Mhz (en Europa). La transferencia de datos de hasta 250 Kbs puede ser transmitido en la banda de 2.4Ghz (16 canales), hasta 40kps en 915Mhz (10 canales) y a 20kps en la de 868Mhz (un solo canal). La distancia de transmisión puede variar desde los 10 metros hasta los 75, dependiendo de la potencia de transmisión y del entorno. Al igual que WiFi, ZigBee usa la DSSS (secuencia directa de espectro ensanchado) en la banda 2.4 Ghz. En las bandas de 868 y 900Mhz también se utiliza la secuencia directa de espectro ensanchado pero con modulación de fase binaria.

¿Como se estructura este estándar?

La siguiente figura nos muestra los campos de los cuatro tipos de paquetes básicos: datos, ACK, MAC y baliza.

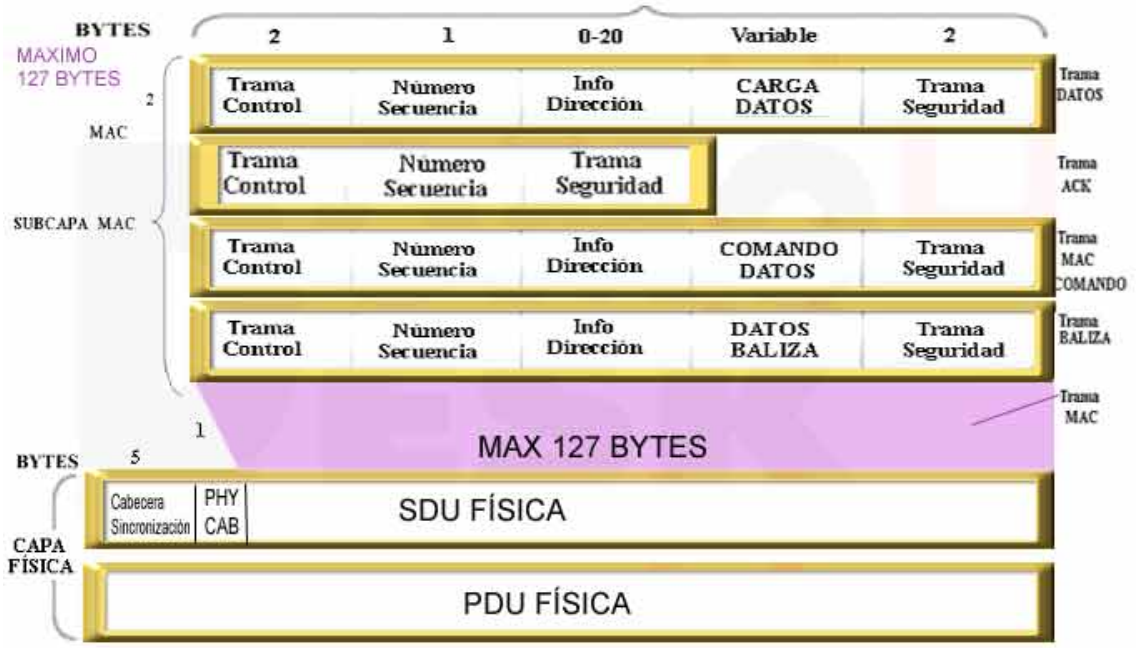


Figura 7: Tramas

El paquete de datos tiene una carga de datos de hasta 104 bytes. La trama está numerada para asegurar que todos los paquetes llegan. Un campo nos asegura que el paquete se ha recibido sin errores. Esta estructura aumenta la fiabilidad en condiciones complicadas de transmisión.

Otra estructura importante es la de ACK, o reconocimiento. Esta trama es una realimentación desde el receptor al emisor, para confirmar que el paquete se ha recibido sin errores. Se puede incluir un 'tiempo de silencio' entre tramas, para enviar un pequeño paquete después de la transmisión de cada paquete.

El paquete MAC, se utiliza para el control remoto y la configuración de dispositivos/nodos. Una red centralizada utiliza este tipo de paquetes para configurar la red a distancia.

Para acabar, el paquete baliza 'despierta' los dispositivos, que escuchan y luego vuelven a 'dormirse' si no reciben nada más. Estos paquetes son importantes para mantener todos los dispositivos y los nodos sincronizados, sin tener que gastar una gran cantidad de batería estando todo el tiempo encendidos.

Acceso al canal. Direccionamiento

Dos mecanismos de acceso al canal se implementan en 802.15.4. Para una red 'sin balizas', un estándar ALOHA CSMA-CA envía reconocimientos positivos para paquetes recibidos correctamente. En una red 'con balizas', una estructura de 'supertrama' se usa para controlar el acceso al canal. La supertrama es estudiada por el coordinador de red para transmitir 'tramas baliza' cada ciertos intervalos (múltiples cada de 15.38 msg, hasta cada 252 sg). Esta estructura garantiza el ancho de banda dedicado y bajo consumo.

Los dispositivos se direccionan empleando 64-bits y un direccionamiento corto opcional de 16 bits. El campo de dirección incluido en MAC puede contener información

de direccionamiento de ambos orígenes y destinos (necesarios para operar punto a punto). Este doble direccionamiento es usado para prevenir un fallo dentro de la red.

¿Qué tipos de dispositivos contiene?

ZigBee tiene tres tipos de dispositivos:

El **coordinador de red**, que mantiene en todo momento el control del sistema. Es el más sofisticado de los tipos de dispositivos, requiere memoria y capacidad de computación.

El **dispositivo de función completa (FFD)** capaz de recibir mensajes del estándar 802.15.4. Este puede funcionar como un coordinador de red. La memoria adicional y la capacidad de computar, lo hacen ideal para hacer las funciones de Router o para ser usado en dispositivos de red que actúen de interface con los usuarios.

El **dispositivo de función reducida (RFD)** de capacidad y funcionalidad limitadas (especificada en el estándar) para el bajo coste y simplicidad. Son los sensores/actuadores de la red.

¿Qué hace que tenga un largo periodo de vida?

El **bajo consumo de potencia** es lo que hace que la tecnología ZigBee tenga un largo periodo de vida sin tener que recargar los dispositivos. Las redes ZigBee son diseñadas para conservar la potencia en los nodos ‘esclavos’. Durante mucho tiempo, un dispositivo ‘esclavo’ está en modo ‘dormido’ y sólo de ‘despierta’ por una fracción de segundo para confirmar que está ‘vivo’ en la red de dispositivos. Por ejemplo, la transición del modo ‘dormido’ al modo ‘despierto’ (cuando transmite) dura unos 15ms y la enumeración de ‘esclavos’ dura unos 30ms.

Las redes ZigBee pueden usar el **entorno ‘con balizas’ o ‘sin balizas’**. Las balizas son usadas para sincronizar los dispositivos de la red, identificando la red domótica, y describiendo la estructura de la ‘supertrama’. Los intervalos de las balizas son determinados por el coordinador de red y pueden variar desde los 15msg hasta los 4 minutos.

El modo ‘sin balizas’ es sencillo: se usa el acceso múltiple al sistema en una red punto a punto cercano. Funciona como una red de dos caminos, donde cada dispositivo es autónomo y puede iniciar una conversación en donde los otros pueden interferir. El dispositivo destino puede no oír la petición o el canal puede estar ocupado.

El modo ‘baliza’ es un mecanismo de control del consumo de potencia en la red. Este modo permite a todos los dispositivos saber cando pueden transmitir. Aquí, los dos caminos de la red tienen un distribuidor que controla el canal y dirige las transmisiones. La principal ventaja de este método de trabajo es que se reduce el consumo de potencia.

El modo ‘sin balizas’, es típicamente usado en sistemas de seguridad, donde los dispositivos, por ejemplo, sensores, detectores de movimiento o de rotura de cristales, duermen el 99,999% del tiempo. Estos elementos ‘despiertan’ de manera regular para

anunciar que siguen en la red. Cuando un evento tiene lugar (se detecta algo), el sensor se ‘despierta’ instantáneamente y transmite la alarma. El coordinador de red, alimentado de la red principal todo el tiempo, recibe el mensaje y activa la alarma respectiva.

El modo ‘baliza’ es más recomendable cuando el coordinador de red trabaja con una batería. Los dispositivos escuchan al coordinador de red durante el ‘balizamiento’ (envío de mensajes a todos los dispositivos, broadcast, entre 0.015 y 252 segundos). Un dispositivo se registra para el coordinador y mira si hay mensajes para él. Si no hay mensajes, el dispositivo vuelve a ‘dormir’, despertando según un horario establecido por el coordinador. Una vez hecho todo el ‘balizamiento’ el coordinador mismo vuelve a ‘dormirse’.

¿Qué es lo que le convierte en un sistema seguro?

La seguridad de las transmisiones y de los datos son puntos clave en la tecnología ZigBee. ZigBee utiliza el modelo de seguridad de la subcapa MAC IEEE 802.15.4, la cual especifica 4 servicios de seguridad.

Control de accesos-el dispositivo mantiene una lista de los dispositivos ‘
Datos Encriptados, los cuales usan una encriptación con un código de 128 bits.
Integración de tramas para proteger los datos de ser modificados por otros.
Secuencias de refresco, para comprobar que las tramas no han sido reemplazadas por otras. El controlador de red comprueba estas tramas de refresco y su valor, para ver si son las esperadas.
Depende del dispositivo final que creemos será nuestra decisión el dotarlo de mas o menos seguridad.

¿Qué importancia adquiere la Capa de Red?

La capa de red (NWK) une o separa dispositivos a través del controlador de red, implementa seguridad, y encamina tramas a sus respectivos destinos. Además, la capa de red del controlador de red es responsable de crear una nueva red y asignar direcciones a los dispositivos de la misma.

La capa de red soporta múltiples configuraciones de red incluyendo estrella, árbol, y rejilla.

1.7. Problemas de las WSN

Es evidente que las tecnologías de redes de sensores nos ofrecen multitud de aplicaciones y utilidades, como veremos en el apartado siguiente, pero también conllevan un importante esfuerzo de producción eficiente tanto hardware como software.

Las características que tienen este novedoso tipo de redes requieren, como hemos visto, de protocolos y estrategias concretas a la hora de crearlas, tanto a nivel de red como a nivel de componentes individuales, ya sean los sensores con sus limitaciones de potencia o procesador, las pasarelas para enrutar de forma eficiente este tipo de redes o las

estaciones base para evaluar y ejecutar respuestas de forma adecuada ante los datos que van recibiendo.

Podemos, por tanto, describir una serie de problemas que están en estudio para conseguir posibles soluciones o mejoras, tanto a nivel hardware como de programación:

- **Optimización del consumo de energía en los nodos sensores**.

Uno de los problemas más importantes es el consumo de energía de los nodos. Para lograr que éste sea mínimo y, por tanto, conseguir un máximo tiempo de vida de la red habrá que tener en cuenta:

La comunicación de mensajes es el primer consumidor de energía.
La CPU puede quedarse en un estado sleep de bajo consumo mientras no tenga que procesar ni enviar nada.
Economizar la distancia de las comunicaciones.
Técnicas de software: programación eficiente de íneas de código.

- **Ancho de banda y cobertura de red limitados.**

Se debe trabajar teniendo en cuenta que estas redes tienen una cobertura limitada, dadas las características de sus componentes, y creando sistemas que se ajusten adecuadamente.

- **Los recursos de computación son limitados.**

Los nodos tienen ciertas limitaciones tanto a nivel del procesador como la capacidad de almacenamiento de la memoria. Dependiendo del tipo de sensor, hay diferencias, aunque gracias a las últimas novedades tecnológicas estas limitaciones se van reduciendo.

- **Soluciones ad-hoc para redes ad-hoc.**

Como hemos visto en el capítulo de protocolos, muchas de las soluciones ad-hoc no son válidas para este tipo de redes, debido a las diferencias existentes tanto en tecnología como en objetivos.

- **La topología de red es muy dinámica.**

Las WSN tienen como uno de sus objetivos crear redes móviles cuya topología va cambiando continuamente, redes caracterizadas por:
Nodos móviles.
Nodos con alta probabilidad de fallo.
Nodos que entran en el sistema.
Cuantos más nodos haya en la red mayor será el rendimiento.

1.8. Aplicaciones de las WSN

La WSN puede consistir en muchos tipos diferentes de sensores, como pueden ser sísmicos, magnéticos, térmicos, acústicos, radar, IR, etc. Los distintos tipos de sensores

existentes pueden monitorizar una gran variedad de condiciones ambientales, que incluyen:

Temperatura, humedad, presión.
Condiciones de luz, movimiento de vehículos, niveles de ruido.
Composición del suelo.
Presencia o ausencia de cierto tipo de objetos.
Niveles de estrés mecánico en objetos (maquinaria, estructuras, etc.).
Características de velocidad, dirección y tamaño de un objeto.

Los nodos sensores pueden adoptar diversas formas de trabajo, pueden actuar en modo continuo, por detección de eventos, por identificación de eventos, toma de datos localizados o como control local de actuadores (idóneo para aplicaciones domóticas).

Si tenemos el tipo de monitorización que van a realizar los sensores, podemos hacer una primera clasificación de aplicaciones, en tres tipos distintos que tendrían las siguientes propiedades:

- **Monitorización del entorno.**

Este tipo de aplicaciones se caracterizan por un gran número de nodos sincronizados que estarán midiendo y transmitiendo periódicamente en entornos puede que inaccesibles, para detectar cambios y tendencias.

La topología es estable y no se requieren datos en tiempo real, sino para análisis futuros. Ejemplos: control de agricultura, microclimas, etc.

- **Monitorización de seguridad.**

Son aplicaciones para detectar anomalías o ataques en entornos monitorizados continuamente por sensores. No están continuamente enviando datos, consumen menos y lo que importa es el estado del nodo y la latencia de la comunicación: se debe informar en tiempo real.

Como ejemplos tenemos el control de edificios inteligentes, detección de incendios, aplicaciones militares, seguridad, etc.

- **Tracking.**

Aplicaciones para controlar objetos que están etiquetados con nodos sensores en una región determinada. La topología aquíı va a ser muy dinámica debido al continuo movimiento de los nodos: se descubrirán nuevos nodos y se formaran nuevas topologías.

- **Redes híbridas**

Son aquellas en las que los escenarios de aplicación reúnen aspectos de las tres categorías anteriores.

El concepto de microsensores comunicados de forma inalámbrica promete muchas nuevas áreas de aplicación. De momento las vamos a clasificar en: militares, entorno, salud, hogar y otras áreas comerciales. Por supuesto es posible ampliar esta clasificación.

- **Aplicaciones militares**

Las WSNs pueden ser parte integral de sistemas militares C4ISRT (command, control, communications, computing, intelligence, surveillance, reconnaissance and targeting) que son los que llevan las órdenes, el control, comunicaciones, procesamiento, inteligencia, vigilancia, reconocimientos y objetivos militares.

El rápido y denso despliegue de las redes de sensores, su autoorganización y tolerancia a fallos las hace una buena solución para aplicaciones militares. Ofrecen una solución de bajo coste y fiable para éstas ya que la pérdida de un nodo no pone en riesgo el éxito de las operaciones.

Ejemplos de aplicación en esta área son:

monitorización de fuerzas aliadas, equipamiento y munición. Cada equipo, tropa, vehículo o arma crítica lleva integrado un sensor para informar de su estado a líderes o niveles superiores. Se usan nodos recolectores donde se recopila toda la información para luego transmitirla a niveles superiores.

Reconocimiento del terreno y fuerzas enemigas. Se pueden desplegar y obtener información valiosa antes de que el enemigo los intercepte sobre rutas de acceso, posibles caminos e incluso movimientos del enemigo.

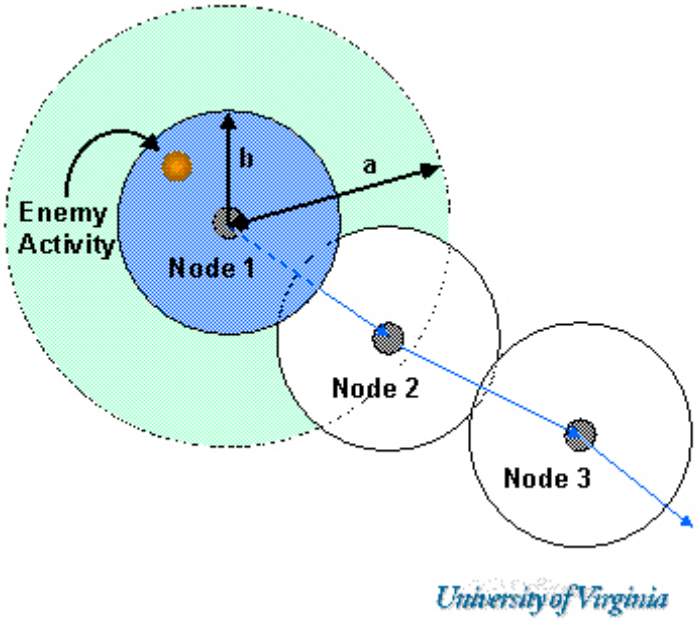


Figura 8: Reconocimiento del enemigo

Adquisición de blancos. Se pueden incorporar los nodos a sistemas de guiado de armas inteligentes.

Valoración de daños, antes o después de los ataques.

Reconocimiento de ataques nucleares, biológicos y químicos, desplegándolos en la región y usándolos como sistema de aviso. También para reconocimiento después de que

ocurra sin tener que exponer a equipos de reconocimiento a radiaciones o agentes químicos.

- **Aplicaciones medioambientales**

En este campo tenemos aplicaciones como el seguimiento de aves, animales e insectos; monitorización de condiciones ambientales que afectan al ganado y las cosechas; irrigación; macro instrumentos para la monitorización planetaria de gran escala; detección química o biológica; agricultura de precisión (monitorización de niveles de pesticidas, polución y erosión del terreno); detección de incendios; investigación meteorológica o geofísica; detección de inundaciones; mapeado de la biocomplejidad del entorno; y estudios de la polución. Podemos destacar cuatro de estas aplicaciones:

detección de fuego en bosques.

Se podrían desplegar millones de sensores de manera estratégica, aleatoria y densa en el bosque que informarán del origen exacto de un incendio antes de que se haga incontrolable. Los sensores podrían tener métodos de obtención de energía como placas solares, ya que serían abandonados durante meses o años sin mantenimiento. además, debido a la densidad de su despliegue serían capaces de cooperar para realizar una recolección de datos cooperativa y evitar obstáculos en las transmisiones, como árboles y rocas que pueden bloquear la línea de visión entre sensores.

- **Mapeado de la biocomplejidad del entorno medioambiental.**

Requiere enfoques sofisticados para integrar información en escalas temporal y espacial. Avances tecnológicos en sensorización remota y recolección de datos automatizada han permitido una resolución espacial, espectral y temporal con un coste por unidad de área que se ha reducido en progresión geométrica. además, la conexión de estos sistemas a Internet permite a usuarios remotos controlar, monitorizar y observar la biocomplejidad del medio ambiente.

Aunque los satélites y sensores aéreos son útiles en la observación a gran escala de la biodiversidad (p.e. complejidad espacial de especies de plantas dominantes), no tienen la precisión suficiente como para observar la biodiversidad a pequeña escala, que es la parte más importante en la biodiversidad de un ecosistema. Por ello, se necesita un despliegue de nodos sensores para observar la biocomplejidad.

Ejemplo: Reserva James en el sur de California con tres redes de monitorización, cada una con 25-100 sensores.

- **Detección de inundaciones.**

Ejemplo: sistema ALERT desplegado en EEUU. Hay sensores de lluvia, nivel de agua y meteorológicos. Proporcionan información a una estación base centralizada. Hay proyectos de investigación que investigan enfoques distribuidos en interacción con los nodos sensores en campo para proporcionar consultas en momentos fijos (snapshot) o en espacios temporales largos (long-running queries).

- **Aplicaciones sanitarias**

Proveer interfaces para los discapacitados; monitorización integral de pacientes; diagnósticos; administración de medicamentos en hospitales; monitorización de los movimientos y procesos internos de insectos u otros pequeños animales; telemonitorización de datos fisiológicos humanos; y seguimiento y monitorización de pacientes en un hospital.

- Telemonitorización de datos fisiológicos humanos.

Los datos recolectados se pueden almacenar durante períodos largos de tiempo, usados para exploración médica. La WSN instalada puede monitorizar y detectar el comportamiento de personas mayores, como por ejemplo una caída. Los sensores, por su reducido tamaño, dan al usuario una mayor libertad de movimiento y permiten a los médicos identificar antes síntomas predefinidos. además, permiten una mayor calidad de vida a los usuarios comparados con los centros de tratamiento. En la facultad de medicina de Grenoble - Francia, se ha diseñado una Vivienda Inteligente para la Salud para validar la viabilidad de dichos sistemas.

- Seguimiento y monitorización de médicos y pacientes.

Cada paciente lleva un sensor pequeño y ligero. Cada sensor tiene una tarea específica, por ejemplo monitorizar la presión arterial y la frecuencia cardiaca. Los médicos también llevan sensores, lo que permite a otros médicos localizarlos en el hospital.

- Administración de medicación en hospitales.

Si colocamos un sensor en la medicación se reduce el riesgo de errores, porque el paciente también llevara un sensor que identifique sus alergias y medicación requerida.

- **Aplicaciones del hogar: domótica**

Los nodos sensores pueden ser introducidos en aparatos domésticos como aspiradoras, microondas, hornos, frigoríficos y VCRs. Esto permite que sean manejados remotamente por los usuarios finales mediante una comunicación que se realizaría vía satélite o Internet.

A través de las redes de sensores pueden crear hogares inteligentes donde los nodos se integran en muebles y electrodomésticos. Los nodos dentro de una habitación se comunican entre ellos y con el servidor de la habitación. Estos servidores de habitaciones se comunican también entre ellos dando asíı conectividad entre distintas habitaciones.

Se crea lo que llamamos un entorno inteligente, cuyo diseño puede tener dos enfoques:

Desde el punto de vista humano: el entorno se adapta a necesidades del usuario final en términos de capacidades de entrada-salida.

Desde el punto de vista tecnológico: hay que desarrollar nuevas tecnologías hardware, soluciones de redes y servicios middleware.

Los sensores pueden integrarse en muebles y aparatos. Se comunican entre ellos y con servidores o actuadores de habitación, que a su vez se comunican con otros servidores de habitación. Todos ellos se integran y organizan con los dispositivos integrados existentes para autoorganizarse, autorregularse y autoadaptarse basándose en modelos de control.

1.9. Plataformas Hardware

Existen distintas plataformas hardware en el mercado, algunas de ellas son las siguientes:

1.9.1 Micaz

La plataforma de sensores MICAz está comercializada por Crossbow. Son una de las últimas generaciones de motes que trabaja en la banda de frecuencias de 2400 MHz a 2483.5 MHz.

Figura 9: MICAz

La familia MICAz usa el Chipcon CC2420, que cumple con la normativa IEEE 802.15.4 y tiene un transceptor de radio frecuencia Zigbee integrado con un micro controlador Atmega 128L. Dispone de 51 pines de I/O y una memoria flash.

En la figura 10 se muestra el diagrama de bloques:

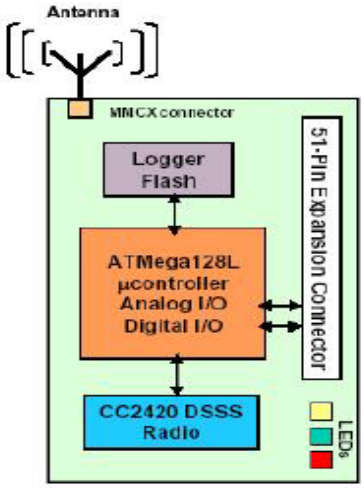


Figura 10: Diagrama de Bloques

Para la programación de estos motes se emplea la *Placa de desarrollo MIB510.* Esta placa actúa como interfaz entre el PC y los motes a través del puerto serie, permitiendo la programación de los dispositivos acoplados.

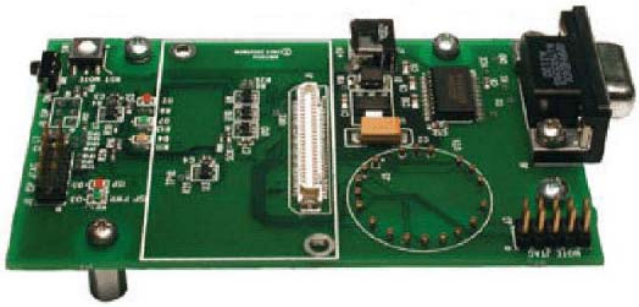


Figura 11: Placa de desarrollo MIB510

La placa MIB510 tiene un procesador (ISP) Atmega16L mediante el cual se programan los motes. El código se descarga al ISP a través del puerto serie RS-232, y es el ISP el que programa el código en el mote. Esta placa tiene conectores para motes MICAz, MICA2 y MICA2DOT.

1.9.2 Mica2

Los motes Mica2 son los módulos de tercera generación de motes que se usan para redes de sensores inalámbricas de baja potencia. Mejoran las características del Mica original:

9 Diseñado específicamente para redes de sensores integradas.
10 Distintas frecuencias de transmisión con amplio rango.
11 Más de un año de batería mediante los modos sleep.
12 Permiten reprogramación inalámbrica a distancia.
13 Amplia variedad de placas de sensores compatibles: luz, temperatura, presión, aceleración, acústica, etc.

Figura 12: Mica2

Las distintas aplicaciones en las que se utilizan estos motes son, principalmente, las WSN, la seguridad y vigilancia, la monitorización ambiental o las redes inalámbricas de gran escala.

Los Mica2 se dividen en tres modelos dependiendo de su frecuencia de uso: MPR400 (915MHz), MPR410 (433MHz), y MPR420 (315MHz). Estos motes usan el Chipcon CC1000, con radio modulada en FSK. Todos los modelos usan un potente microcontrolador Atmega128L y una radio de frecuencia sintonizable en un rango amplio.

El diagrama de bloques del mote Mica2 se muestra en la figura 13:

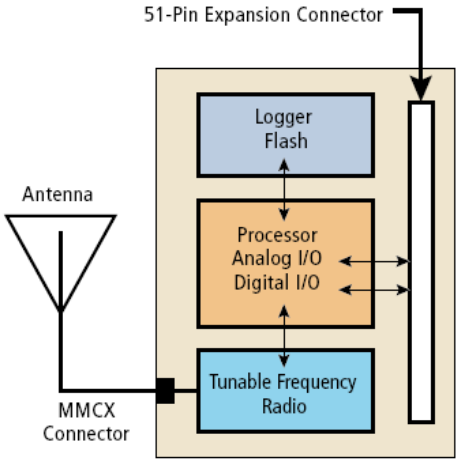


Figura 13: Diagrama de Bloques.

1.9.3 Mica2dot

Los Mica2Dot son un tipo de motes diseñados especialmente para aplicaciones donde el tamaño físico es fundamental. Al igual que los Mica2 hay tres modelos dependiendo de su frecuencia: MPR500 (915MHz), MPR510 (433MHz), y MPR520 (315MHz). El resto de características son similares a las de los Mica2, lo más relevante es la forma física y el reducido tamaño que poseen, tal y como podemos ver en la figura 14.

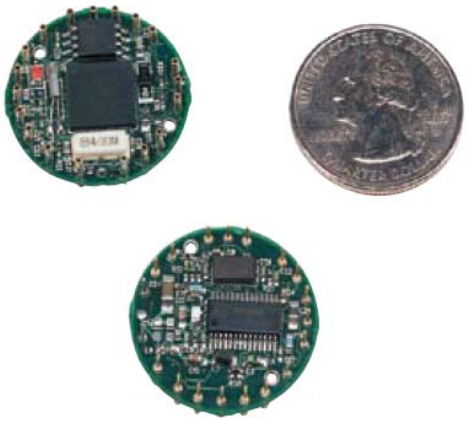


Figura 14: Mica2dot

Está basado en el microcontrolador Atmel ATmega128L de baja potencia y opera con TinyOS desde su memoria flash interna. Tiene 18 pines de expansión para conectar 6 entradas analógicas, I/O digital y una interfaz de comunicaciones serie o UART.

Características:

- Transmisión multicanal 868 / 916 MHz, 433 MHz o 315 MHz de rango extendido.
- Permite programación remota inalámbrica.
- Compatible con el mote Mica2.
- Dispone de un sensor de temperatura, monitor de la batería y LED.

El diagrama de bloques de este dispositivo se muestra en la figura 15:

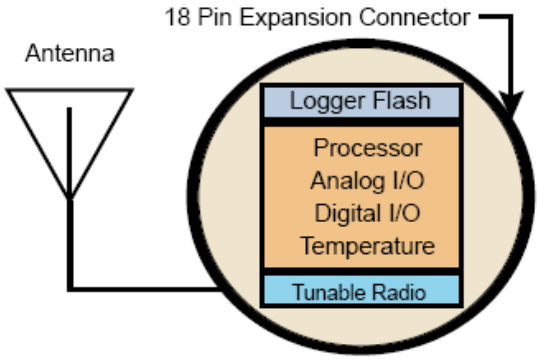


Figura 15: Diagrama de Bloques

1.9.4 Telos

Los motes TelosB (TPR2400) reúne todo lo esencial para estudios de laboratorio en una plataforma simple, incluyendo la capacidad de programación por USB, una antena integrada con sistema radio IEEE 802.15.4, un procesador de bajo consumo con una memoria extendida y un conjunto de sensores, concretamente en el modelo TPR2400.

Figura 16: TelosB

Las características generales de los TelosB son:

- Transmisión RF de acuerdo con la norma IEEE 802.15.4/ZigBee.
- Banda de frecuencias desde 2.4 a 2.4835 GHz.
- Velocidad de transferencia de datos de 250 kbps.
- Antena integrada.
- Micro-controlador MSP430 a 8MHz con 10kB de RAM.
- Bajo consumo.
- Memoria flash externa de 1 Mb para almacenamiento de datos.
- Programación y toma de datos vía USB.
- Conjunto de sensores de luz, temperatura y humedad.
- Soporta TinyOS para implementación y comunicación de redes.

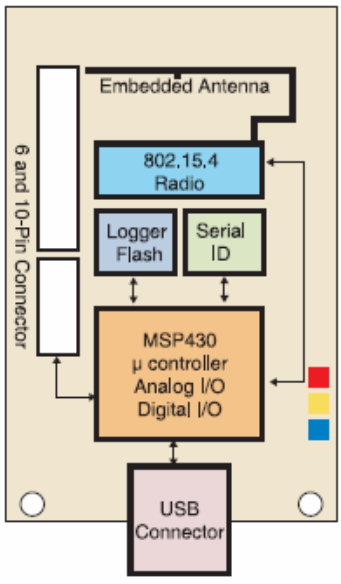


Figura 17: Diagrama de Bloques

Esta plataforma consigue un bajo consumo de potencia permitiendo una larga vida a las baterías además de tener un tiempo mínimo en el estado de wakeup, otro de los objetivos dentro de las estrategias de bajo consumo.

Los TelosB se alimentan de 2 baterías AA, aunque si es conectado mediante el puerto USB para programación o comunicación, la alimentación la proporciona el ordenador.

También se proporciona la capacidad de añadir dispositivos adicionales. Los dos conectores de expansión de los que dispone pueden ser configurados para controlar sensores analógicos, periféricos digitales y displays LCD.

Con todas estas características, el mote TelosB no sólo proporciona más facilidad para programación, más flexibilidad y más presentación, sino que es el que menos consumo de potencia ofrece permitiendo alargar la vida de los nodos considerablemente y siendo ésta su principal baza frente a los otros dispositivos. Por todo ello, serán los dispositivos que utilizaremos en este proyecto.

1.10 Resumen comparativo

Las diferentes plataformas de motes que hemos visto se diferencian en sus características hardware lo que les proporciona distintas cualidades a unas de otras: mayor procesador, frecuencias de transmisión, velocidad de transmisión de datos, consumo de energía, etc.

En la siguiente tabla podemos ver la evolución que han sufrido los distintos prototipos y, lo que es más importante para nosotros, la comparación con las prestaciones que tienen y ofrecen los TelosB.

Mote Type	*WeC*	*René*	*René 2*	*Dot*	*Mica*	*Mica2Dot*	*Mica 2*	*Telos*
Year	1998	1999	2000	2000	2001	2002	2002	2004
Microcontroller								
Type	AT90LS8535		ATmega163		ATmega128			TI MSP430
Program memory (KB)	8		16		128			60
RAM (KB)	0.5		1		4			2
Active Power (mW)	15		15		8		33	3
Sleep Power (μW)	45		45		75		75	6
Wakeup Time (μs)	1000		36		180		180	6
Nonvolatile storage								
Chip	24LC256				AT45DB041B			ST M24M01S
Connection type	I^2C				SPI			I^2C
Size (KB)	32				512			128
Communication								
Radio	TR1000				TR1000	CC1000		CC2420
Data rate (kbps)	10				40	38.4		250
Modulation type	OOK				ASK	FSK		O-QPSK
Receive Power (mW)	9				12	29		38
Transmit Power at 0dBm (mW)	36				36	42		35
Power Consumption								
Minimum Operation (V)	2.7		2.7		2.7			1.8
Total Active Power (mW)	24				27	44	89	41
Programming and Sensor Interface								
Expansion	none	51-pin	51-pin	none	51-pin	19-pin	51-pin	10-pin
Communication	IEEE 1284 (programming) and RS232 (requires additional hardware)							USB
Integrated Sensors	no	no	no	yes	no	no	no	yes

Figura 18: Evolución de los motes

En la figura siguiente, observamos más concretamente los tiempos de cambio de estado, asíı como la potencia consumida en cada estado, obteniendo siempre los mínimos resultados para los Telos, y consiguiendo un mayor tiempo de vida del nodo.

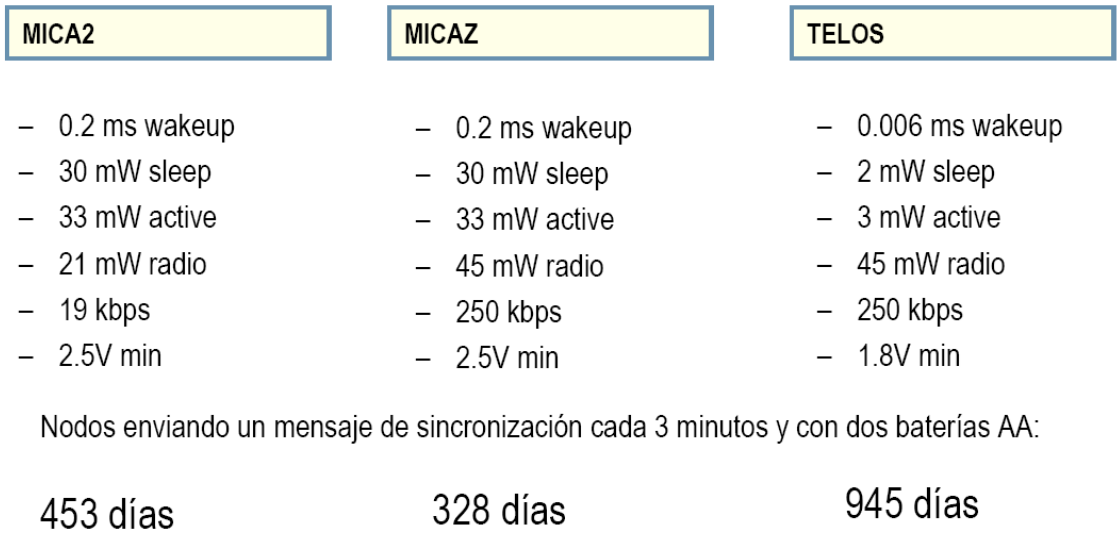


Figura 19: Comparativa de tiempos y consumos

TINYOS

2.1.Introducción

TinyOS en un sistema operativo para trabajar con redes de sensores, desarrollado en la Universidad de Berkeley. TinyOS puede ser visto como una colección de components software y herramientas avanzadas, el cual cuenta con un amplio uso por parte de comunidades de desarrollo, dada sus características de ser un proyecto de código abierto (Open Source). Este "conjunto de programas" contiene numerosos algoritmos, que nos permitirán desde generar enrutamientos, como también aplicaciones pre-construidas para sensores. Además soporta diferentes plataformas de nodos de sensores, arquitecturas bases para el desarrollo de aplicaciones.

Como se verá más adelante, el lenguaje en el que se encuentra programado TinyOS es orientado a componentes que deriva de C, cuyo nombre es NesC. Además existen variadas herramientas que ayudan el estudio y desarrollo de aplicaciones para las redes de sensores, que van desde aplicaciones para la obtención y manejo de datos, hasta sistemas completos de simulación. Sin duda esta respuesta puede ampliarse bastante, y es objetivo de este documento presentar todas las características que TinyOS posee, y por ende lo definen.

Aunque puede trabajar en otras plataformas, las plataformas apoyadas son Linux (RedHat 9), Windows 2000, y Windows XP. Es importante mencionar que el proyecto se inicio sobre BSD (Unix), por lo cual para el funcionamiento en Windows se hace necesario la instalación de *Cygwin,* un simulador de plataformas unix, siendo necesario tener básicos conocimientos acerca de este entorno. Además muchas de las herramientas disponibles se encuentran desarrolladas para plataformas JAVA, por tanto también es necesaria su instalación. La instalación de éstas en sistema Windows se realiza de forma transparente, puesto que existe un paquete ejecutable, el cual además realiza toda la configuración del sistema.

La programación de dispositivos se puede realizar a través de distintos puertos, dependiendo del módulo con el que se cuente. Estos pueden ser serial, paralelo o ethernet. Es entonces necesario contar con al menos uno de estos puertos en el PC.

2.2. Introducción a TinyOS

El diseño de TinyOS está basado en responder a las características y necesidades de las redes de sensores, tales como reducido tamaño de memoria, bajo consumo de energía, operaciones de concurrencia intensiva, diversidad en diseños y usos, y finalmente operaciones robustas para facilitar el desarrollo confiable de aplicaciones. Además se encuentra optimizado en términos de uso de memoria y eficiencia de energía.

El diseño del Kernel de TinyOS está basado en una estructura de dos niveles de planificación:

-Eventos:

Pensados para realizar un proceso pequeño (por ejemplo cuando el contador del timer se interrumpe, o atender las interrupciones de un conversor análogo-digital). Además pueden interrumpir las tareas que se están ejecutando.

-**Tareas:**

Las tareas son pensadas para hacer una cantidad mayor de procesamiento y no son críticas en tiempo (por ejemplo calcular el promedio en un arreglo). Las tareas se ejecutan en su totalidad, pero la solicitud de iniciar una tarea, y el término de ella son funciones separadas. Esta característica es propia de la programación orientada a componentes, la cual se presentará en detalle en la sección siguiente.

Con este diseño se permite que los eventos (que son rápidamente ejecutables), puedan ser realizados inmediatamente, pudiendo interrumpir a las tareas (que tienen mayor complejidad en comparación a los eventos).

El enfoque basado en eventos es la solución ideal para alcanzar un alto rendimiento en aplicaciones de concurrencia intensiva. Adicionalmente, este enfoque usa las capacidades de la CPU de manera eficiente y de esta forma gasta el mínimo de energía.

TinyOS se encuentra programado en NesC, un lenguaje diseñado para reflejar las ideas propias del enfoque de componentes, incorporando además un modelo de programación que soporta concurrencia, manejo de comunicaciones y fácil interacción con el medio (manejo de hardware). TinyOS y NesC se encuentran profundamente relacionados, es por eso que a continuación presentaremos un pequeño resumen con algunas de las características de este lenguaje.

2.3. NesC

NesC es un lenguaje de programación basado en C, orientado a sistemas embebidos que incorporan el manejo de red. Además soporta un modelo de programación que integra el manejo de comunicaciones, las concurrencias que provocan las tareas y eventos y la capacidad de reaccionar frente a sucesos que puedan ocurrir en los ambientes donde se desempeña (por ejemplo, sensar).

También realiza optimizaciones en la compilación del programa, detectando posibles carreras de datos que pueden ocurrir producto de modificaciones concurrentes a un mismo estado, dentro del proceso de ejecución de la aplicación. Además simplifica el desarrollo de aplicaciones, reduce el tamaño del código, y elimina muchas fuentes potenciales de errores.

Básicamente NesC ofrece:

Separación entre la construcción y la composición. Hay dos tipos de componentes en NesC: módulos y configuraciones. Los módulos proveen el código de la aplicación, implementando una o más interfaces. Estas interfaces son los únicos puntos de acceso a la componente. Las configuraciones son usadas para unir los componentes entre sí,

conectando las interfaces que algunas componentes proveen con las interfaces que otras usan.

- Interfaces bidireccionales: tal como ya se explicaba anteriormente las interfaces son los accesos a los componentes, conteniendo comandos y eventos, los cuales son los que implementan las funciones. El proveedor de una interfaz implementa los comandos, mientras que el que las utiliza implementa eventos.

Unión estática de componentes, vía sus interfaces. Esto aumenta la eficiencia en tiempos de ejecución, incrementa la robustez del diseño, y permite un mejor análisis del programa.

Finalmente y a modo de resumen, mencionamos los principales aspectos que el modelo de programación NesC ofrece y que deben ser entendidos para el entendimiento del diseño con TinyOS:

- Tal como se mencionó el modelo de NesC está formado por interfaces y componentes.
- Una interfaz puede ser usada o puede ser provista, los componentes son módulos o configuraciones.
- Una aplicación se verá representada como un conjunto de componentes, agrupados y relacionados entre sí, tal como se observa en la figura 21.

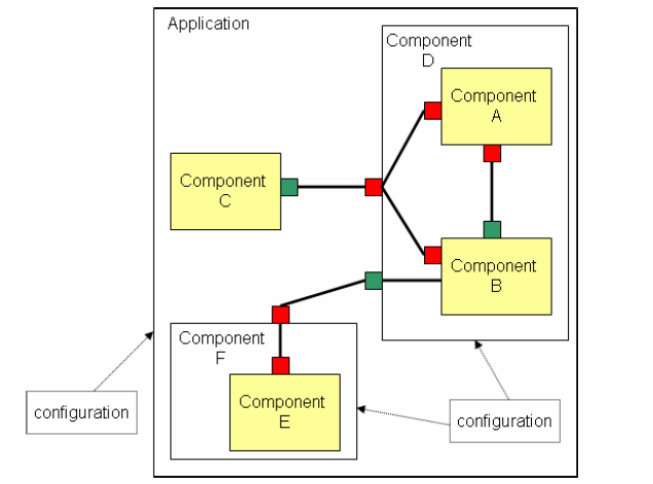


Figura 20. Aplicación**.**

- Las interfaces se utilizan para operaciones que describen la interacción bidireccional; el proveedor de la interfaz debe implementar comandos, mientras que el usuario de la interfaz debe implementar eventos.
-
- Existen dos tipos de componentes:
-
    - Módulos, que implementan especificaciones de una componente.
    - Configuraciones, que se encargarán de unir (*wire*) diferentes componentes en función de sus interfaces, ya sean comandos o eventos.

En la figura 21 se muestra un diagrama de bloques en el que se describe el proceso de compilación para una aplicación TinyOS.

Ahora bien, comprendido estos conceptos, es posible dar el paso siguiente hacia el entendimiento del diseño en TinyOS.

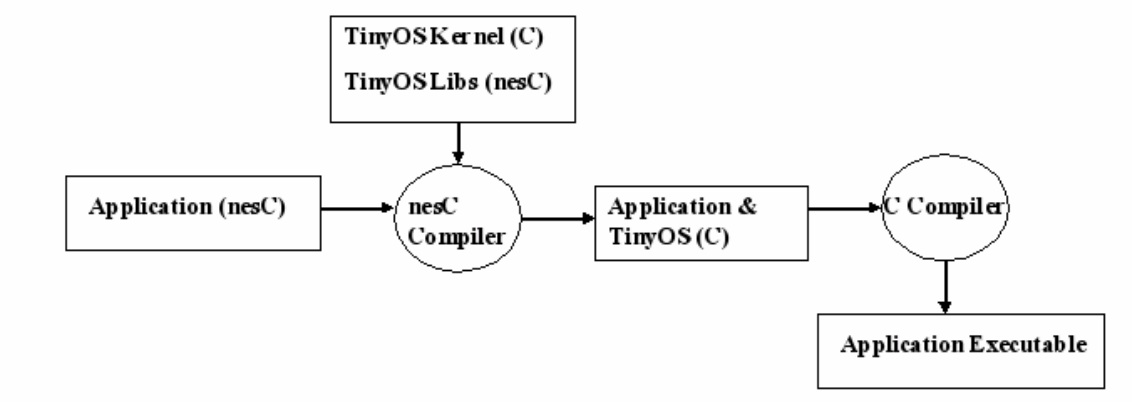


Figura 21. Compilación de una aplicación TinyOS**.**

2.4 TinyOS*.*

TinyOS posee un modelo de programación característico de los sistemas embebidos: el modelo de componentes. Tal como se ha mencionado TinyOS esta escrito en NesC, lenguaje que surge para facilitar el desarrollo de este tipo de aplicaciones. A continuación se presentan las ideas de este modelo.

-Arquitectura basada en componentes:

TinyOS se encuentra construido sobre un conjunto de componentes de sistema, las cuales proveen la base para la creación de aplicaciones. En la figura 23 se muestra el modelo de una componente. Conectar estas componentes, usando un conjunto de especificaciones detalladas, es lo que finalmente definirá una aplicación. Este modelo es esencial en sistemas embebidos para incrementar la confiabilidad, sin sacrificar desempeño.

-Concurrencia en tareas y en eventos:

Las dos fuentes de concurrencia en TinyOS son las tareas y los eventos. Los componentes entregan tareas al planificador, siendo el retorno de éste de forma inmediata, aplazando el cómputo hasta que el planificador ejecute la tarea. Los componentes pueden realizar tareas siempre y cuando los requerimientos de tiempo no sean críticos. Para asegurar que el tiempo de espera no sea muy largo, es que se recomienda programar tareas cortas, y en caso de necesitar procesamientos mayores, se recomienda dividirlo en múltiples tareas. Las tareas se ejecutan en su totalidad, y no tiene prioridad sobre otras tareas o eventos. Así tambien los eventos hasta completarse,

pero estos sí pueden interrumpir otros eventos o tareas, con el objetivo de cumplir de la mejor forma los requerimientos de tiempo real.

Todas las operaciones de larga duración deben ser divididas en dos estados: la solicitud de la operación y la ejecución de ésta. Específicamente si un comando solicita la ejecución de una operación, éste debiese retornar inmediatamente mientras que la ejecución queda en mano del planificador, el cual deberá señalizar a través de un evento, el éxito de la operación.

Una ventaja secundaria de elegir este modelo de programación, es que propaga las abstracciones del hardware en el software. Tal como el hardware responde a cambios de estado en sus pines de entrada/salida, nuestros componentes responden a eventos y a los comandos en sus interfaces.

Ahora se tiene claro que las aplicaciones TinyOS son construidas por componentes. Un componente provee y usa interfaces. Estas interfaces son el único punto de acceso a la componente. Además un componente estará compuesta de un espacio de memoria y un conjunto de tareas.

A continuación se detallan los cuatro elementos que conforman una componente:

1. Manejador de comandos.
2. Manejador de eventos
3. Un *frame* de tamaño fijo y estáticamente asignado, en el cual se representa el estado interno de la componente.
4. Un bloque con tareas simples.

En la figura 22 se muestra el modelo gráfico de una componente, en la cual se observan los elementos mencionados anteriormente.

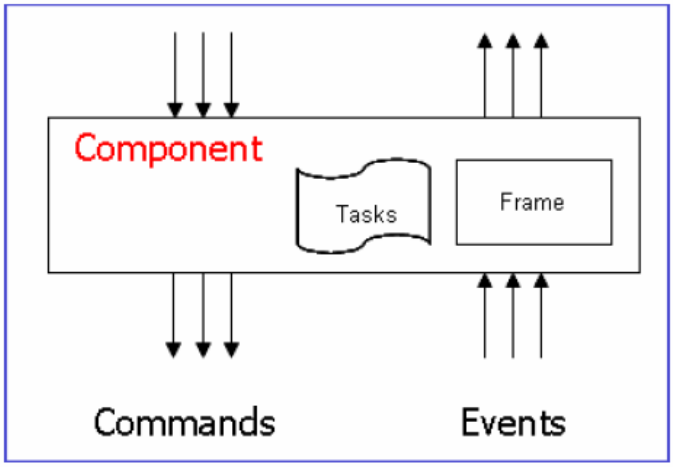


Figura 22. Modelo de componentes de TinyOS y su interacción

Los comandos son peticiones hechas a componentes de capas inferiores. Estos generalmente son solicitados para ejecutar alguna operación. Existen dos posibles casos de uso de los comandos. El primero es para operaciones bifase, donde los comandos

retornan inmediatamente, no generando bloqueos por la espera de la ejecución, es decir, una vez que realizan la petición, el planificador será el encargado de ejecutar lo solicitado, generando un evento que indique el fin de la operación. El otro es el caso de realizar una operación no bifase, donde ésta se realizará completamente, y por tanto no habrá evento retornado.

Los manejadores de eventos son invocados por eventos de componentes de capas inferiores, o por interrupciones cuando se está directamente conectado al hardware. Similar a los comandos, el *frame* será modificado y las tareas agregadas. Los eventos son capaces de interrumpir tareas, no así viceversa.

Las tareas se ejecutan hasta terminar y pueden ser sólo interrumpidas por eventos (podríamos decir que eventos tienen mayor prioridad). Las tareas son entregadas a un planificador (*task scheduler*) que en este caso está implementado con método FIFO. Debido a esta implementación, las tareas se ejecutan secuencialmente y no deben ser excesivamente largas. Alternativamente al planificador de tareas FIFO, puede ser reemplazado por planificadores basados en prioridades (priority-based) ó por planificadores basados en plazos (*deadline-based*), los cuales pueden ser implementados sobre TinyOS.

2.4.1Tipos de Componentes.

En general los componentes se clasifican en una de estas categorías:

- **Abstracciones de Hardware:**

Mapean el hardware físico en el modelo de componentes de TinyOS. Por ejemplo, en el módulo que se observa en la figura 24, la componente de radio RFM, es representativa de esta clase, ya que exporta comandos para manipular los pines de entrada-salida conectados al RFM y entrega eventos, informando a otros componentes acerca del bit de transmisión o recepción. Su *frame* contiene información acerca del estado actual de la componente (si se encuentra transmitiendo o recibiendo, la tasa de transferencia de bits, etc).

El RFM detecta las interrupciones del hardware que se transforman en el bit de evento de RX o en el bit TX, dependiendo del modo de operación.

- **Hardware Sintético:**

Los componentes de hardware sintéticos simulan el comportamiento del hardware avanzado. Un buen ejemplo de tal componente es el Radio Byte (véase el figura 24). Intercambia datos de entrada o salida del módulo RFM y señaliza cuando un byte ha sido completado. Las tareas internas realizan la codificación y decodificación de los datos. Conceptualmente, esta componente es una máquina de estado que podría ser directamente modelada en el hardware. Desde el punto de vista de los niveles superiores, esta componente provee una interfaz, funcionalmente muy similar a la componente de abstracción de hardware UART: proporcionan los mismos comandos y señalan los mismos eventos, se ocupan de datos del mismo tamaño, e internamente

realizan tareas similares (buscando un bit o un símbolo de inicio, realizando una codificación simple, etc.).

- **Componentes de alto nivel:**

Realizan el control, enrutamientos y toda la transferencia de datos. Un representante de esta clase es el módulo de mensajes ("*messaging module*"), presentado en la figura 24. Éste realiza la función de llenar el buffer de los paquetes antes de la transmisión y envía mensajes recibidos a su lugar apropiado. Además, los componentes que realizan cálculos sobre los datos o su agregación, entran en esta categoría.

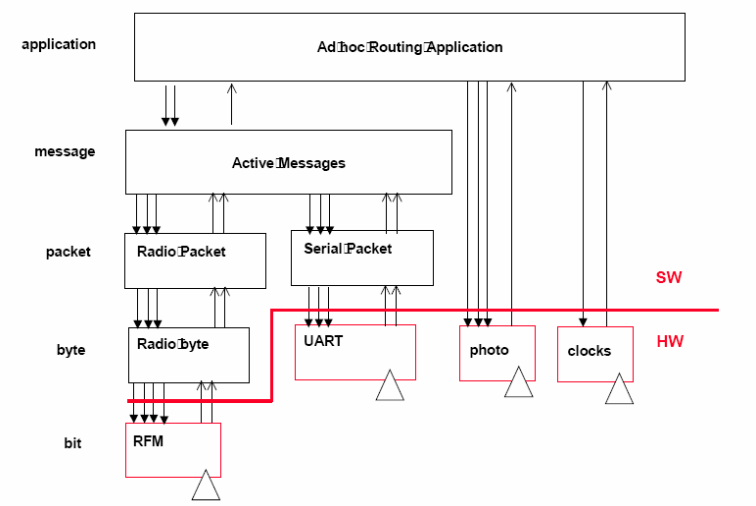


Figura 23. Grafo de componentes para un sistema MultiHop**.**

Otra de las características de TinyOS es que el modelo de componentes permite la fácil migración a otro hardware, dada la abstracción de hardware que se logra con el modelo de manejo de eventos. Además, la migración es particularmente importante en las redes de sensores, ya que constantemente aparecen nuevas tecnologías, donde de seguro los diseñadores de sistemas querrán explorar el compromiso entre la integración a nivel físico, los requerimientos de energía, y el costo del sistema, requerimientos claves para la optimización.

2.4.2. Ejemplo de un componente

Una componente típica que incluye un frame, eventos, comandos y tareas es mostrada en la figura 24.

Gráficamente, el componente se presenta como un conjunto de tareas, un bloque de estado (*frame* del componente), un conjunto de comandos (triángulos al revés), un conjunto de manejadores (triángulos), flechas para abajo para los comandos que utiliza, y flechas para arriba (prepicadas) para los comandos que señala.

En la figura 26 se muestra el código de este componente. Como se puede observar el módulo principal se divide en dos partes: una que contiene las interfaces provistas y la otra que contiene las que utiliza. Es deber del programador hacer coincidir los formatos de eventos y comandos entre las distintas interfaces. De todas formas errores son verificados en tiempo de compilación.

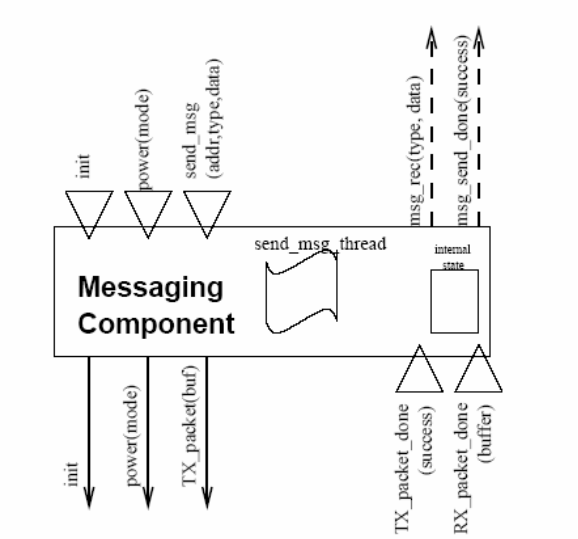


Figura 24. Representación gráfica de la componente “Active Messages” (AM).

```
module AMStandard
{
  provides {
    interface StdControl as Control;
    interface CommControl;

    // The interface are as parameterised by the active message id
    interface SendMsg[uint8_t id];
    interface ReceiveMsg[uint8_t id];

    // How many packets were received in the past second
    command uint16_t activity();
  }

  uses {
    // signaled after every send completion for components which wish to
    // retry failed sends
    event result_t sendDone();

    interface StdControl as UARTControl;
    interface BareSendMsg as UARTSend;
    interface ReceiveMsg as UARTReceive;

    interface StdControl as RadioControl;
    interface BareSendMsg as RadioSend;
    interface ReceiveMsg as RadioReceive;
    interface Leds;
    //interface Timer as ActivityTimer;
  }
}
```

Figura 25. Archivo que describe la componente**.**

2.4.3 Implementación de un componente

En TinyOS los componentes se unen en tiempo de compilación, considerando las especificaciones dadas en los componentes de configuración, es decir, se unen respetando las relaciones declaradas por las interfaces. Para facilitar la composición, cada interfaz está descrita en el inicio de cada archivo de componentes. Éste entrega una lista de los comandos que acepta y los eventos que manipula, como también el conjunto de eventos que señaliza y los comandos que usa. Se puede visualizar el modelo como una caja negra, donde solo se ven entradas y salidas. La completa descripción de estas interfaces superiores e inferiores, son usadas en el tiempo de compilación para generar automáticamente archivos de cabecera.

Todo esto es manejado por el compilador de lenguaje NesC. Un claro ejemplo en que el compilador debe verificar múltiples uniones, es que si un evento simple debe ser manejado por varios componentes, en tiempo de compilación el código será automáticamente generado para enviar el evento a tantos lugares como sea necesario.

2.4.4 Ejemplo de una aplicación completa

En la siguiente figura se muestra una vista simplificada de una aplicación implementada con TinyOS. Cada nodo representa una componente, y las flechas representan las uniones de las interfaces. Cada flecha es rotulada con el correspondiente nombre de la interfaz.

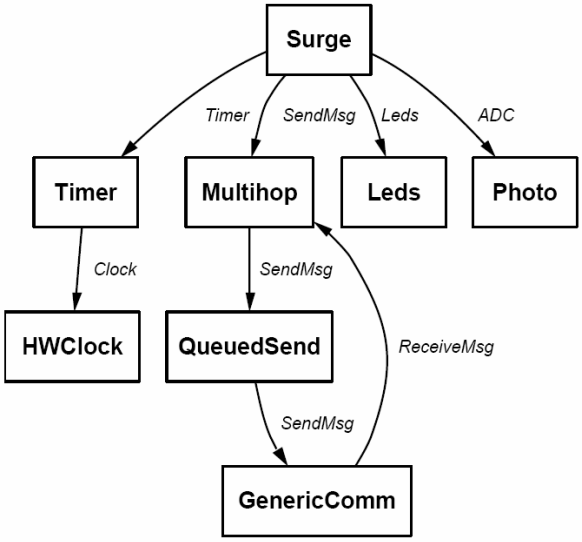


Figura 26: Aplicación Surge.

La aplicación se llama Surge, y tiene como función sensar periódicamente una variable, que en este caso es luminosidad, para luego ser enviada a través de la red inalámbrica. El objetivo de esta aplicación es entregar el valor medido a la base. Para eso cada nodo es capaz de identificar a un nodo padre, el cual le permitirá llegar a la estación base.

La aplicación solo debe incorporar las partes que necesita de TinyOS, como son el código de inicio de sistema (MAIN), el temer (Timer), un sensor (Photo), acceso a los leds (Leds), y enrutamiento de mensajes Multihop (Multihop). Además explícitamente se especifican las dependencias de ambiente en función de las interfaces. La aplicación requiere un timer (interfaz Timer), un sensor (interfaz ADC), leds (interfaz Leds) y finalmente comunicación (interfaz Send).

2.4.5 Modelo de Comunicación.

Una pregunta a resolver es ¿cómo se manejarán los mensajes en la red? Sin duda implementar socket en protocolo TCP/IP no es una buena idea, dado que se necesitaría excesiva memoria para almacenar los paquetes que van llegando además de los bloqueos que se generan mientras no se lea el mensaje. También es necesario evitar el excesivo tráfico sobre la red, por lo cual mensajes de recibo, secuencias de retransmisión, etc. son inaceptables.

La solución propuesta es trabajar con la metodología de “Active Messages”. Para su funcionamiento cada mensaje debe contener el nombre de un manejador de eventos. Para esto el que envía debe declarar un buffer en el frame, para así colocar el nombre del manejador, luego solicitar el envío y esperar que llegue la respuesta de realizado. El receptor entonces se encarga de invocar el correspondiente manejador de eventos.

De esta forma no se generan bloqueos o esperas en el receptor, el comportamiento es similar a que ocurriese un evento, y el almacenamiento es simple.

2.5. Herramientas de TinyOS

2.5.1 Listen

Esta utilidad tiene la misión de mostrar por consola todos los paquetes que se reciban por un determinado puerto TCP. Es bastante útil, cuando se quiere comprobar que es lo que va a recibir una aplicación del *pc.*

Esta utilidad se encuentra en el directorio /tools/java/net/tinyos/tools/.

2.5.2 SerialForwarder

Esta utilidad tiene la misión de ligar un puerto de comunicaciones como el *com, tossim-radio, tossim-uart* con un puerto TCP del ordenador, de manera que sirve de pasarela entre los dos, es una aplicación muy útil que permite realizar una aplicación de *pc* que permita enviar por TCP datos, y estos datos serán recibidos por el *SerialForwarder* y reenviados al puerto que se le haya especificado y viceversa es decir todos los paquete que sean enviados por el sensor al puerto que esta escuchando el *SerialForwarder* serán retransmitidos con todas las conexiones activas que haya en el puerto TCP en el que se encuentra escuchando.

Por detecto, posee el puerto *COM1* y el puerto *TCP 9001* pero se pueden cambiar en cualquier momento.

Esta aplicación se encuentra en /tools/java/net/tinyos/sf/ y su invocación se hace mediante la maquina virtual de java, como nota especial decir que para su correcta invocación, se ha de invocar a la aplicación desde el directorio /tools/java ya que los otros directorios existen por que son un *mapping* del sistema de paquetes que se hautilizado en la programación de la utilidad.

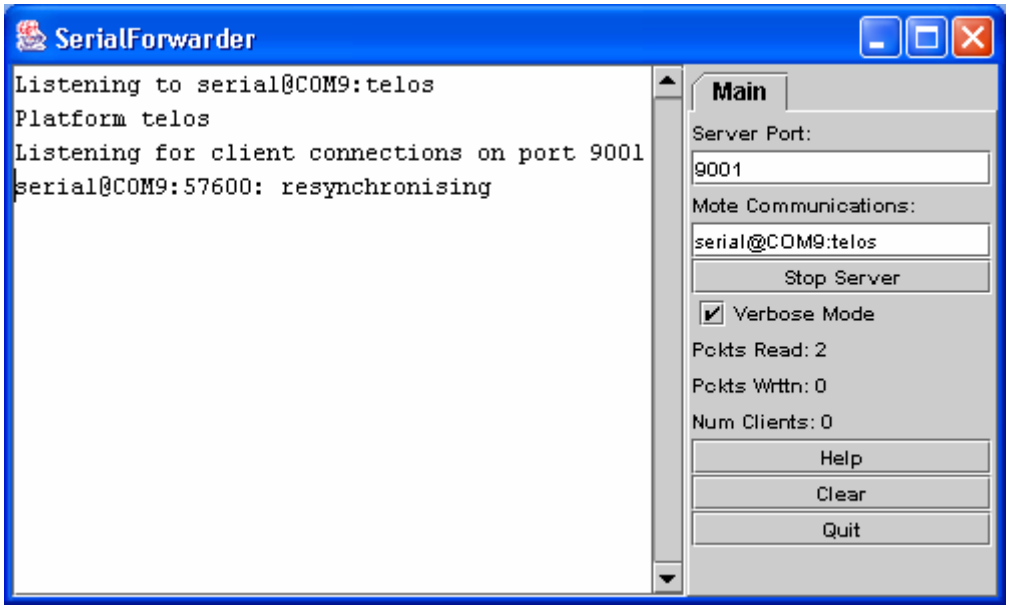


Figura 27. SerialForwarder.

2.5.3 NesDoc

Esta utilidad permite generar documentación automáticamente a partir del código fuente de un programa. Por cada fichero fuente genera un fichero HTML con un gráfico que describe el conexionado de los componentes del fichero a través de sus interfaces y una descripción textual sacada de los comentarios de los ficheros fuente.

Para generar esta documentación, escribe make <platform> docs desde el directorio de la aplicación deseada. El documento resultante se localiza en docs/nesdoc/<platform>.docs/nesdoc/<platform>/index.html que es el índice principal de todas las aplicaciones documentadas

2.5.4 Motelist

Esta herramienta proporciona el puerto COM para la comunicación entre el PC y el mote conectado a él. Con el mote conectado al ordenador, el comando que se debe elecutar es ‘motelist’.

2.5.5 MIG (Message Interface Generator)

MIG es una herramienta utilizada para generar automáticamente las clases de Java correspondientes a los tipos de mensaje activos que utiliza la aplicación del mote. MIG lee en nesC las definiciones struct para los tipos de mensajes en la aplicación del mote y genera una clase Java por cada tipo de mensaje cuidando los detalles del empaquetado y desempaquetan los campos en el formato del byte del mensaje.

MIG se usa conjuntamente con el paquete net.tinyos.message, que proporciona un número de rutinas para enviar y recibir mensajes a través de las clases MIG generadas. *NCG (nesC Constant Generator)* es una herramienta para extraer constantes de archivos nesC para el uso con otras aplicaciones y es típicamente usado en conjunto con MIG.

Cada campo en una clase MIG generada tiene al menos 8 métodos asociados con ella:

- isSigned_*fieldname*: – indica si este campo es o no una cantidad firmada.
- isArray_*fieldname* – indica si este campo es o no un array.
- get_*fieldname* – devuelve el valor de este campo.
- set_*fieldname* – establece el valor de este campo.
- offset_*fieldname* – devuelve el offset (en bytes) de este campo.
- offsetBits_*fieldname* - devuelve el offset (en bits) de este campo.
- size_*fieldname* – devuelve la longitud (en bytes) de este campo.
- sizeBits_*fieldname* - devuelve la longitud (en bits) de este campo.

El compilador nesC incluye una utilidad para la generación automática de documentación a partir de los archivos que contienen el código fuente. La estructura del archivo de documentación se basa en los distintos archivos que componen el código fuente de la aplicación, en las interfaces usadas, y en la plataforma usada (mica, mica2, mica2dot) y el formato de la documentación es html. Nos da información de cada uno de los componentes que se utilizan, las interfaces que ofrecen y las que hacen uso de ellas, un esquema grafico de cómo están relacionados los componentes entre ellos.

Para la generación de la documentación solo tenemos que escribir en el directorio de la aplicación el comando:

“make <plataforma> docs” y el directorio donde se almacena es C:\tinyos\cygwin\opt\tinyos-1.x\doc\nesdoc\mica2 donde mic2 es el nombre de la plataforma.

2.5.6 TOSSIM / TinyViz

Es un simulador de eventos discretos para TinyOS. TOSSIM (TinyOS SIMulator) se compila directamente desde componentes TinyOS a la plataforma destino

especificada. Gracias a esto se pueden introducir sentencias en el código generado que informen del estado de la simulación. Por ejemplo se puede saber cuando la simulación transmite un paquete, enciende un led, provoca una interrupción del reloj, etc. Permite una simulación bastante completa de una red. Entre sus posibilidades se encuentra la simulación de las transmisiones de datos a nivel de bit fijando una probabilidad de error de bit. También puede tomar lecturas de los sensores, los datos obtenidos de ellos son en principio aleatorios, aunque existe la posibilidad de que sean introducidos por el usuario de TOSSIM.

TinyViz es la interfaz gráfica de TOSSIM. Permite ver la topología de la red aunque la calidad de la imagen que ofrece no es buena. Soporta la adición de plug-ins con nueva funcionalidad.

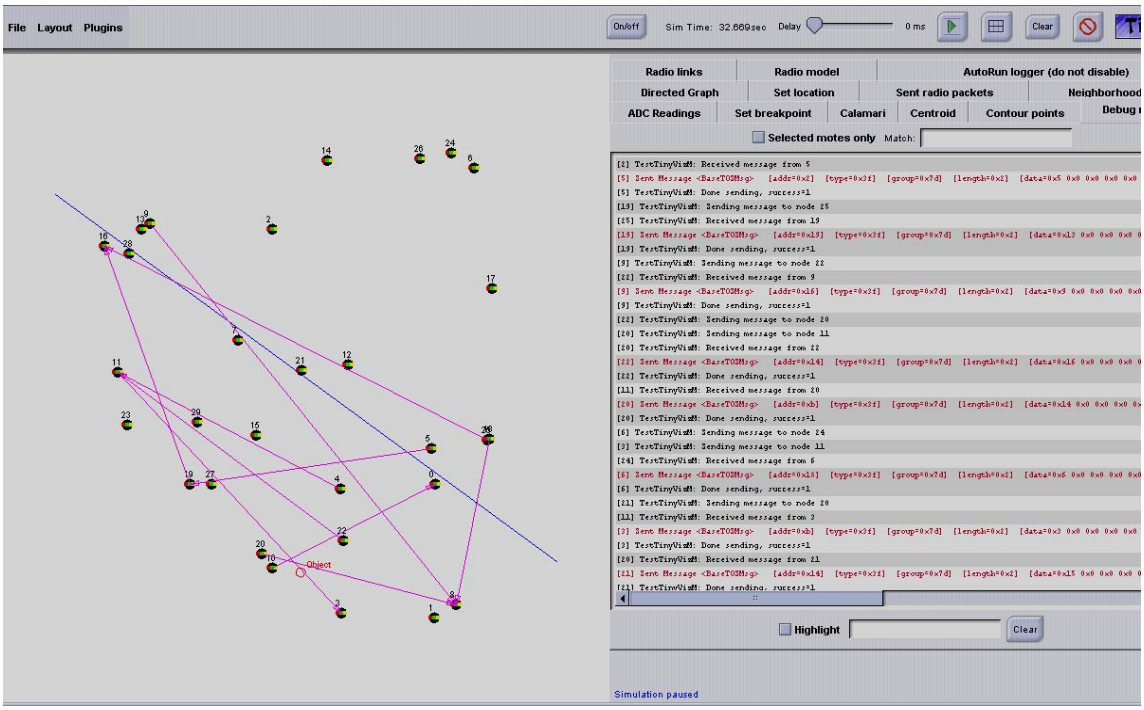


Figura 28. TinyViz.

2.5.7 TinyDB

TinyDB es un sistema de procesado de consultas para extraer información de una red de sensores con TinyOS. Convierte la red en una tabla de una base de datos distribuida, donde existe una columna por cada tipo de dato que se pretende leer (temperatura, luz, etc.). Las consultas se realizan en el lenguaje TinySQL, una extensión del lenguaje de consultas SQL. Las extensiones a SQL permiten que al pedir una lectura se puedan especificar la frecuencia de muestreo, el periodo de tiempo durante el que se tomarán muestras e incluso TinyDB puede ajustar estos anteriores parámetros para lograr conseguir un determinado tiempo de vida de los nodos. Para retransmitir consultas y respuestas los nodos se organizan en una estructura de árbol que permite reducir el número de mensajes generados. Cuando un nodo recibe una consulta, éste estará mandado datos hacia el nodo sumidero durante el tiempo y según los criterios que la consulta especifique.

TinyDB está implementado como un framework extensible. A través de esta extensibilidad se pretende que, en un futuro, se añadan nuevos eventos y atributos de los nodos y se pueda invocar comandos para la realizaciónde tareas de control y actuación

2.5.8 Surge View

Surge View es una aplicación java que viene incluida con TinyOS. Permite monitorizar una red y analizar el funcionamiento del mallado de la red. Sus características incluyen descubrimiento y configuración automática de la red; visionado de la topología de red; almacenamiento y visualización de estadísticas de la red como rendimiento, calidad de los enlaces, etc.; y una herramienta gráfica para el visionado de datos.

2.5.9 Oscope:

Es una interfaz gráfica implementada en Java, la cual nos pemite la representación gráfica de los lecturas enviadas por distintos sensores, pudiendo tener la representación gráfica de una misma magnitud desde varios sensores de forma simultanea.

Permite almacenar los datos representados para un posterior procesado de ellos. Permite tambien realizar tareas de escalado, zoom y mover los ejes para obtener la más óptima representación gráfica de los datos y crear un archivo donde almacena los posible erores que se puedan dar durante su funcionamiento.

Figura 29. Osciloscopio.

En este proyecto se ha implementado esta herramienta, pero con una serie de modificaciones que se comentan a continuación:

Sólo se puede representar las lecturas que recibe unicamente desde un sensor, han sido modificadas la escalas del eje X y eje Y para que se pueda mostrar mejor aún las gráficas. Se ha quitado botones como Contrl Panel, Load Data, ya que no son necesarios en la aplicación.Tambien se ha añadido la utilidad de poder mostrar el tipo de comando que se esta ejecutando en el nodo, el cual está enviando las lectuas para ser representadas gráficamente.

2.6 Deluge 2.0

Deluge es un sistema de reprogramación de los nodos a través de la red que permite programar una nueva imagen del programa a ejecutarse en el nodo de forma inalámbrica propagando dicha imagen sobre la red inalámbrica y permitiendo que cada nodo se programe con la nueva imagen.

Las características que proporciona son las siguientes:

- **Soporta multihop:** permite programar inalámbricamente todos los nodos en una red multihop.

- **Propagación epidémica:** el resultado continuo de la propagación por todos los nodos ayuda a asegurar el alcance de todos los nodos sin conectividad intermitente.

- **Almacena múltiples imágenes de programa**: con cada nodo almacenando múltiples imágenes de programa, rápidamente se puede intercambiar diferentes programas en la red sin cargarlos continuamente.

- **Opera en redes heterogéneas**: arranca diferentes códigos de aplicaciones en diferentes nodos.

- **Golden image**: en la memoria flash externa se almacena una imagen de programa con el mínimo soporte para la programación de redes. Siempre tiene una parte de código que permite recuperación.

- **Bootloader aislado:** El bootloader es el responsable de programar el microcontrolador y recuperar los errores de programación al cargar la Golden Image.

- **Recuperación ante fallos de programación**: Tras la carga incompleta de los componentes básicos de Deluge, permite volver a un estado donde los motes puedan aceptar la carga de un nuevo programa.

- **Comprobación redundante de la integridad de datos**: asegura que las imágenes de programa en todos los nodos no son corruptas.

- **Pequeño consumo RAM**: Menos de 150 bytes.

El principal objetivo de esta versión es aumentar la robustez y utilidad. Actualmente Deluge 2.0 es soportada por las siguientes plataformas: Mica2, Mica2-dot, MicaZ, Telos rev. A, Telos rev. B, Tmote Sky, y TOSSIM.

Esta versión introduce algunas mejoras:

- Verificar la imagen antes de programar.
- Comprobar el voltaje del sistema antes de programar.
- Protección de escritura hardware.
- Información prolija.
- Recuperación ante fallos de programación.
- Estructuras de datos protegidas.
- Autodetección de TOSBase.
- Auto - resume de imágenes incompletas.
- Autodetección de imágenes idénticas.
- Protección de error de operador.
- Reset images.
- Descarga de imágenes Deluge de cualquier nodo.

2.6.1 Conjunto de Herramientas java para Deluge:

Las herramientas java para Deluge proporcionan la capacidad de realizar un ping para comprobar el estado de un nodo sensor, inyectar una nueva imagen, realizar un reinicio de una imagen cargada en un nodo sensor y extraer la imagen almacenada en un nodo sensor. Para ello el único requisito necesario es tener configurado la herramienta SerialForwarder correctamente con el puerto COM que se está utilizando bien con una conexión serie, o través de un TOSBase.

- **Ping:**

El comando Ping es útil para comprobar el estado de la red. Un ping devuelve información sobre la imagen actual que se está ejecutando y la posición de cada una de las imágenes almacenadas. Para cada imagen devuelve la siguiente información:

- Program Name: nombre de la aplicación principal.
- Compile Time: hora exacta de compilación de la aplicación.
- Platform: la plataforma TinyOS para la cual fue compilada.
- User ID: el usuario UNIX que compilo la aplicación.
- Hostname: el host que compilo la aplicación.
- User Hash: el hash del usuario y el hostname.
- Num Pages: Tamaño de la imagen en unidades de páginas de Deluge.

El comando a ejecutar para realizar un ping es: java.net.tinyos.tools.Deluge –p

- **Inyectar Imagen (Inject):**

Este comando se utiliza para inyectar una imagen, especificando el archivo tosimage y el número de imagen. Toda la información sobre las versiones permanecen en la red y no en el PC desde donde es ejecuta el comando.
El comando a ejecutar para inyectar una imagen es:
java net.tinyos.tools.Deluge –in –to=tos_image.xml –imgnun=num.

- **Reiniciar una imagen (Reboot):**

Este comando se usa para reiniciar una imagen cargada en un sensor, la cual se especifica con el número de imagen a reiniciar. Este proceso esta señalizado por los leds. Como ventaja decir que el comando Reboot y el comando Inject son independientes, lo que significa que hasta que no se complete el proceso de inyectar una imagen, no se puede reiniciar.

El comando a ejecutar para realizar un reinicio es:
java net.tinyo.tools.Deluge ---reboot –imgnum=número imagen.

- **Borrar una Imagen(Erase):**

Este comando se usa si se quiere eliminar una imagen de un nodo y evitar así su propagación por la red. El comando es: java net.tinyos.tools.Deluge –erase –imgnum=numero imagen.

- **Resetear una Imagen(Reset):**

Con este comando conseguimos resetear toda la información de una imagen cargada en un sensor, con ello evitamos que una imagen se extienda por la red. Pero solo afecta al nodo conectado directamente al PC.

El comando es: java net.tinyos.tools.Deluge ---reset -imgnum=numero imagen.

- **Extraer una imagen de un sensor(Dump):**

Este comando se utiliza para descargar una imagen que se encuentra en un sensor de la red. El comando es: java net.tinyos.tools.Deluge –dump –imgnum=numero imagen –outfile=fichero destino.

2.6.2 Detalles de la programación de red:

- **Tosboot:**

Ofrece un conjunto de mecanismos necesarios para reprogramar un sensor con una imagen de programa almacenada en el sensor. Los parámetros son pasados a TosBoot a través de una memoria no volátil, de esta manera asegura la correcta la correcta operación, incluso cuando ocurren inesperadas interrupciones, durante el proceso de arranque, Tosboot inicia una secuencia de parpadeos de los leds si la plataforma lo soporta. Cuando termina si hay petición de reprogramar, TosBoot borra la memoria flash y escribe el archivo binario, una vez finalizado el bit de programación es reseteado y TosBoot salta a la primera instrucción de la aplicación. Si no hay orden de programación, salta a la aplicación ejecutada.

TOSBoot hace una serie de cosas para ayudar a proteger el nodo de una amplia gama de fracasos durante la programación:

1. TOSBoot comprueba la tensión de alimentación para asegurarse de que está en un nivel seguro antes de programar el nodo. Muchos microcontroladores pueden operar en un nivel de tensión de alimentación que es inferior a la que se requiere para escribir en la memoria flash.
2. TOSBoot utiliza información CRC almacenada por el Deluge para verificar la imagen. Antes de la programación, TOSBoot verificar todos los CRCs. Si los CRCs no coinciden, TOSBoot cancelará el proceso de programación y continuar el arranque el nodo de la forma habitual.
3. TOSBoot chequea la dirección de cada escritura para evitar sobrescribir.
4. SI ocurren errores durante la programación del nodo, TOSBoot lo intentará varias veces otra vez, y si también fallan, TOSBoot intentará programar el sensor con la aplicación Golden Imagen.

- **Golden Image:**

Es una aplicación TinyOS completa y comprobada cuya función es la de que las plataformas puedan soporta la protección contra escritura en la memoria flash para la aplicación Golden Image. La intención es tener un mecanismo para poder volver a tener el nodo a un estado recuperable, de este modo Golden Image ofrece la mínima funcionalidad para soportar Deluge y así poder reprogramarse de forma inalámbrica.

2.7 TinySchema

TinySchema es una colección de componentes TinyOS que gestiona un pequeño grupo de nombres de atributos, comandos y eventos que pueden ser fácilmente consultados, señalizados e invocados desde dentro o fuera de una red de motes.

2.7.1 Atributos, Comandos y Eventos:

Un atributo TinySchema es muy similar a una columna en un sistema de bases de datos tradicionales. Tiene un nombre y un tipo. Además, TinySchema le permite asociar código TinyOS arbitrario a cada atributo para obtener y establecer el valor del atributo. Una vez que un atributo es creado, éste puede ser recuperado o actualizado a través de una interfaz unificada facilitada por TinySchema. También se puede construir su propia aplicación para manipular los atributos basados en las interfaces proporcionados por TinySchema. Típicamente, hay tres clases de atributos:

- **Atributos del sensor:**

Éstos pueden ser ráfagas de lecturas de los sensores tales como la temperatura y sensores a la luz, acelerómetros, magnetómetros, etc.… También pueden ser computados los valores de los sensores después de aplicar alguna calibración o de un procesamiento lógico de señales.

- **Atributos introspectivos:**

Estos son los valores de estados internos de software o hardware, por ejemplo, la versión de software de ellos, matrices de enrutamiento del nodo en árbol, voltaje de la batería, etc. Son muy útiles para la vigilancia de la salud y las estadísticas de un mote de red.

- **Atributos Constantes:**

Estos son valores constantes asignado a un mote en tiempo de programación o en tiempo de ejecución, por ejemplo, el nodo id, identificador de grupo, nombre, ubicación, etc.

Un comando TinySchema es muy similar a un procedimiento almacenado en un sistema de bases de datos tradicionales. Se compone de un nombre, una lista de argumentos y un tipo de retorno. Se puede asociar código TinyOS arbitrario a cada comando. TinySchema proporciona una interfaz unificada para invocar estos comandos. Normalmente, hay dos clases de comandos:

- **Comandos Actuadores:**

Son los que realizan alguna acción física en un mote, por ejemplo, reiniciar un mote, flash LED, sonido zumbador, etc.

- **Comandos de Sintonizado:**

Son los comandos que ajustan los parámetros internos, por ejemplo, la política de enrutamiento, el número de retransmisiones, frecuencia de muestreo, etc.

Los eventos de TinySchema se introducen para capturar eventos asíncronos a las redes de sensores, por ejemplo, pulsar un botón, etc.

TinySchema ofrece interfaces para el registro y la invocación de eventos, así como asociar comandos TinySchema con eventos como callbacks cuando los acontecimientos se señalan.

Actualmente todos los atributos y todos los comandos deben ser estáticamente construidos en cada mote. Como nota, destacar que se esta estudiando el poder integrar con las máquinas virtuales que se están desarrollando para la TinyOS como Mate y Mottle, para permitir la creación dinámica de los atributos y comandos.

2.7.2 Componentes Attr, Command, Event, Tipos y estructuras definidas en TinySchema:

- **Attr.NC:**

Este componente ofrece 4 interfaces:

1. La interfaz StdControl la cual se usa para inicializar los atributos.
2. la interfaz AttrRegister para crear Atributos de cualquier tipo, la cual esta parametrizada para instanciarla cuando se crea un atributo.

3. La interfaz AttrRegisterConst es similar a la anterior, pero solo difiere en que se usa para crear atributos de valores constantes.
4. La interfaz AttUse la cual se usa para poder acceder o modificar los valores de los atributos (comandos get y set respectivamente).

- **Command.NC:**

Este componente al igual que el anterior ofrece las siguientes interfaces:

1. La interfaz StdControl la cual se usa para inicializar los comandos.
2. La interfaz CommandRegister para crear comandos de cualquier tipo, la cual está parametrizada para instanciarla cuando se crea una atributo.
3. La interfaz CommandUse se usa para poder acceder a todos los comandos, realizar invocaciones tanto locales como en otros motes, etc..

- **Event.NC**:

1. La interfaz StdControl la cual se usa para inicializar los eventos.
2. La interfaz EventRegister para crear eventos.
3. La interfaz EventUse para usar y señalizar eventos desde rellamadas de comandos.

Una vez se ha definido los 3 componentes que implementan TinySchema, se va a describir los tipos y estructuras definidas en los archivos cabecera, los cuales se usan para poder agrupar los atributos, comandos y eventos en una sola estructura y poder acceder a ellos de forma simultánea:

TinySchema define un conjunto de tipos de datos y de errores los cuales están definidos en el archivo SchemaType.h. A continuación se enumeran los tipos de datos y de errores:

- VOID: Tipo void.. Definido para comandos que no devuelven ningún valor.
- INT8 and UINT8: Tipo entero de 8 bits con signo y sin signo respectivamente.
- INT16 and UINT16: Tipo entero de 16 bits con signo y sin signo respectivamente.
- INT32 and UINT32: Igual que los anteriores pero de 32 bits.
- TIMESTAMP: no soportado aun.
- STRING: String ASCII con character nulo.
- COMPLEX_TYPE: no soportado aun.

Los códigos de error usados en todas las interfaces de TinySchema son:

- SCHEMA_SUCCESS: método finalizado correctamente (success).
- SCHEMA_ERROR: hay algún error.
- SCHEMA_RESULT_READY: el resultado devuelto ya está almacenado en su lugar de destino. Se usa para no-split-phase atributos y comandos.
- SCHEMA_RESULT_NULL: el valor devuelto es nulo.
- SCHEMA_RESULT_PENDING: el valor devuelto no está almacenado aun en su lugar correspondiente. Hay que esperar al evento que señaliza que el valor devuelto ya esta en su lugar correspondiente. Se usa para split-phase atributos y comandos.

Comenzando por las estructuras y siguiendo el mismo orden que en el apartado anterior, se iniciará comentando las estructuras utilizadas por el componente Attr.NC:

```
typedef struct {
TOSType type;
uint8_t nbytes;
uint8_t idx; //index into AttrDesc array
uint8_t id; // id for AttrRegister interface dispatch
int8_t constIdx;  // index for constant values
char *name;
} AttrDesc;
```

Figura 30. Estructura AttrDesC.

En la estructura anterior se almacena un atributo con todos sus campos, nombre, valor, id, type (TosType), numero de bytes. Cuando se tiene varios atributos se usa la siguiente estructura:

```
typedef struct {
  uint8_t numAttrs;
  AttrDesc attrDesc[MAX_ATTRS];
} AttrDescs;
```

Figura 31. Estructura AttrDescs.

Con la estructura anterior se utiliza el array attrDesc en la que se puede encontrar todos los atributos creados, y con el campo numAttrs devuelve el número de atributos que existen.

Para la utilización del componente Command y sus métodos, se usa al igual que con el componente Attr ,otras dos estructuras, las cuales son:

```
typedef struct {
      uint8_t idx; // index into
CommandDesc array
      char *name;
      uint8_t id; // id for
CommandRegister interface dispatch
      TOSType retType;
      uint8_t retLen;
      ParamList params;
} CommandDesc;
```

Figura 32. Estructura CommandDesc.

Esta estructura contiene toda la información necesaria de un comando: nombre, id, tipo, longitud, lista de parámetros. Cuando existen más comandos usamos la siguiente estructura:

```
typedef struct {
      uint8_t numCmds;
      CommandDesc commandDesc[MAX_COMMANDS];
} CommandDescs;
```

Figura 33. Estructura CommandDescs.

Al igual que en el caso de los atributos, la estructura anterior contiene a todos los comandos creados.

Las estructuras utilizadas por el componente Event cumplen la misma función que las anteriores y son estructuras muy similares:

```
typedef struct {
      uint8_t idx; // index into EventDesc
array
      char name[MAX_EVENT_NAME_LEN + 1];
      uint8_t cmds[MAX_CMD_PER_EVENT];
      uint8_t numCmds;
      bool deleted;
      ParamList params;
} EventDesc;
```

Figura 34. Estructura EventDesc.

```
typedef struct {
uint8_t numEvents;
EventDesc eventDesc[MAX_EVENTS];
} EventDescs;

typedef struct {
EventDescPtr      eventDesc;
ParamVals         *eventParams;
} EventInstance;

typedef struct {
EventInstance     events[MAX_EVENT_QUEUE_LEN];
bool              inuse;
short             head;
short             tail;
short             size;
} EventQueue;

typedef EventDescs *EventDescsPtr;
```

Figura 35. Estructuras EvenDescs y EventDescsPtr.

Sólo queda por explicar cada uno de los comandos que usa cada componente, al igual que otras estructuras como son: ParmaList, ParamVal, etc..
Para ello, la mejor forma de explicarlos es mostrando ejemplos de cómo crear atributos, comandos y eventos.

2.7.3 Ejemplos Prácticos de Atributos y Comandos :

- **Atributos :**

La creación de atributos sigue siempre el mismo patrón, el cual se describe a continuación:
El primer paso consiste en registrar los atributos que se quieren crear, utilizando el comando attrRegister de la interfaz AttrRegister, a dicho comando se le pasan como argumentos el nombre del atributo, tipo de valor que devuelve, etc..

Una vez registrado el atributo sólo tenemos que implementar los eventos: starAttr(),getAttr() y setAttr().El evento starAttr() será señalizado cuando se quiera iniciar el atributo creado, y los otros dos, serán señalizados cuando se quiera realizar una lectura o modificación del atributo respectivamente.

A continuación se muestra el código correspondiente a la creación de 2 atributos que son nodeid y group, los cuales acceden a las variables TOS_LOCAL_ADDRESS y TOS_AM_GROUP.

```
includes AM;

// component to expose certain global variables as attributes
module AttrGlobalM
{
      provides interface StdControl;
      uses
      {
            interface AttrRegister as NodeIdAttr;
            interface AttrRegister as GroupAttr;
      }
}
implementation
{
      command result_t StdControl.init()
      {
            if (call NodeIdAttr.registerAttr("nodeid", UINT16, 2) != SUCCESS)
                  return FAIL;
            if (call GroupAttr.registerAttr("group", UINT8, 1) != SUCCESS)
                  return FAIL;
            return SUCCESS;   //registramos los atributos nodeid y group
      }

      command result_t StdControl.start()
      {
            return SUCCESS;
      }

      command result_t StdControl.stop()
      {
            return SUCCESS;
      }

      event result_t NodeIdAttr.startAttr()
      {
            return call NodeIdAttr.startAttrDone();
      }

      event result_t NodeIdAttr.getAttr(char *name, char *resultBuf, SchemaErrorNo
*errorNo)
      {
            *errorNo = SCHEMA_RESULT_READY;
            *(uint16_t*)resultBuf = TOS_LOCAL_ADDRESS;
            return SUCCESS;
      }
                  //Implementamos el método get para el atributos nodeid.
      event result_t NodeIdAttr.setAttr(char *name, char *resultBuf)
      {
            TOS_LOCAL_ADDRESS = *(uint16_t*)resultBuf;
            return SUCCESS;
      }//Modificamos su dirección
```

Figura 36. Módulo AttrGlobalM.NC.

```
event result_t GroupAttr.startAttr()
      {
            return call GroupAttr.startAttrDone();
      }
                  //cuando se llama al start() del atributo group
      event result_t GroupAttr.getAttr(char *name, char *resultBuf,
SchemaErrorNo *errorNo)
      {           //para acceder al atributo Group
            *errorNo = SCHEMA_RESULT_READY;
            *(uint8_t*)resultBuf = TOS_AM_GROUP;
            return SUCCESS;
      }
      event result_t GroupAttr.setAttr(char *name, char *resultBuf)
      {
            TOS_AM_GROUP = *(uint8_t*)resultBuf;
            return SUCCESS;
      }           //modificamos el atributo Group
}
```

Figura 37. Continuacion Módulo AttrGlobalM.NC

```
configuration AttrGlobal
{
      provides interface StdControl;
}
implementation
{
      components Attr, AttrGlobalM;

      StdControl = AttrGlobalM;
      AttrGlobalM.NodeIdAttr -> Attr.Attr[unique("Attr")];
      AttrGlobalM.GroupAttr -> Attr.Attr[unique("Attr")];
}
```

Figura 38. Configuración AttrGlobal.

En el archivo de configuración se puede ver como sólo se utiliza el componente Attr, el cual está parametrizado para crear los dos atributos (nodeid y group). Importante resaltar que siempre deben ofrecer la interfaz Stdcontrol implementada en su Módulo, para poder así activarlos y acceder a ellos.

En los atributos Split-phase lo único que cambia es que se tiene que esperar a que el evento getAttrDone sea señalizado para poder tener el resultado cuando usamos el comando getAttr.

- **Comandos:**

De forma análoga a la creación de los atributos, los comandos se crean siguiendo los mismos pasos, pero utilizando los comandos e implementando los eventos de las interfaces correspondientes.

Primero se registra el comando(nombre del comando, atributos que se le pasan al comando, etc.) . Seguidamente se implementa el evento comandFunc(), que es donde reside la funcionalidad del comando, y como ultimo, puesto que normalmente se hace uso de la interfaz AttUse, se implementan todos sus eventos.

En el ejemplo que se muestra a continuación, se puede ver como se implementa un comando, los pasos a seguir (registrar comando, implementar sus comandos y sus eventos), y el archivo de configuración.(conexión de interfaces, componentes utilizados , etc.).

```
module CommandPotM
{
      provides interface StdControl;
      uses
      {
            interface CommandRegister as SetPot;
            interface Pot;
            interface Leds;
      }
}
implementation
{
      command result_t StdControl.init()
      {
            ParamList paramList;
            setParamList(&paramList, 1, UINT8);
            if (call SetPot.registerCommand("SetPot", VOID, 0,
&paramList) != SUCCESS)
                  return FAIL;
            return SUCCESS;
      }

      command result_t StdControl.start()
      {
            return SUCCESS;
      }

      command result_t StdControl.stop()
      {
            return SUCCESS;
      }

      event result_t SetPot.commandFunc(char *commandName, char
*resultBuf, SchemaErrorNo *errorNo, ParamVals *params)
      {
            uint8_t arg;
            *errorNo = SCHEMA_RESULT_READY;
            call Leds.redToggle();
            arg = *(uint8_t*)(params->paramDataPtr[0]);
            return call Pot.set(arg);
      }}
```

Figura 39. Módulo CommandPotM.NC

```
configuration CommandPot
{
      provides interface StdControl;
      provides interface Pot as NewPot;
}
implementation
{
      components Command, CommandPotM, PotC, LedsC;

      StdControl = CommandPotM;
      NewPot = PotC;
      CommandPotM.SetPot -> Command.Cmd[unique("Command")];
      CommandPotM.Pot -> PotC.Pot;
      CommandPotM.Leds -> LedsC.Leds;
}
```

Figura 40. Configuración CommandPot.NC.

2.8 Componente utilizados en la creación de los Atributos y Comandos:

- **2.8.1 Atributos:**

En la aplicación Telos en NesC, en el archivo de configuración(Telos.NC) se puede encontrar todos los componentes que se utilizan, los cuales unos estan unidos al componente Main, y el resto al componente TelosM.NC, como se puede apreciar en la siguiente figura:

Antes de empezar a describir los Atributos y los componentes usados, es muy importante explicar el concepto de split-phase.

Debido a que TinyOS no tiene bloqueo en las operaciones, todas las operaciones que son de larga duración, se dividen en dos partes: solicitud y terminacion. Normalmente los comandos son solicitudes de una ejecución de una operacion, si la operación es split-phase, el comando devuelve inmediatamente y la fase de terminación será señalizada con un evento, si la operación es no-split-phase no tiene evento de terminación. Para explicar mejor este concepto, se va explicar un par de ejemplos**:** un ejemplo seria encender un Leds. Si por otro lado la operación fuera enviar un paquete con send(), un componente puede invocar enviar el comando send() para inicializar la transmision de un mensaje de radio, y el componente de comunicación señaliza el evento sendDone() cuando la transmision se ha completado.

Este sencillo modelo de concurrencia permite alta concurrencia con pocos gastos de recursos de memoria, en contraste con un hilo basado en el modelo de concurrencia que consume muchos hilos de memoria. Se espera que en un futuro los motes tengan la suficiente memoria para poder soportar el modelo de concurrencia multi-thread.

Se puede hacer una evidente clasificación en dos grupos, los que son del tipo split-phase y los que no son split-phase:

- **Atributos Split-Phase:**
  - **HamamatsuC:**

Se comenzará explicando el componente HamamatsuC.NC, el cual se utiliza para crear los atributos “SolarTelos” e “InfraTelos”:

Este componente que se encuentra en el directorio /tinyos-1.x/tos/platform/telos ofrece varias interfaces ADC,ADCSingle y ADCMultiple para poder obtener lecturas simples y multiples de los sensores de luz solar y luz infraroja que se encuentran en el sensor TelosB.

En cada uno de los atributos creados, lo que se debe hacer es conectar la interfaz ADC ofrecida por el componente HamamatsuC con el módulo de del atributo(HamaTelosM.NC), según sea el atributo conectaremos la intefaz TSR o PAR,

y además se debe conectar el módulo TelosM con la interfaz stdControl del componente HamamatsuC. Con estas conexiones ya se puede obtener lecturas procedentes de los sensores TSR y PAR.

```
implementation{

components Attr,HamamatsuC,HamaTelosM,LedsC;
StdControl=HamaTelosM.StdControl;
HamaTelosM.HamaControl->HamamatsuC.StdControl;
HamaTelosM.Luzsolar->Attr.Attr[unique("Attr")];
HamaTelosM.Luzif->Attr.Attr[unique("Attr")];
HamaTelosM.TSR->HamamatsuC.TSR;
HamaTelosM.PAR->HamamatsuC.PAR;
HamaTelosM.Leds->LedsC;
}
```

Figura 41. Configuración Atributo HamaTelos.

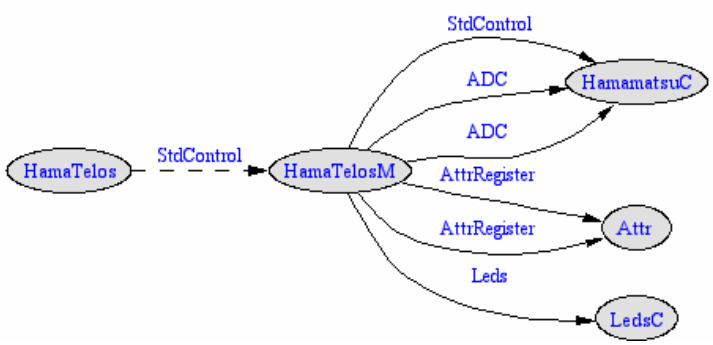


Figura 42. Componentes de la aplicación HamaTelos.

Comentar que las lecturas lo que se obtiene es un valor devuelto por el conversor ADC, y ese valor primero hay que pasarlo a voltaje y despues hacer una conversión a luxes, y cada conversión a luxes depende del sensor utilizado.

- **HumidityC:**

El siguiente componente que se va a explicar es el componente HumidityC.NC, el cual se usa para obtener lecturas procedentes de los sensores de humedad y temperatura que lleva instalados el mote TelosB.

Al igual que el componente HamamatsuC, ofrece varias interfaces ADC, dos de ellas para las lecturas de temperatura y humedad, otras dos para controlar los errores en las interfaces anteriores, y otra que controla la inicializacion,arranque, y parada del componente HumidityC, denominada interfaz SplitControl.

Para crear el atributo TempTelos y HumedTelos, hay que conectar en el archivo de configuración TempTelos.NC como se muestra en el recuadro:

```
components  HumidityC, TempTelosM,Attr,LedsC;
        StdControl  =TempTelosM.StdControl;
        TempTelosM.SplitControl -> HumidityC.SplitControl;
        TempTelosM.HumidityAttr -> Attr.Attr[unique("Attr")];
        TempTelosM.TempAttr -> Attr.Attr[unique("Attr")];
  TempTelosM.HumidityError ->HumidityC.HumidityError;
  TempTelosM.TemperatureError ->HumidityC.TemperatureError;
  TempTelosM.Temperature->HumidityC.Temperature;
  TempTelosM.Humidity->HumidityC.Humidity;
 TempTelosM.Leds->LedsC;
```

Figura 43. Configuración Atributo Humidity.

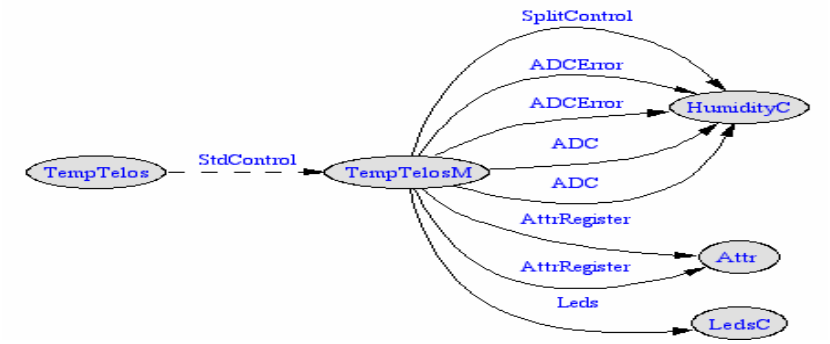


Figura 44. Componentes del Atributo TempTelos.

Comentar que con la temperatura , se ha tenido que hacer las conversiones a grados centigrados y con la humedad realizar una conversion a un valor relativo,respectivamente.

- **InternalVoltajeC e InternalTempC:**

Por ultimo en esta categoria de atributo split-phase comentar como se ha implementado los atributos referentes a variables de estado del TelosB como son el voltaje interno del microcontrolador MSP430 y su temperatura interna:

Aquí los componentes que se han utilizado para poder medir el voltaje interno del microcontrolador y su temperatura interna son : InternalVoltajeC e InternalTempC respectivamente.Los podemos encontrar en el siguiente directorio /tinyos-1.x/tos/platform/msp430 debido a que estos sensores estan situados en el microcontrolador. Ambos son muy similares ya que ofrecen el mismo tipo de interfaces, eso si cada una conectada a distintos puertos, para obtener los valores del senor internos de temperatura y del voltaje del microcontrolador.

El conexionado de interfaces para pode obtener el valor del voltaje interno una vez se ha hecho la conversion del valor del conversor ADC a Voltios es:

```
components Attr,InternalVoltageC,VoltIntM,LedsC;

        StdControl=VoltIntM.StdControl;
         VoltIntM.VoltControl->InternalVoltageC.StdControl;
        VoltIntM.VoltajeAttr->Attr.Attr[unique("Attr")];
        VoltIntM.ADCSingle->InternalVoltageC;
        VoltIntM.Leds->LedsC;
                }
```

Figura 45. Configuración Atributo VoltajeInterno.

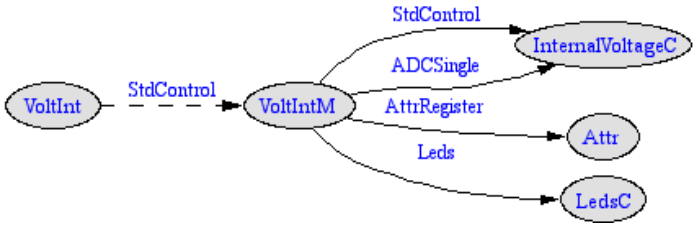


Figura 46. Componentes del Atributo VoltIntM.

El conexionado de interfaces para poder obtener el valor de la temperatura interna del microcontrolador msp430, una vez se ha realizado la conversion del valor del conversor ADC a grados centigrados es:

```
        components Attr,LedsC,TempIntM,InternalTempC;

        StdControl=TempIntM.StdControl;
        TempIntM.TempCont->InternalTempC.StdControl;
        TempIntM.TempIntAttr->Attr.Attr[unique("Attr")];
        TempIntM.ADCSingle->InternalTempC;
        TempIntM.Leds->LedsC;
                }
```

Figura 47. Configuración Atributo Temperatura interna.

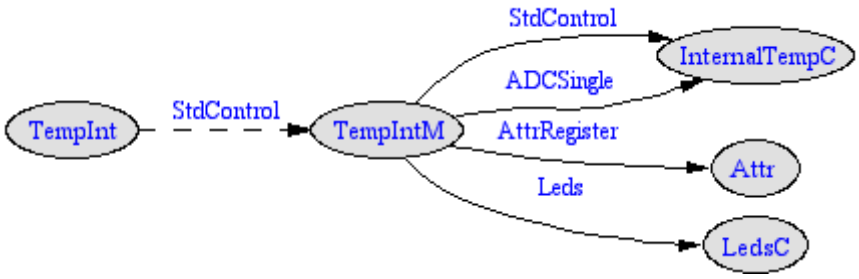


Figura 48. Componente Atributo TempeInt.

- **Atributos No Split-Phase:**

Estos atributos son mucho mas sencillos que los anteriores y menor número, ya que sólo hay tres: el AttrGlobal.NC que está compuesto por dos atributos, y el ApTelos.NC.

AttrGlobal.NC: está compuesto por dos atributos el "nodeid" y "group". Su implementación es muy sencilla, y con estos dos atributos podemos consultar y modificar tanto la dirección como el grupo de un nodo que contenga estos atributos.

ApTelos.Nc:en su configuracion encontramos el componente CC2420ControlM.NC el cual se usa para poder modificar la potencia de transmision de la radio que lleva el telosB (CC2420). Con este atributo podemos consultar y modificar la potencia de transmision de la radio, pero solo en un rango de valores permitidos. Estos valores son desde 3 a 31(enteros). El tres corresponde a -25dbm y el 31 a 0dbm(1mW).

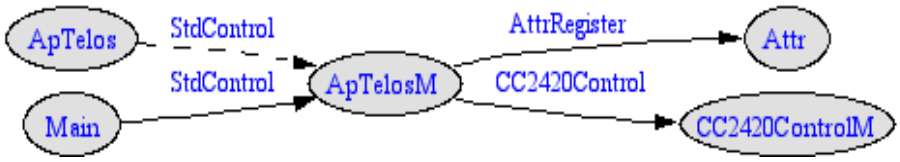


Figura 49. Atributo ApTelosM.

2.8.2 Comandos:

Para la creacion de los comandos, se ha utilizado en todos ellos los mismos componentes e interfaces que son: Command.NC y AttrUse.NC los cuales se pueden encontrar en /tinyos-1.x/tos/lib/Command.NC y en /tinyos-1.x/tos/lib/Attr.NC.

El componente Command.NC se utliiza para registrar comandos, implementar sus eventos, y para realizar las invocaciones de comandos. La interfaz AttrUse.NC se usa para poder acceder a los atributos para consultar o modificar si se puede el atributo,cuando se señaliza el evento commandFunc de un comando.

Para dar una mejor vision del conexionado, muestro el archivo de configuracion del comando "GetTSR" el cual utilizamos para obtener una lectura del sensor de luz solar TSR(total solar radiation):

```
implementation
 {
        components Command, CommandSolarM, Attr,LedsC;

        StdControl = CommandSolarM;
        CommandSolarM.RegisterTSR ->
Command.Cmd[unique("Command")];
        CommandSolarM.AttrUse -> Attr.AttrUse;
        CommandSolarM.Leds->LedsC; }
```

Figura 50. Comando CommandSolar

Figura 51. Componentes del Comando ComandSolar

2.9 Compilar y ejecutar en TinyOS

La sintaxis utilizada en la consola Cygwin para compilar una aplicación en TinyOS es la siguiente:

1) Identificar el puerto COM con el comando motelist.
   Para ello se debe conectar el mote al PC, abrir una consola Cygwin y teclear dicho comando. Este comando devolverá el COM que debes utilizar.

2) Compilar e instalar la aplicación TinyOS

El comando adecuado es make telosb install,<*id_nodo*> bsl,<*N*> ó make telosb reinstall,<*id_nodo*> bsl,<*N*> en el directorio de la aplicación deseada, donde <*id_nodo*> es el la dirección que se le quiera asignar al mote (este valor es opcional) y <*N*> es el número de puerto asignado con el comando motelist menos uno.

La diferencia entre install o reinstall está en que install,<*id_nodo*> compila la aplicación para la plataforma seleccionada y lo programa; y en cambio, reinstall,<*id_nodo*> instala el programa precompilado, pero no recompila.

2.10 Conclusiones.

TinyOS es un sistema operativo de código abierto, diseñado para las redes de sensores inalámbricas. Ofrece una arquitectura basada en componentes, que permite la innovación y rápida puesta en práctica de aplicaciones para este tipo de sistemas, mientras se reduce al mínimo el tamaño del código, según los requisitos inherentes en redes de sensores (memoria y energía, por ejemplo).

NesC y TinyOS se encuentran profundamente relacionados, cualquier avance o desarrollo de uno se verá reflejado en el otro. Es por esto que se hace necesario en su totalidad el buen manejo de este lenguaje.

Aunque hay algunas tareas que son críticas en tiempo, como son el manejo de la comunicación por radiofrecuencia o el obtener datos de algún sensor, TinyOS no se focaliza en dar garantías estrictas de Tiempo Real. Esto se refleja en que si frente a la llegada de algún evento, puede ocurrir que no sea posible la atención, éste y su información será perdido. Ahora bien, estas falencias son muy bien compensadas con el modelo de manejo de eventos.

TinyOS esta programado sobre un lenguaje estático, por lo cual no existen asignaciones de memoria de manera dinámica y los gráficos de llamados quedan totalmente definidos en el tiempo de compilación. Esto facilita enormemente el análisis de programas, y por ende la detección de errores.

La biblioteca de componentes de TinyOS incluye protocolos de red, servicios distribuidos, manejadores de sensores, y herramientas de adquisición de datos, que pueden ser ocupadas directamente, o manipuladas para responder a necesidades más específicas.

El modelo de manejo de eventos en la ejecución de TinyOS permite una administración al detalle de la energía, aún permitiendo mantener toda la flexibilidad que los sistemas de comunicación e interfaces físicas imponen.

Se debe tener cuidado con que se produzcan carreras de datos, esto debido a modificaciones concurrentes a un estado compartido. El compilador NesC es capaz de detectar posibles problemas de este tipo, mas es el usuario el que reparará el error.

TinyOS ha migrado a varias plataformas, corroborando de esta forma su estabilidad y su verificación de funcionamiento. Además se han desarrollado aplicaciones capaces de realizar simulaciones, herramienta que una amplia comunidad utiliza para desarrollar y probar varios algoritmos y protocolos.

Continuamente grupos numerosos de desarrolladores están contribuyendo código al sitio del *sourceforge*, y están trabajando activamente.

APLICACIÓN DESARROLLADA.

3.1. Introducción.

La aplicación desarrollada en este PFC tiene como objetivo fundamente el control de forma centralizada de varias redes de sensores de forma simultanea desde un único GUI, pudiendo realizar lecturas de los sensores integrados en los nodos sensores, almacenar los valores en ficheros para su posterior uso, realizar representaciones gráficas de la evolución de las magnitudes que se puede medir en el medio con los sensores a lo largo del tiempo, controlar los principales parámetros de funcionamiento de un sensor como son la potencia de transmisión de la radio, el grupo de red al que pertenece, o su dirección dentro de un grupo de red.

La aplicación instalada en los motes, permite realizar lecturas de la luz solar y la luz infrarroja, lecturas de temperatura y humedad, hacer ping a los sensores, valor de la potencia de la radio, el voltaje interno y la temperatura del microcontrolador msp430, eso en lo que se refiere a lecturas, y en cuanto a capacidad de modificación se puede cambiar el grupo y la dirección de un sensor de la red, reprogramar sensores con distintas imágenes, borrar imágenes, descargar imágenes de los sensores.

También permite realizar representaciones gráficas de los atributos en unidades correspondientes del sistema internacional, es decir: para temperaturas en grados centígrados, para ambas lecturas de luz en Luxes, la humedad en humedad relativa en %, la potencia de transmisión de la radio en Wm, la tensión del microcontrolador en Voltios.

Ejecutar acciones como encender un led del sensor, activar el Timer del sensor, o actuar sobre algún mecanismo conectado al sensor, programando previamente los componentes que lo manejan.

Reprogramar de los nodos sensores de la red desde nuestra aplicación, a través de el envío de imágenes precompiladas desde nuestra aplicación, obtener información de todas las imágenes que tiene un nodo sensore almacenadas, incluyendo la que se está ejecutando, descargar una imagen almacenada en un nodo sensor y borrar y resetear los datos referentes a una imagen almacenada o instalada en un nodo sensor.

A lo largo de este capítulo, se va a poder conocer la aplicación desarrollada denominada “WSNMonitor”desde dos puntos de vista, por un lado se va a ver la aplicación desarrollada desde el punto de vista del código NesC, y por otro lado desde el punto de vista del código Java. Esto es así porque para la construcción de esta aplicación, se ha podido diferenciar 2 partes: la referente a la programación de los nodos sensores utilizando el código NesC, y la del diseño de la GUI y el manejo de comunicaciones con la red de nodos sensores desde nuestra GUI a través de las clases java creadas.

Aplicación Desarrollada.

Una vez explicado el motivo por el cual se ha hecho esta división, se comenzará explicando la parte que concierne a la red de sensores, como son los mensajes enviados desde los nodos sensores hacia el PC y la estructura general de nuestra aplicación NesC; para una vez concluido esto, se explicarán las clases javas que se encargan de la comunicación con la red de sensores desde el PC, las librerías utilizadas para la construcción de la interfaz Gráfica de Usuario y el Osciloscopio, y para finalizar este apartado se explicarán las funciones que podemos realizar desde dicha Interfaz, y un ejemplo de su uso con capturas de pantalla.

3.2 Programación de los Nodos Sensores de la Red.

En este apartado se va a explicar las funciones programadas en los nodos sensores con el uso del lenguaje de programación NesC.

Tanto para la adquisición de datos, como para las comunicaciones del sensor con el PC, se ha hecho uso de TinySchema, el cual es esencial para la elaboración de este proyecto, ya que gracias a el, se ha podido crear componentes y atributos totalmente independientes, con la ventaja de poder añadirlos o eliminarlos en un nodo sensor de forma independiente a la GUI que se ejecuta en el PC, y su uso para la invocación de los comandos desde la GUI.

Otro componente importante que se ha añadido en la programación de los nodos sensores, ha sido el componente Deluge, con el cual se ha podido añadir las funciones de poder reprogramar los nodos sensores desde la GUI, de forma automática con tan solo seleccionar un número de imagen y la imagen en cuestión a inyectar en el sensor, descargar imágenes desde un sensor y almacenarlas en el PC, y obtener información sobre las imágenes cargadas en un nodo sensor.

Como ya se ha comentado en el capitulo 2 de esta memoria TinySchema, la creación de atributos y comandos, y una descripción genérica de los atributos y comandos que se han utilizado en la programación de los nodos sensores, pero sin embargo no se ha hablado de las comunicaciones entre los nodos de la red y el PC, se dará paso a explicar los mensajes utilizados en las comunicaciones, y por ultimo en otro apartado se dará una visión global de la estructura de la aplicación NesC programada en los nodos sensores.

3.2.1 Estructuras de los Mensajes.

Antes de hablar de los mensajes utilizados en esta aplicación, es imprescindible de explicar como se envían los mensajes por la red, ya que se envían encapsulados dentro de la parte de datos del mensaje del tipo definido por TinyOS TOSMsg.

TOSMsg es un formato de mensajes (para cada tipo de sensor hay uno distinto) el cual viene especificado en la siguiente ruta: /tinyos-1.x/tos/plataform/telos/AM.h. Se puede decir que su función es hacer de contenedor genérico para el envío de mensajes entre plataformas iguales, es decir, un mensaje enviado por un sensor TelosB no puede ser reconocido por un sensor tipo Mica, Mica2dot, etc...

Cada tipo mensaje se distingue por un valor distinto del campo AM_TYPE del mensaje, y cuando se recibe un mensaje se ejecuta el manejador de mensaje correspondiente al AM_TYPE recibido.

Para poder encapsular nuestros mensajes dentro del campo de datos de los mensajes TOSMsg que son los que realmente se envían, lo que debemos hacer, es escribir en nuestra aplicación NesC el siguiente código:

```
TOS_Msg nombres; // creamos el contenedor.
struct ListaAttr *info;  //creamos nuestra estructura.
Info=(struct ListaAttr*)nombres.data;

 /* hacemos que nuestra estructura apunte al campo data del mensaje
nombres*/

call SendNombres.send(TOS_BCAST_ADDR,sizeof(struct
ListaAttr),&nombres);
```

Figura 52. Estructura de los Mensajes

Al tener una comunicación bidireccional, podemos diferenciar 2 tipos de mensajes: los que van desde los sensores a la estación base, y los que su sentido es el contrario, es decir, desde el PC hasta los nodos destino.

Se iniciará mostrando y explicando las estructuras de los mensajes que van desde los nodos sensores hacia el PC.

Los sensores sólo envían un tipo de Mensaje denominado “ListaAttr”, en el cual se puede enviar: los nombres de los comandos que tiene el sensor, los nombres de atributos que posee el sensor, su dirección, el grupo al que pertenece, un identificador para saber que tipo de información transporta, y los datos que nos interesan.

En el apartado sobre MIG (Message Interface Generation) se puede encontrar toda la información necesaria para poder crear a partir de estructuras, clases javas para que la aplicación “WSNMonitor” pueda procesar los mensajes envidados desde los sensores con destino el PC.

```
struct ListaAttr {

uint8_t type;
uint8_t nodeid;
uint8_t groupid;
uint16_t dato;
float fAdc;
unión{
char name[10];};
};
```

Aplicación Desarrollada.

Figura 53. Estructura Mensaje ListaAttr

El campo type es un entero de 8 bits sin signo, la función de este campo de la estructura es como se verá en profundidad mas adelante identificar si se recibe un ping, un lectura de un atributo non split-phase, si es de un split-phase, si el valor es para enviarlo al osciloscopio, así hasta 6 casos distintos se pueden tener.

-El campo nodeID contiene la dirección TOS_LOCAL_ADDRESS del nodo fuente que envía el mensaje.

-El campo groupid contiene el grupo TOS_AM_GROUP del nodo fuente que envía el mensaje.

-El campo dato es un entero de 16 bits en el cual se envían datos que son valores enteros, como por ejemplo la potencia de la radio.

-El campo Fadc es un entero en coma flotante de tamaño 32 bits para enviar los valores que se obtienen de las lecturas procedentes de atributos como temperatura, humedad, voltaje del microcontrolador, etc... Los cuales contienen decimales, ya que estos valores están convertidos a valores del sistema internacional.

-Por ultimo el campo name[10] es un array de caracteres el cual se usa para enviar los nombres de los atributos y comandos que contiene un sensor. Puntualizar que se tubo que hacer uso de una unión para poder enviar el array, ya que sino no se enviaba todas las posiciones de memoria del array, y en consecuencia la transmisión no era correcta.

En el recuadro inferior se puede ver como se rellena la estructura anterior para enviar los nombres de los comandos, en la aplicación NescC:

```
info->name[0]=attrDescs->attrDesc[attr].name[0];
info->name[1]=attrDescs->attrDesc[attr].name[1];
info->name[2]=attrDescs->attrDesc[attr].name[2];
info->name[3]=attrDescs->attrDesc[attr].name[3];
info->name[4]=attrDescs->attrDesc[attr].name[4];
info->name[5]=attrDescs->attrDesc[attr].name[5];
info->name[6]=attrDescs->attrDesc[attr].name[6];
info->name[7]=attrDescs->attrDesc[attr].name[7];
info->name[8]=attrDescs->attrDesc[attr].name[8];
info->name[9]=0;
info->type=1;
   info->dato=numattr;
    info->groupid=TOS_AM_GROUP;
          info->nodeid=TOS_LOCAL_ADDRESS;
          call Leds.greenToggle();
          call SendNombres.send(TOS_BCAST_ADDR,sizeof(struct
ListaAttr),&nombres);
```

Figura 54. Nombres de Comandos.

Seguidamente se comentará los mensajes que se envían desde la aplicación"WSNMonitor" a los sensores de toda la red. Hay dos tipos: los que transportan comandos para ser invocados en los correspondientes nodos destino, y los que únicamente se encargan de indicar que las muestras se envíen al PC, son para procesarlas en el Osciloscopio de nuestra aplicación.

Para esta transmisión se ha hecho uso de la estructura CommandMsg , la cual contiene los siguientes campos:

```
Struct CommandMsg{
  short nodeid;
uint32_t seqNo;
char data[0];
};
```

Figura 55. CommandMsg

-El campo nodeid contiene la dirección de destino.

-El campo seqNo se usa para indicar un número de secuencia.

-El campo data contiene el nombre del comando que vamos a enviar. Para copiar el nombre del comando que queremos enviar hacemos uso de la función "setCommandName(CommandMsg m, String s) " que se encarga de copiar el string que se le pasa como argumento, en el array de caracteres data, y añade un carácter nulo, para indicar el fin del string.

Un ejemplo de envío de este tipo de mensajes, (desde la clase PcTx.java que es la encargada de las comunicaciones entre el PC y la red de sensores), es el mostrado en el recuadro inferior:

```
try{
            CommandMsg packet= new CommandMsg(TamañoMsg);
             int pos =setCommandName(packet,nombre);
              packet.set_nodeid(dest);
             motes[group -1].send(dest,packet);
```

Figura 56. Envio de Mensajes CommandMsg

Sólo queda por explicar el mensaje de tipo Reset que tiene los mismos campos que el mensaje de tipo CommandMsg, pero distinto AM_TYPE. La explicación de que tenga los mismos campos es que al recibir un mensaje con el AM_TYPE =124, (en los sensores)que es el que nos ocupa lugar, lo que se hace es hacer llamar al timer para que salte tantas veces como valor tenga la variable muestras, y en cada salto del timer se realiza una invocación con el comando que contiene en campo data.

De esta forma lo que se consigue es enviar de forma indirecta 2 órdenes: por un lado se activa el envío de tantas muestras como tenga la variable muestras, y por otro lado , se envía el nombre de comando que se debe ejecutar en las invocaciones.

```
struct ResetMsg
{
  short nodeid;
uint32_t seqNo;
char data[0];
};
```

Figura 57. Estructura Mensaje ResetMsg

Por ultimo mostrar ambo AM_TYPES definidos en el archivo Ammensajes.h que es encuentra dentro del archivo Telos en el directorio Apps de TinyOS.

```
enum{
AM_RESETMSG=124,
AM_LISTAATTR=23,
};
```

Figura 58. AM_TYPES.

3.2.2 Estructura de la Aplicación Telos.NC

Una vez se ha hablado de TinyOS, del lenguaje NesC, de TinySchema, de Deluge, de todos los atributos creados y comandos para poder realizar lecturas de los atributos y los mensajes enviados en las comunicaciones con el PC. Se puede dar paso a explicar la aplicación (que podemos encontrar en el directorio apps/Telos), la cual se instalará en los sensores TelosB. Se iniciará mostrando el esquema gráfico que se obtiene cuando se utiliza la herramienta Nesdoc de TinyOS. El comando ejecutado en TinyOS estando en el directorio /apps/Telos , es el siguiente:

Make telosb docs.

Aplicación Desarrollada.

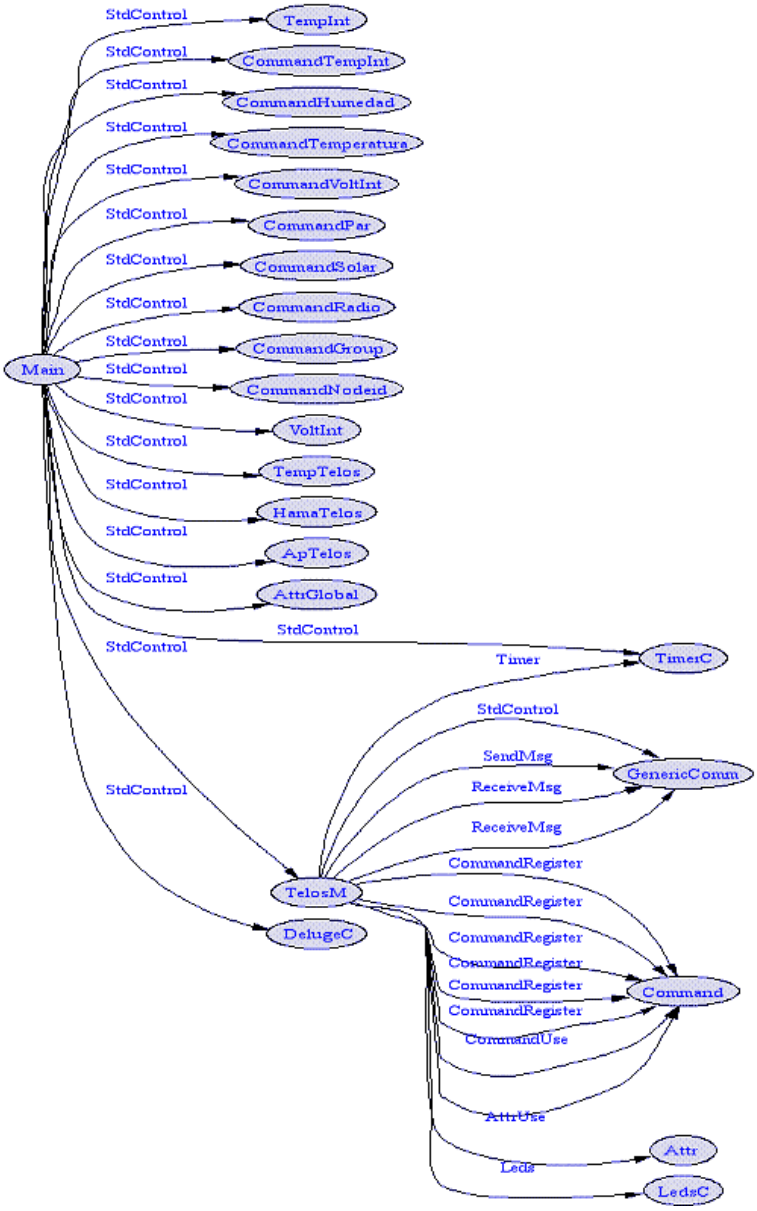


Figura 59. Estructura Telos.NC

Como se aprecia en la imagen superior, todos los atributos y comandos están conectados con el componente Main, esto es una condición de TinySchema para poder utilizarlos desde la aplicación. También están conectados al componente Main, el componente DelugeC, el TimerC y el modulo de la aplicación TelosM.NC.

La aplicación ofrece la interfaz StdControl en la cual en sus 3 comandos(init(),star() y stop()) se realiza las siguientes tareas: registramos nuevos comandos los cuales se implementan en el componente de la aplicación principal, se activan los atributos que hay en la aplicación, se crean e inicializan variables que se usarán en el componente TelosM.NC, se definen estructuras de los mensajes que se van a utilizar, también se definen mensajes TOS_Msg, se inicializan los Leds, los contadores.

Seguidamente hay definidas las interfaces de enviar y recibir mensajes: con las cuales se van a conectar con la red de sensores y PC.

Como solo se envían un tipo de mensajes, solo hay una interfaz SendMsg, la cual se usa para enviar los datos que nos se solicitan como resultado de las invocaciones o como respuesta aun mensaje Ping.

Respecto de la interface de recepción de mensajes receivedMessage, se encuentran dos, ya que se puede recibir dos tipos de mensajes con AM_TYPE distintos. La interfaz ReceivedMessage , es la parte fundamental de la aplicación NesC ya que casi todos los mensajes que se reciban serán del tipo AM_COMANDMSG los cuales son procesados por esta interfaz.

En esta parte del código lo que se hace es invocar los comandos que se envían desde la aplicación java "WSNMonitor" a los sensores. Si se hacen invocaciones de comandos no split-phase se envían las respuestas dentro del mismo evento receivedMessage. Si la invocación es de un comando de tipo split-phase, el valor devuelto por la ejecución del comando es enviado en el evento CommandUse.comandDone.

La interfaz OscopeMsg es utilizada para activar la función del Osciloscopio, el cual ya se ha explicado anteriormente(en estructuras de mensajes) como funciona, al igual que la anterior interfaz, según sea el tipo de invocación de comando, el dato devuelto por el comando se enviará en el evento comandUse.commandDone si es una operación split-phase, o en este caso en el evento Fired del timer que se usa como bucle for, para realizar tantas invocaciones como muestras se soliciten para representar en el osciloscopio.

3.3 Programación de la GUI "WSNMonitor"

Para la implementación de la Interfaz Gráfica de Usuario se ha hecho uso de un conjunto de librerías java, como son Swing, Java2D, la librería Message, la parte Java de la herramienta Deluge, y un conjunto de clases creadas para la creación de la GUI, para el controlo de las comunicaciones y para el buen funcionamiento de la aplicación.

En consecuencia se puede hacer una distinción entre todas las clases que están dentro de la aplicación que es objetivo de este proyecto, y agruparlas por funcionalidades como son la comunicación, la construcción de la GUI, y las clases que implementan las funcionalidades de la GUI.

3.3.1 Clases Java que Controlan las Comunicaciones Con la Red De Sensores Inalámbricas

- **Clase Pctx.java:**

Esta clase se podría decir que después de la clase Telos que contiene al método main, es la más importante, ya que se encarga de las comunicaciones entre el PC y la red de sensores, a través del SerialForwarder, y del nodo que hace de estación base (con la aplicación TransparentBase instalada, la cual permite trabajar con distintos grupos de redes de sensores).También cumple la función de procesar los datos recibidos por los mensajes procedentes de la red de sensores, para su posterior almacenamiento y procesamiento de los datos, y también obedece a la instrucciones que se ejecutan desde la Interfaz Grafica de Usuario de la aplicación.

Para el envío de comandos se utiliza el método" Comando(String nombre,short dest,short group)".El cual se le pasamos como parámetros de entrada el nombre del comando que queremos enviar, la dirección de destino y el grupo al cual se envía.

Si hay que enviar también algún dato para modificar por ejemplo la potencia de la radio, o modificar el atributo nodeid o groupid de un nodo se utilizará el método "public void Comando(String nombre,short dest,short group,short value)", donde value es el dato a enviar.

Para enviar el comando Ping se puede enviar o bien a un nodo concreto de un grupo de red o hacer un broadcast a todos los grupos. Por tanto hay dos métodos que se llaman igual, pero al segundo no se le pasan parámetros de entrad y al primero si.

- public void SendPing(short dest,short group).
- public void SendPing().

También destacar el método Ping, el cual se encarga de ir almacenando de forma ordenada cada uno de los nodos que responden al comando Ping enviado, cuando se envían un ping a todos los grupos, de forma que todos los nodos de un mismo grupo aparecen en la misma lista de nodos de un grupo. Este método devuelve la lista de grupos con todos los nodos que se han recibido ordenados por grupos.

La lista de grupos es un array de objetos de tipo ListaGrupos, el cual es una clase escrita en java , que esta formada por un array de objetos de tipo ListaGrupos. En el constructor de la clase ListaGrupos le pasamos el grupo que queremos almacenar, y dentro de el, tiene un array para almacenar las direcciones de los distintos nodos que pertenecen a ese grupo. Los nodos son añadidos a los grupos mediante el método AnadirNodo(int nodeid), que comprueba si ya existe esa dirección antes de añadirlo.

- **Clase MoteIF.java:**

Esta clase la cual se usa para crear un array de objetos MoteIF de tamaño igual al número mas alto de todos los grupos(para poder acceder a todos los grupos que existan) es la que se encarga de conectarse con el SerialForwarder y registrar los mensajes que se van a procesar. Para ello la clase Pctx implementa la interfaz MessageListener la cual es invocada cada vez que se recibe un mensaje.

El constructor de la clase MoteIF usa la variable de entorno MOTECOM, para determinar como se conecta la aplicación java a la estación base, que actúa como repetidor.

```
motes=new MoteIF[NUM_GRUPOS];
        System.out.println("creamos objeto Pctx");
        for (i=1;i<=NUM_GRUPOS;i++){
        motes[k] = new MoteIF(PrintStreamMessenger.err,i);
        motes[k].registerListener(new ListaAttr(),this);
              motes[k].registerListener(new CommandRx(),this);
```

Figura 60.MoteIF

En el recuadro que se muestra se puede ver como se crea el array de objetos de tipo MoteIF, y en el constructor se le pasa el grupo de red que se le asigna a ese objeto MoteIF.Asi cuando se desee enviar un mensaje del grupo de red 8 se utilizará el método send() del mote[7].

Antes de continuar es importante explicar dos variables importantes que son: NUM_GRUPOS y NUM_NODOS, que representan el número de grupos y el numero de nodos por grupo respectivamente.

Una vez que ya se tiene los objetos necesarios para poder enviar mensajes a la red, se pasa a explicar como se ha procesa los mensajes que se reciben desde la red de sensores a través de la estación base y el SerialForwarder que conecta con la estación base.

- **Interfaz MessageListener:**

Su función es la de escuchar mensajes del tipo net tinyos.message.Message. Tiene un único método que es: public void messageReceived(int to, Message m). Donde to es la dirección de destino y m es el mensaje recibido. Este método señaliza la recepción de mensajes.

En la clase Pctx.java se implementa dicha interfaz y su método de la siguiente forma:

```
public void messageReceived(int dest, Message msg){
                MensajesRx++;
            System.out.println("mensaje recibido\n");
                        if(msg instanceof ListaAttr)
             ListaReceived(dest,(ListaAttr)msg);
```

Figura 61.Message Listener

En el método “ListaReceived(dest,(ListaAttr)msg)” de la clase PcTx.java es donde según el valor del campo Type, mediante un switch se procesa el mensaje llamando a un método u otro de la clase Pctx.java.

```
public void ListaReceived(int
dest,ListaAttr msg){
          int type=msg.get_type();
           switch(type){
             case 1:
            ListAtributos(msg);
            break;
            case 2:
             ListComandos(msg);
            break;
            case 3:
             Ping(msg);
            break;
            case 4:
            PotRadio(msg);
            break;
            case 5:
            Adc(msg);
            break;
            case 6:
            Osciloscopio(msg);
            break;
            default:
            //PotRadio(msg);
            break;
         }
                        }
```

Figura 62. Manejador de Mensajes Recibidos.

```
public class ListaGrupos {
                 static int NUM_NODOS=25;
                public int lista[];
                private int k=0;
                int i;
                public int numgrupo;
        public ListaGrupos(int num){
                 numgrupo=num;
                 lista=new int[NUM_NODOS];
                 for(int j=0;j<lista.length;j++)
                 lista[j]=-1;
                }
        protected void  AñadirNodo(int nodeid ){
                                if (nodeid!=-1){
                for(i=0;i<lista.length;i++){
                                        if(nodeid==lista[i])
                                           return;}
                                                lista[k]=nodeid;
                                         k++;
                                        }
                                                }}
```

Figura 63. Clase Lista Grupos.

3.3.2 librerías para la creación de la Interfaz Gráfica de Usuario.

- **Java2D**

En este punto se comentará brevemente la API 2D de java, la cual se hace uso en la clase GraphPanel.java, para dibujar el eje de coordenadas, las líneas, para indicar el mote y el comando del gráfico, los gráficos que mostramos, y las herramientas como el zoom, el desplazar los ejes X e Y a izquierda o derecha y hacia arriba o abajo respectivamente, y las demás utilidades que presenta dicha clase.

La API Java2D amplía muchas de las capacidades gráficas de la biblioteca AWT (Abstract Window Toolkit - Herramientas Abstractas de Ventanas), permitiendo la creación de mejores interfaces de usuario y de aplicaciones Java mucho más impactantes visualmente. El rango que abarcan todas estas mejoras es muy amplio, ya que comprende el renderizado, la definición de figuras geométricas, el uso de fuentes de letras, la manipulación de imágenes y el enriquecimiento en la definición del color.

También permite la creación de bibliotecas personalizadas de gráficos avanzados o de efectos especiales de imagen e incluso puede ser usada para el desarrollo de animaciones u otras presentaciones multimedia al combinarla con otras APIs de Java,

como puedan ser JMF (*Java Media Framework* - Entorno de Trabajo de Java para Medios Audiovisuales) o Java 3D.

El API 2D de Java nos permite fácilmente:

- Dibujar líneas de cualquier anchura.
- Rellenar formas con gradientes y texturas.
- Mover, rotar, escalar y recortar texto y gráficos.
- Componer texto y gráficos solapados.

- **Javax.swing:**

Swing es una plataforma independiente, Model-View-Controller Gui framework para Java. Sigue un simple modelo de programación por hilos, y posee las siguientes principales características:

- Independencia de plataforma:

Swing es una plataforma independiente en ambos términos de su expresión (java) y de su implementación (no-nativa interpretación universal de widgets).

- Extensibilidad:

Swing es una arquitectura altamente particionada que permite la utilización de diferentes pluggins en específicos interfaces de diferentes frameworks: Los usuarios pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto. En general, los usuarios de swing puenden extender el framework para: extender clases existentes (framework); proveyendo alternativas de implementación para elementos esenciales.

- Orientado a componentes:

Swing es un framework basado en componentes. La diferencia entre objetos y componentes es un punto bastante sutil: concisamente, un componente es un objeto de buena conducta con un patrón conocido y especificado característico del comportamiento.

- Customizable:

Dado el modelo de representación programático del framework de swing, el control permite representar diferentes 'look and feel' (desde MacOS look and feel hasta Windows XP look and feel). Más allá, los usuarios pueden proveer su propia implementación look and feel, que permitirá cambios uniformes en el look and feel existente en las aplicaciones Swing sin efectuar ningún cambio al código de aplicación.

- Lightweight UI:

La magia de la flexibilidad de configuración de Swing, es también debido al hecho de que no utiliza los controles del GUI del OS nativo del host para la representación, pero usa "algo" de sus controles programados, con el uso de los apis 2D de Java.

Con esta librería implementamos la clase Telos.java la cual contiene al método main, y en la cual dibujamos todos los paneles, botones, menús, manejo de eventos, y demás objetos que representamos y forman la Interfaz Gráfica de Usuario de la aplicación. En este bloque podemos añadir las siguientes clases:

-Panel deluge.java: dibuja la ventana deluge, sus componentes y maneja sus eventos.
-FiltroFile: filtra archivos .xml en la ventana para seleccionar archivos.

Para la elaboración des panel que representa el Osciloscopio, se ha utilizado ambas librerías una para representar las gráficas y otra para dibujar los botones y el manejo de eventos que se dan durante su ejecución.

3.3.3 Deluge

El paquete Deluge esta compuesto por 14 clases java las cuales cada una tiene una función específica. La clase principal de este paquete es la clase Deluge.java, ya que desde hay es donde se instancian objetos del resto de las clases en función de cual es el comando a ejecutar, que en la aplicación corresponde con el botón pulsado. Debido a que este conjunto de clases se explican con mayor profundidad en el capitulo dos, aquí se mencionara dichas clases y se dará una breve explicación de cada una de ellas.

Las principales clases del paquete Deluge, son:

-Pinger.java: para ejecutar un ping.
-ImageInjector.java: para inyectar imágenes en los nodos sensores.
-Rebooter.java: para reiniciar una imagen almacenada en un nodo sensor.
-Eraser.java: para borrar una imagen almacenada en un nodo sensor.
-DelugeCrc.java: para comprobación del CRC en los mensajes.
-Downdloader.java: para descargar imágenes desde un nodo sensor.
-DelugeReqMsg, DelugeDataMsg, DelugeAdvMsg, NetProMsg : para las comunicaciones con los nodos sensores que tienen instalados el componente DelugeC.
-DelugeConsts.java: constantes utilizadas por Deluge.

3.4 Manual de Usuario de de la GUI “WSNMonitor”

Puesto que ya se ha explicado anteriormente la parte NesC de la aplicación, ahora se dará paso a la explicación de la interfaz Gráfica de Usuario. Se comenzará explicando el uso de la interfaz grafica, debido a que ya se ha explicado como se realiza la transmisión y recepción de mensajes desde el PC hacia la red, como se usa Deluge, y el Osciloscopio, y por tanto sólo queda explicar como se usa la interfaz.

Para poder empezar a utilizar la aplicación “WSNMonitor” se debe inicialmente llevar a cabo varios pasos:

1. Abrir la consola Cygwin para ejecutar SerialForwarder desde TinyOS.
2. Conectar el nodo que actúa como estación base al PC.

3. Iniciar la herramienta SerialForwarder desde Cygwin, para que se comunique con el nodo que actúa de estación base. Para ello ejecutamos el comando: java net.tinyos.tools.sf.SerialForwarder –comm serial@COM9:telos. En nuestro caso el nodo que hace la función de estación base, esta conectado al puerto COM9.

4. Ejecutar comando desde consola Cygwin : java net.tinyos.schema.Telos

Con este comando lo que se hace es llamar a la clase Telos.java del paquete schema, el cual contiene el método main con lo que la aplicación "WSNMonitor" se inicia.

5. Una vez iniciada la aplicación se debe introducir: el número de grupos que se va a tener en nuestra red o redes de sensores (introducir el grupo de red mayor de todos los grupos existentes), y el número máximo de nodos que vamos a tener por grupo de red.

6. Finalizado el punto 5, se podrá visualizar la interfaz gráfica de nuestra aplicación. Como se puede ver en la imagen inferior.

7. Seguidamente lo debemos hacer un ping a todos los nodos de todos los grupos de red que hemos introducido en el punto 5. Para ello, se debe pulsar el botón Actualizar Lista.

8. Una vez transcurridos un par de segundos, se puede asegurar que han respondido al ping todos los nodos que estén activos, y se procederá a pulsar el botón"Mostrar Lista Grupos", donde se mostrará todos los nodos de cada uno de los grupos que están en la red sensores.

Llegado a este punto, se pueden realizar las siguientes tareas:

- consultar los atributos que dispone un nodo específico.
- Consultar o modificar algún atributo de un nodo específico mediante el uso de comandos.
- Representar gráficamente una secuencia de lecturas de un atributo de un nodo específico.
- Hacer uso de las posibilidades de reprogramación de los sensores a través de Deluge.

A continuación se Irán explicando cada una de estas tareas de forma más profunda.

Consulta del número de Atributos:

Para realizar una consulta de los atributos que tiene un nodo especifico, tan solo hay que seleccionar un nodo de la lista de nodos que se muestra en la parte central, y seguidamente pulsar el botón Mostrar Atributos.

Consultar o Modificar Atributos mediante el uso de Comandos :

Para realizar una consulta del valor de un atributo, tenemos que seleccionar por un lado un nodo de la lista de nodos, y por otro lado el comando correspondiente que realiza la lectura del atributo en cuestión, y una vez hecho esto se deber pulsar el botón Obtener Consulta, y obtendremos el resultado mostrado en una ventana , en la que podremos guardar, copiar, cortar el valor mostrado.

De manera similar para realizar una modificación de un atributo, seleccionaremos un nodo de la lista de nodos y pulsaremos el botón Modificar Parámetro. Una vez pulsado aparecerá una ventana con los posibles atributos a modificar y un campo para introducir el valor a modificar.

En este punto se va explicar de forma esquemática, cual es la función de cada comando que podemos ver al pulsar el botón Mostrar Comandos:

-NumSample: nos devuelve el valor de la variable que controla el número de muestras que se toman cuando está activo el osciloscopio.

-NumAttr: Este comando se usa para mostrar los atributos que hay en un sensor determinado. No devuelve ningún valor de lectura.

-NumComand: igual que el anterior pero en este caso para los comandos de un nodo en particular.

-SetSample: para modificar el número de muestras que se van a enviar, cuando esta activo el osciloscopio.

-StopOscil: este comando se utilizar para finalizar el envío de lecturas de un nodo sensor con destino el osciloscopio.

-GetAddr: nos devuelve el valor de la variable TOS_LOCAL_ADDRESS(dirección).

-SetAddr: para modificar el valor de la variable TOS_LOCAL_ADDRESS (dirección)

-GetGroup: nos devuelve le valor de la variable TOS_AM_GROUP(grupo de red)

-SetGroup: para modificar el valor de la variable TOS_AM_GROUP(grupo de red)

-PotTelos: para obtener el valor de la potencia de transmisión de la radio.

El rango de valores va desde 0(valor mínimo) hasta 31(valor máximo).

-PotSetTelos: para modificar el valor de la potencia de transmisión de la radio.

El rango de valores va desde 0(valor mínimo) hasta 31(valor máximo).

-GetTSR: devuelve el valor de la luz solar en luxes.

-GetPAR: devuelve el valor de la luz infrarroja en luxes.

-VoltajeMi: devuelve el valor del voltaje interno del microcontrolador MSP430.

-GetTempe: devuelve el valor de la temperatura ambiental en grados centígrados.

-GetHumdt: devuelve el valor de la humedad relativa ambiental en %.

-TempeInte: devuelve el valor de la temperatura interna del microcontrolador MSP430.

Representación Gráfica de las lecturas de los atributos de un nodo sensor:

En esta ocasión, al igual que en las anteriores como primer paso hay que seleccionar un nodo de la lista. después, pulsar el botón Start Osciloscopio, aparecerá un cuadro de dialogo que pedirá el numero de muestras y seguidamente el comando en cuestión. Una vez realizados estos pasos se Irán visualizando en el gráfico, cada una de la lecturas que se van recibiendo desde el nodo sensor seleccionado. Para abortar el envío de lecturas, tan sólo hay que pulsar el botón Stop Osciloscopio.

Para hacer zoom, o desplazar los ejes X o Y y demás opciones , se dispone de los botones en la parte inferior de la gráfica. También se pueden almacenar las lecturas recibidas en un archivo, con tan sólo pulsar el botón Guardar Datos, a su vez durante el funcionamiento del osciloscopio los posibles errores durante su ejecución, se pueden acceder en el archivo blog.

Reprogramación mediante Deluge:

Para poder utilizar Deluge tan solo hay que acceder a la barra del menú principal y en Deluge, pinchar en Abrir. Una vez hecho esto, aparece un panel de información que pide

que se introduzca un número de grupo de red para el cual se va reprogramar o hacer uso. Si existieran varios grupos distintos y se quiere trabajar con todos de forma simultanea, tan sólo habría que abrir tantas ventanas Deluge con su grupo correspondiente como grupos hubiera en la red.

Una vez hecha esta aclaración, en la ventana se pueden ver varios botones, cada uno para cada una de las utilidades de la herramienta de Deluge, y dos ChekcBox en la parte superior, cuyo fin es elegir entre enviar los mensaje de ping a todos los nodos de un grupo(broadcast), o tan solo a uno en concreto de la red(unicast).

Las funciones que se pueden realizar desde esta ventana son:

Consultar el estado de las imágenes almacenadas en un sensor, pulsando el botón Ping.

Reiniciar una imagen cargada en un nodo sensor con tan solo pulsar el botón Reboot e introducir el número de imagen a reiniciar.

Inyectar una imagen pulsando el botón Imagen Inject, para ello antes tendremos que seleccionar un numero de imagen y posteriormente aparecerá un navegador para buscar la imagen a inyectar.

Borrar una imagen de un grupo de red, para ello solo tenemos que pulsar el botón Borrar Imagen y seleccionar el número de imagen a borrar.

Recetar los datos de una imagen almacenada en un nodo sensor, pulsando el botón Remeterá Imagen y eligiendo el número de imagen a resetear.

Descargar una imagen almacenada en unos nodos sensor, pulsando para ello el botón Descargar Imagen y eligiendo posteriormente el nombre del archivo a guardar.

Y por ultimo tenemos el botón de Limpiar Área que sirve para borrar todos lo escrito en el componente JtextArea.

Por ultimo destacar que disponemos de un menú con diversas opciones:

Fichero: Para cerrar la aplicación.

Editar : con el que podemos realizar las operaciones de copiar, pegar y cortar.

Contador de Mensajes: para consultar el número de mensajes que enviamos y que recibimos. Se puede actualizar pulsando el botón Actualizar, y resetear con el botón Resetear.

Deluge: como se ha dicho anteriormente para abrir la ventana Deluge.

Ayuda: pulsando el botón Ayuda recibimos información general para poder utilizar la aplicación “WSNMonitor”, y con el pulsando el botón Acerca De nos aparece un panel con el autor y director de dicha aplicación.

3.5 Ejemplos prácticos del uso de la GUI “WSNMonitor”.

Como bien se ha explicado en el punto anterior los pasos a seguir para primero ejecutar la GUI desarrollada en este PFC, y posteriormente hacer uso de ella, en este

apartado lo que se va a intentar es dar un punto de vista practico de la aplicación desarrollada y reflejar su buen funcionamiento.

Se comenzará haciendo una operación básica como es hacer detectar todos los nodo, disponibles en una red, que en este ejemplo serán dos, para ir después mostrando todas las funcionalidades que se han implementado en esta aplicación:

Una vez tenemos el serialForwarder correctamente configurado y activado, iniciamos la GUI:

Como podemos ver en la siguiente figura se muestra la GUI una vez iniciada.

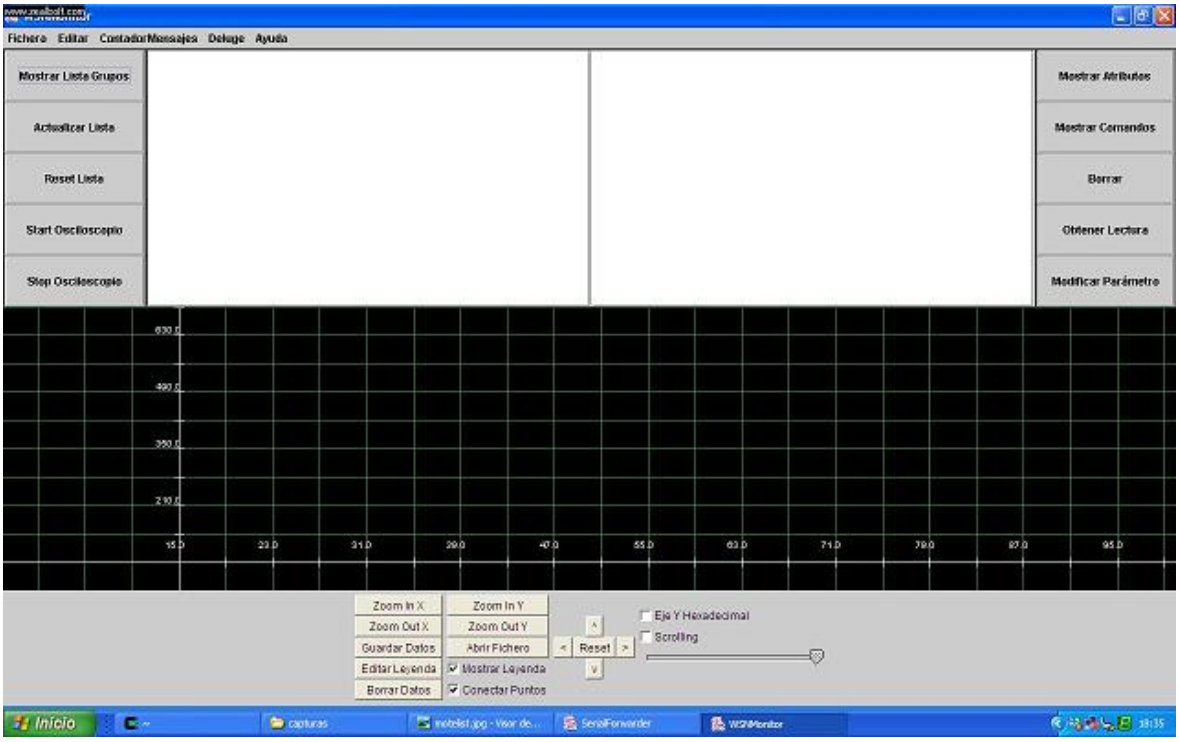


Figura 64. WSNMonitor

a continuación se enviará el comando ping a todos los nodos que estén activos, para poder detectar todos los nodos a los que se les puede realizar consultas.

Aplicación Desarrollada.

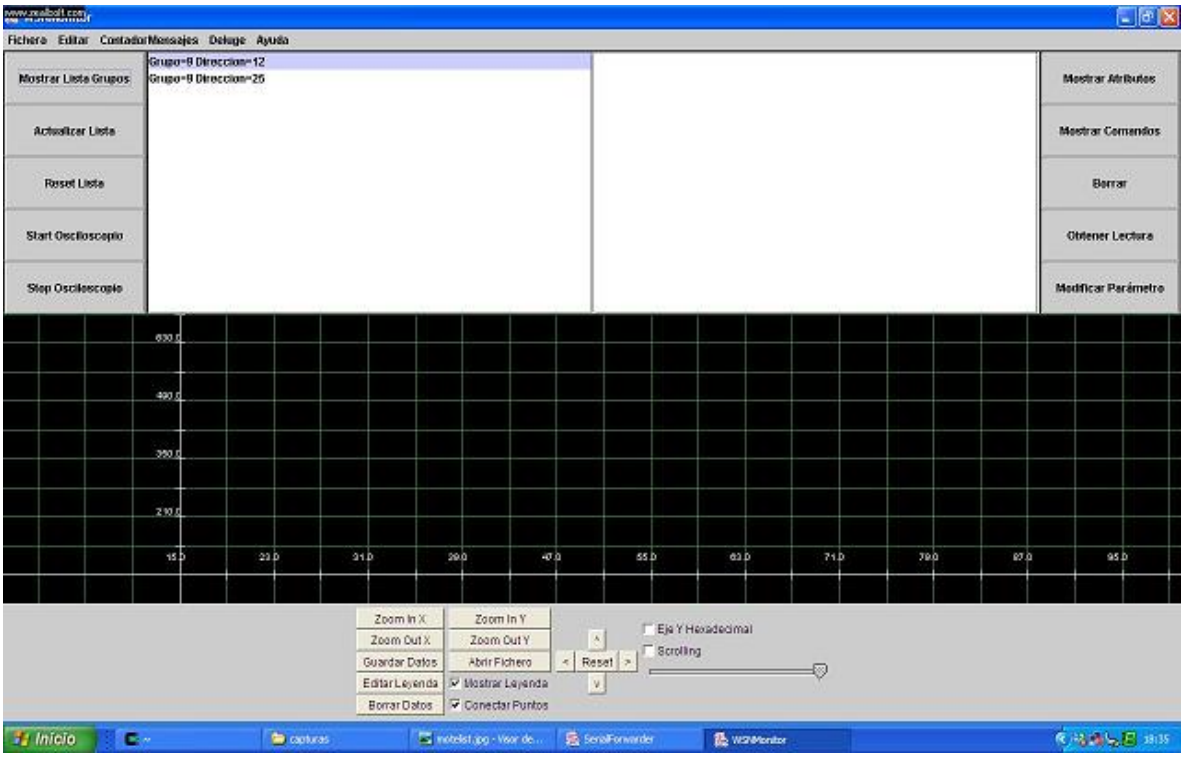


Figura 65. Nodos activos en la red de sensores

Como se puede apreciar, aparecen en la lista dos nodos pertenencientes al grupo 9 y con direcciones 12 y 25 respectivamente.

Si se quiere mostrar los comandos, tan sólo hay que seleccionar un nodo de la lista y pulsar el botón correspondiente, y el valor aparecerá en el centro de la pantalla.

Aplicación Desarrollada.

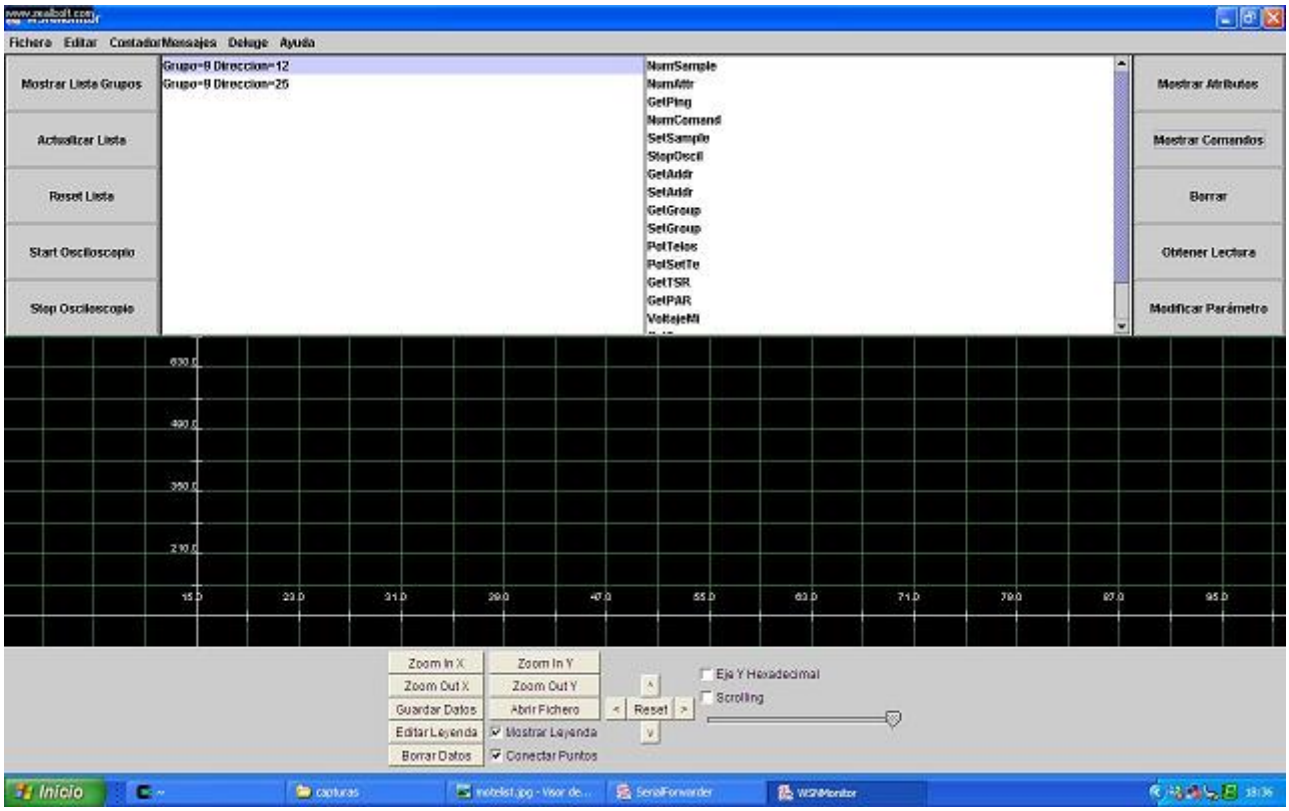


Figura 66. Comandos

El siguiente paso será realizar una consulta de alguna magnitud medible por el nodo sensor en el medio en el que se encuentra, para este ejemplo se realiza una lectura de la temperatura.

Aplicación Desarrollada.

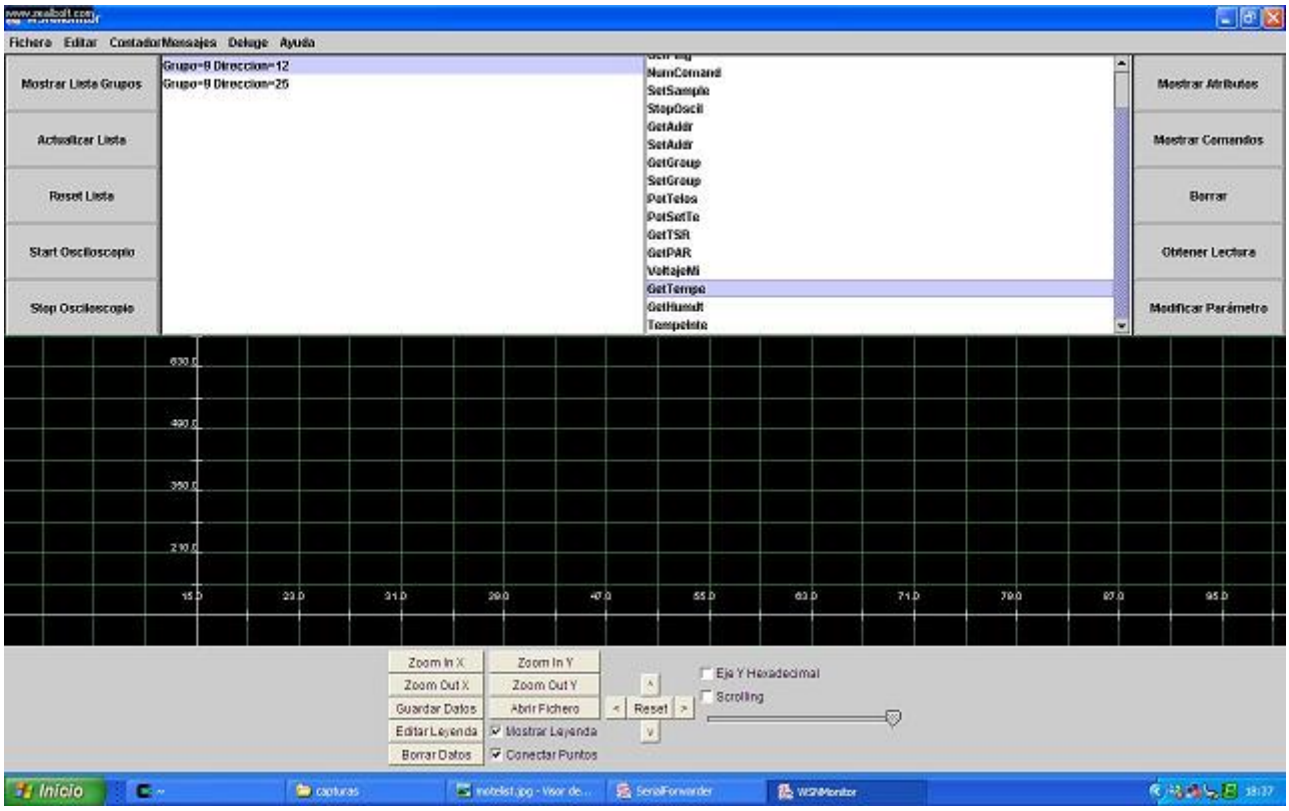


Figura 67. Seleccionar Comando

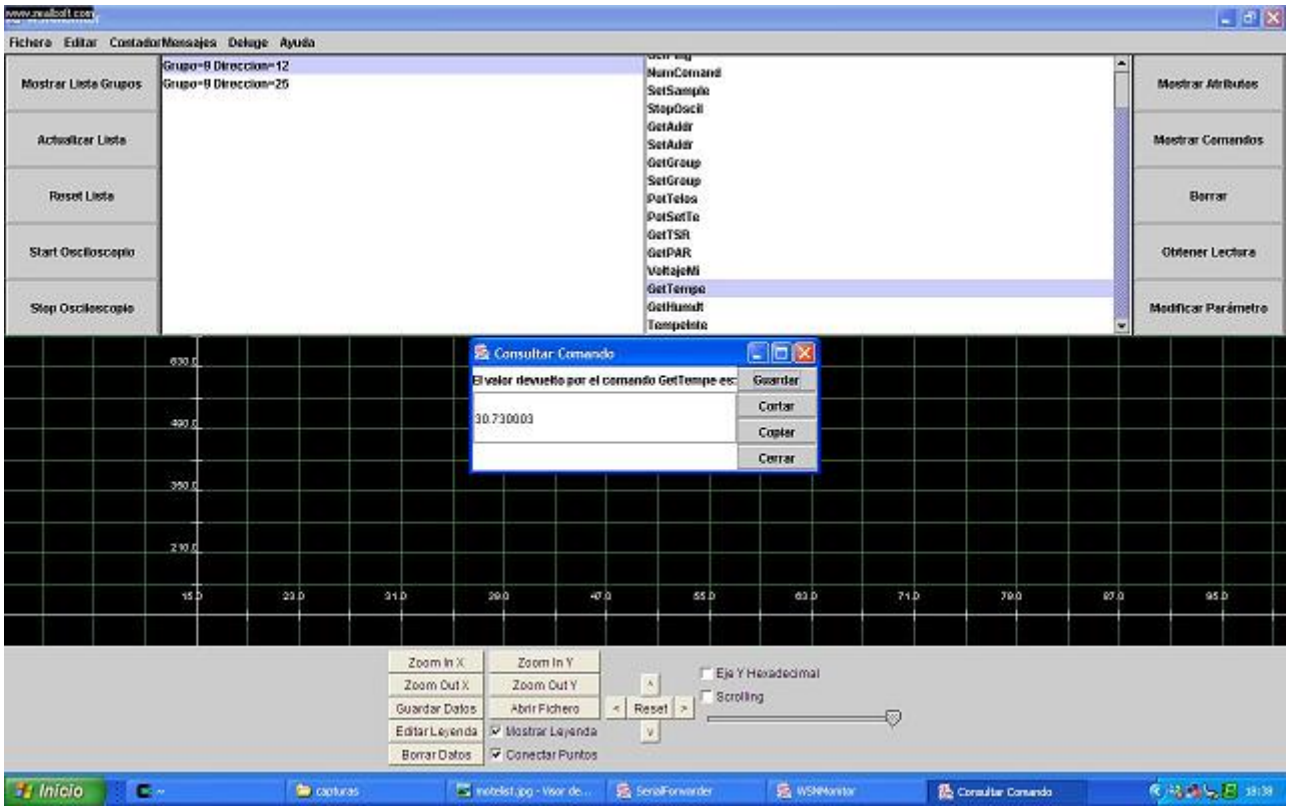


Figura 68. Valor de la Temperatura.

Como se puede ver en la imagen superior ,aparece el valor de la lectura de Comando GetTempe, el cual devuelve la temperatura en grados centígrados,(30,79ºC).

Una vez realizada una consulta como es muy similar a la modificación de un parámetro de un nodo sensor, directamente se mostrará la captura de modificar el grupo de uno de los nodos de la lista. Como se muestra en la imagen se aprecia que se va a cambiar el número del grupo del nodo 12 al grupo 4.

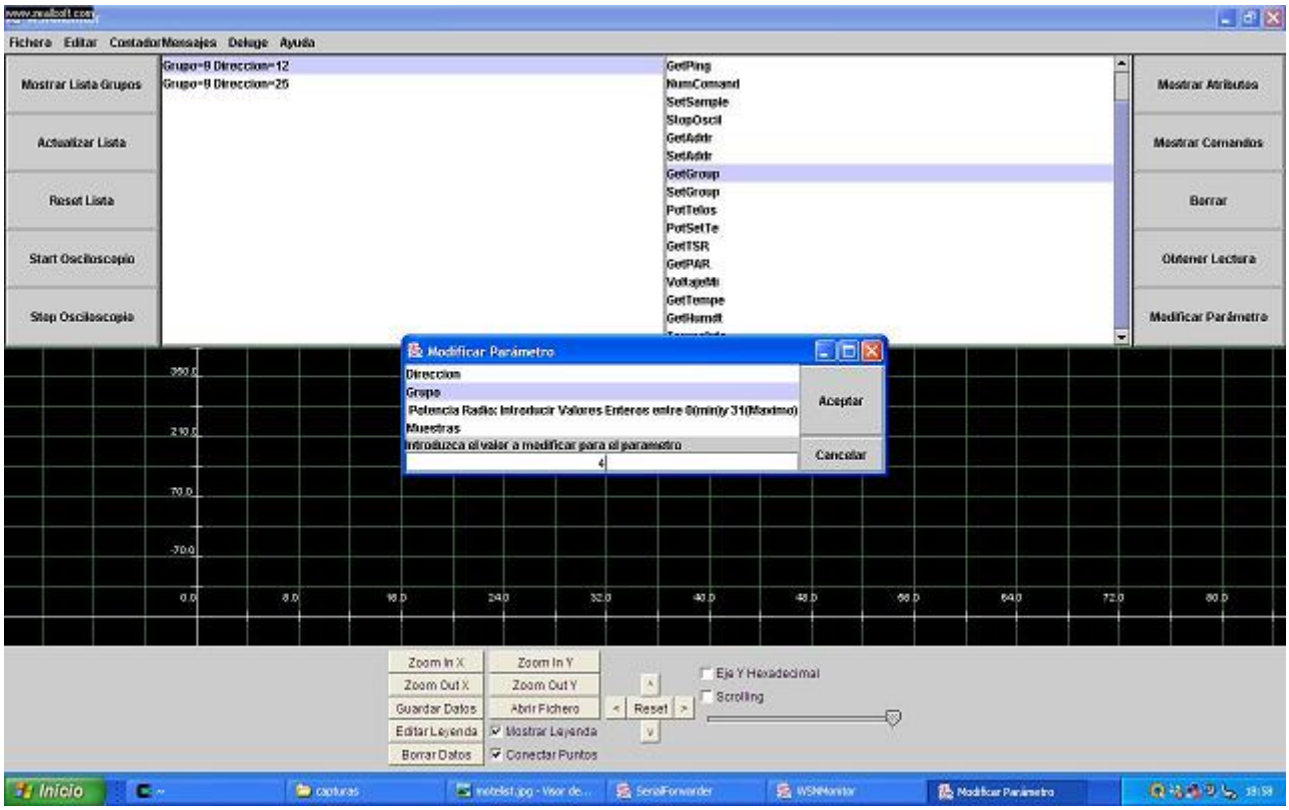


Figura 69. Modificar Grupo de red

Aplicación Desarrollada.

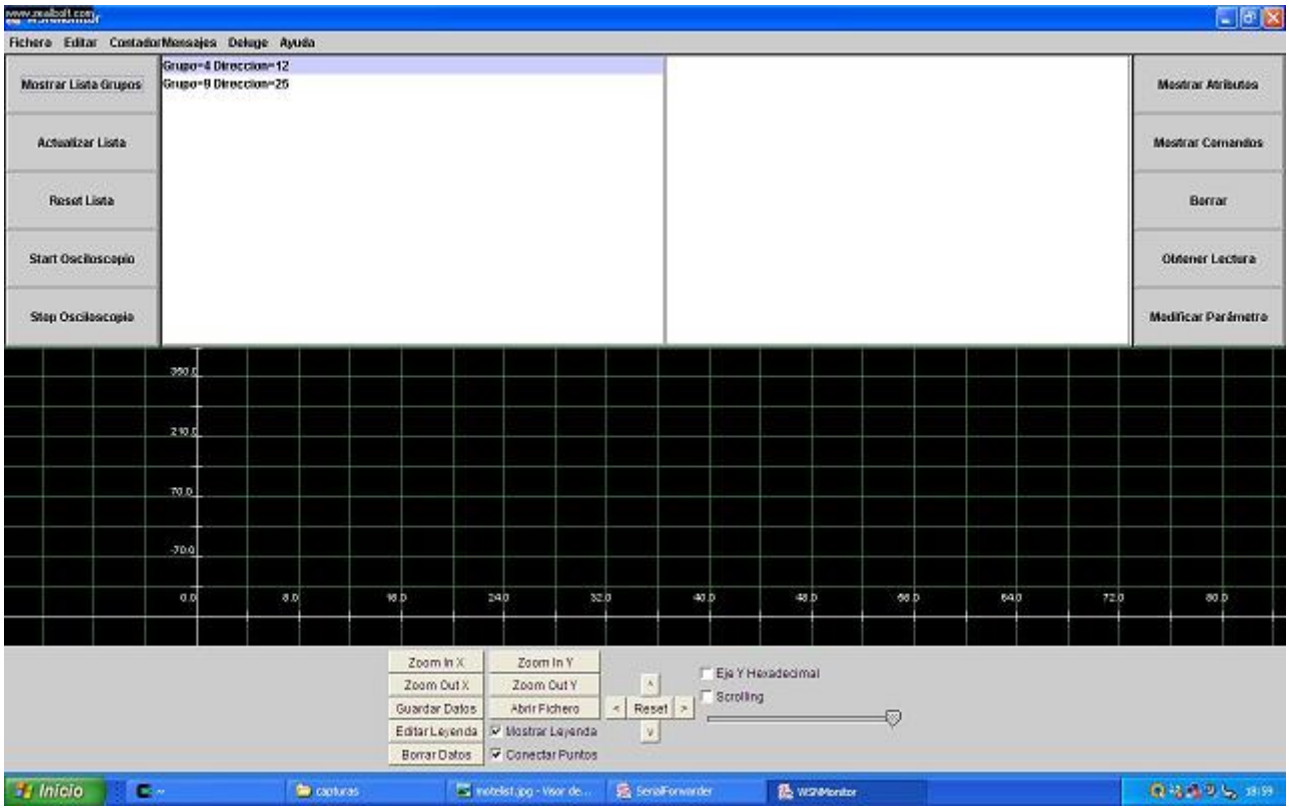
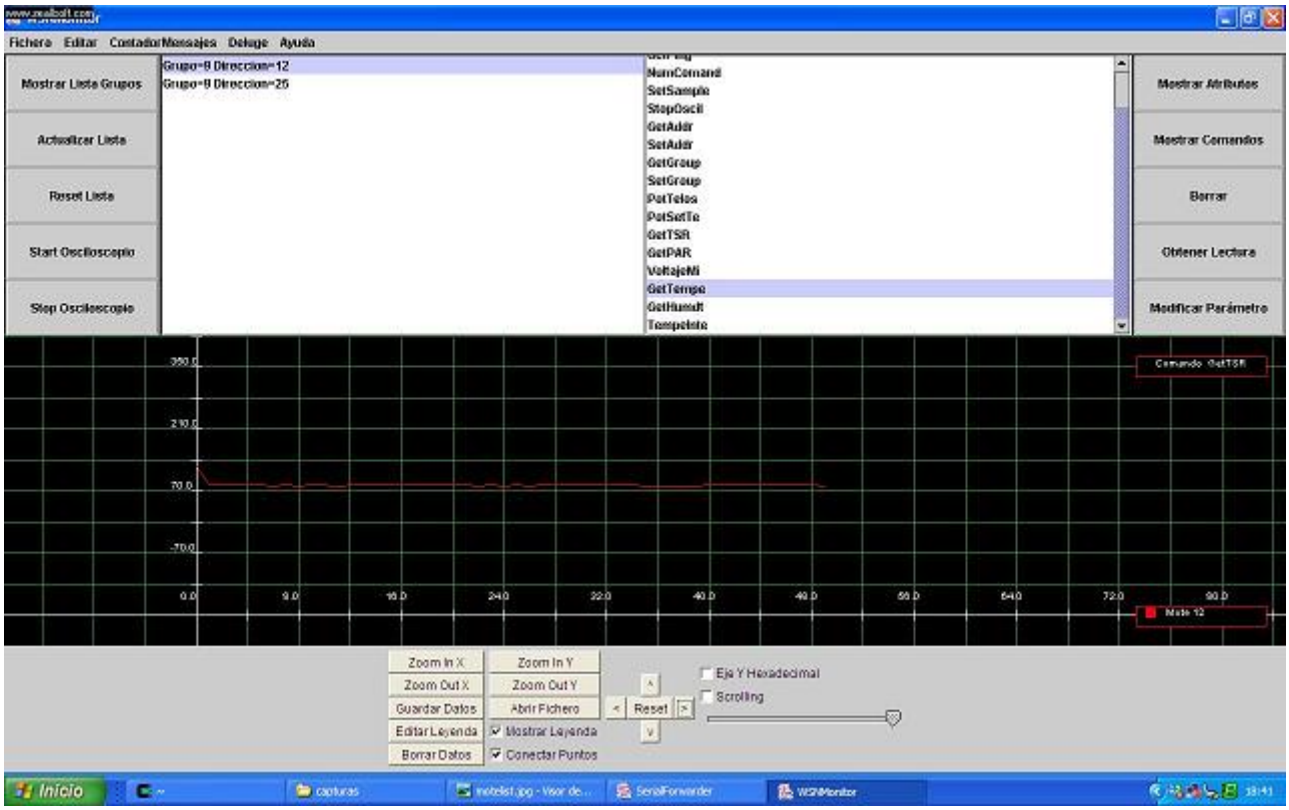


Figura 70.Grupo de Red modificado.

Ahora se dará paso a mostrar una representación gráfica de la magnitud luz solar en uno de los nodos, para ello sólo es necesario introducir el número de muestras, el comando a ejecutar, y haber seleccionado previamente un nodo de la lista.



Aplicación Desarrollada.

Figura 71.Seleccionar Comando.

Para terminar esta serie de ejemplos reales, se ejecutará un ping y la descarga de una imagen en un nodo sensor desde la GUI desarrollada.

Para poder usar la herramienta Deluge, hay que comenzar pinchando en el menu Deluge/ Abrir. Una vez hecho click en Abrir nos pedirá el grupo de red con el cual se va a trabajar.

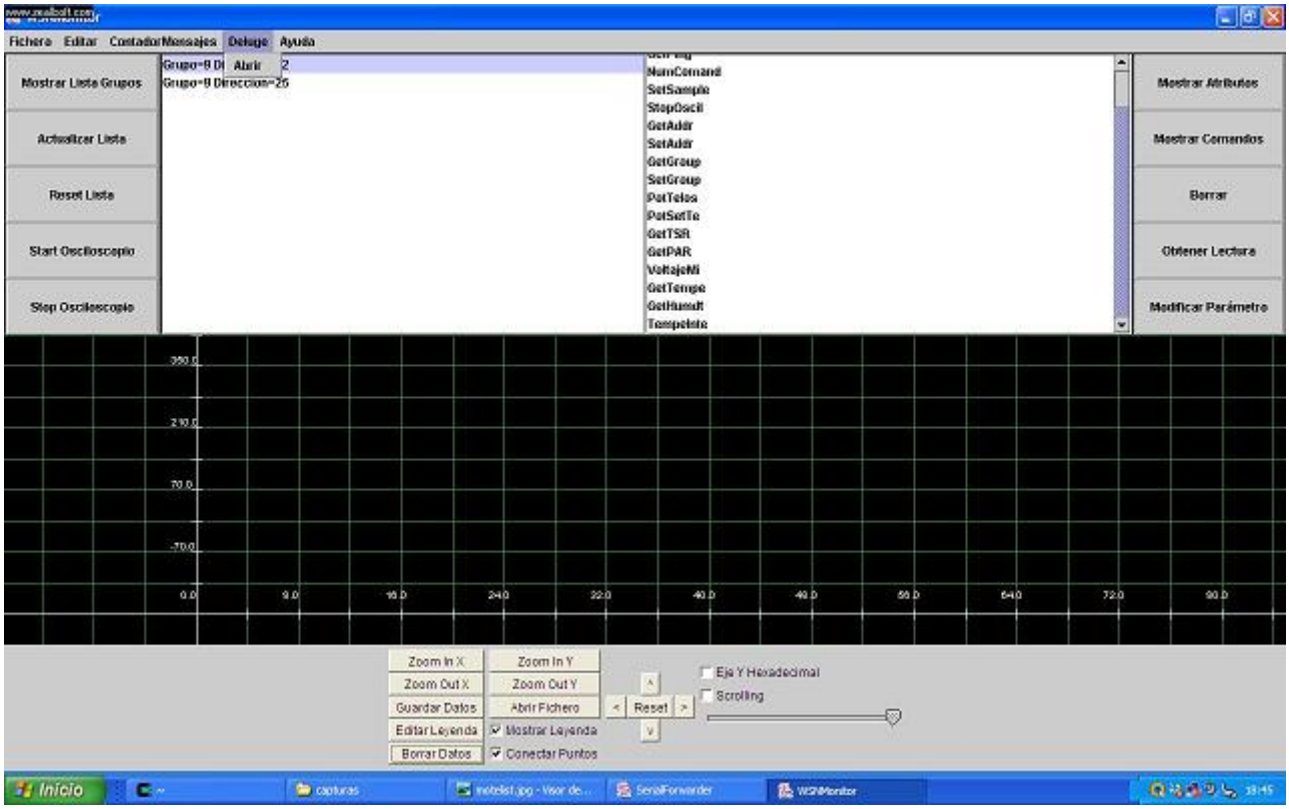


Figura 72. Abrir Deluge.

Como se muestra en la siguiente imagen, aparece la ventana abierta con el titulo Deluge y con el número del grupo correspondiente con el cual se va a reprogramar sus nodos, u obtener información, o cualquiera de las opciones que se pueden realizar desde estas ventana.

Aplicación Desarrollada.

Figura 73. Ventana Deluge

Siempre es recomendable hacer un Ping para obtener la informacion de sus imágenes tanto almacenadas como la que se está ejecutando, para ello sólo hay pinchar en unicast, elegir la dirección del nodo y pulsar el botón Ping, el cual al cabo de unos segundos mostrará en el área de texto toda la información relevante sobre las imágenes del nodo seleccionado. También se puede hacer un Ping a todo un grupo de red, pare ello sólo hay que marcar en el botón broadcast.

Aplicación Desarrollada.

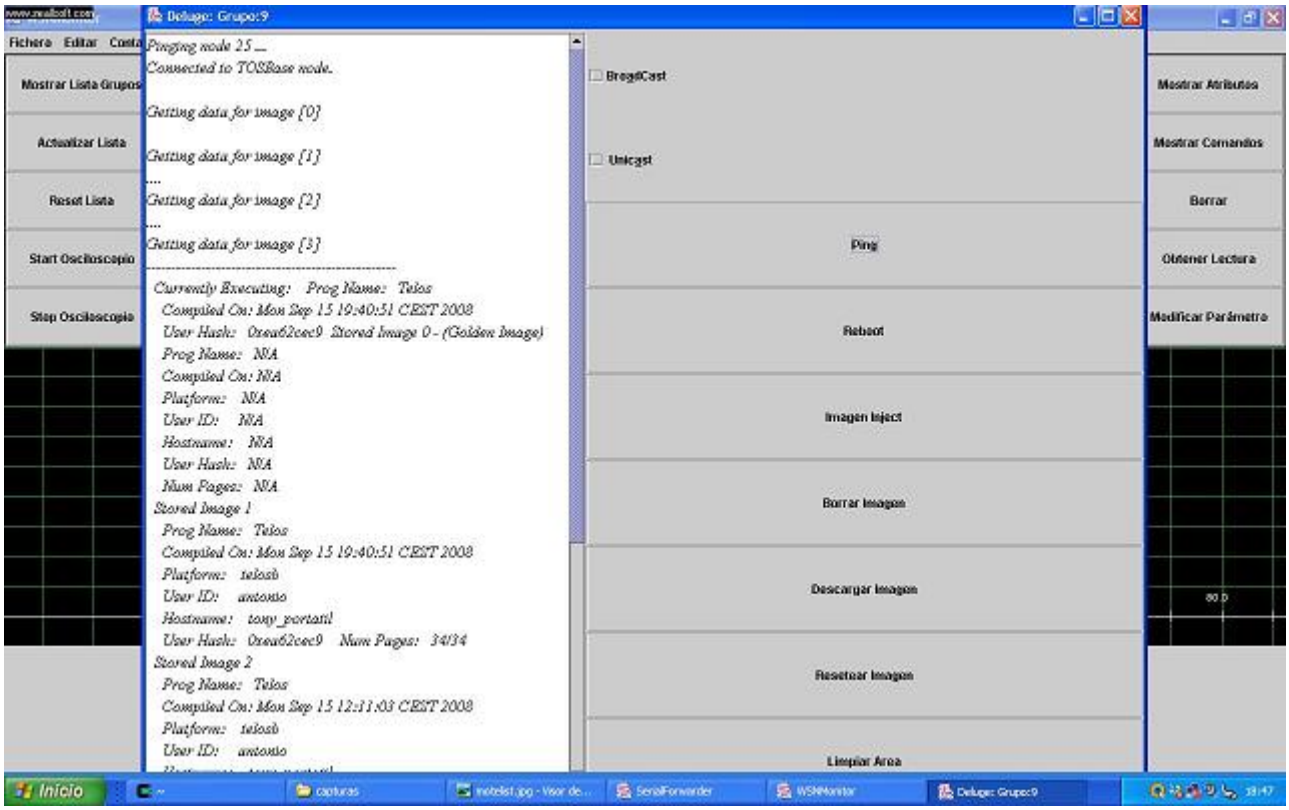


Figura 74. Ping Deluge

Una vez obtenida la información de todas las imágenes almacenadas en el nodo sensor con el que se está trabajando, ahora se va a realizar una descarga de la imagen número dos almacenada en el nodo sensor. Para ello tan sólo hay que pulsar en el botón Descargar Imagen, e introducir el número de imagen a descargar y el nombre con el cual se desee guardar la imagen en el PC donde se ejecute la GUI.

Aplicación Desarrollada.

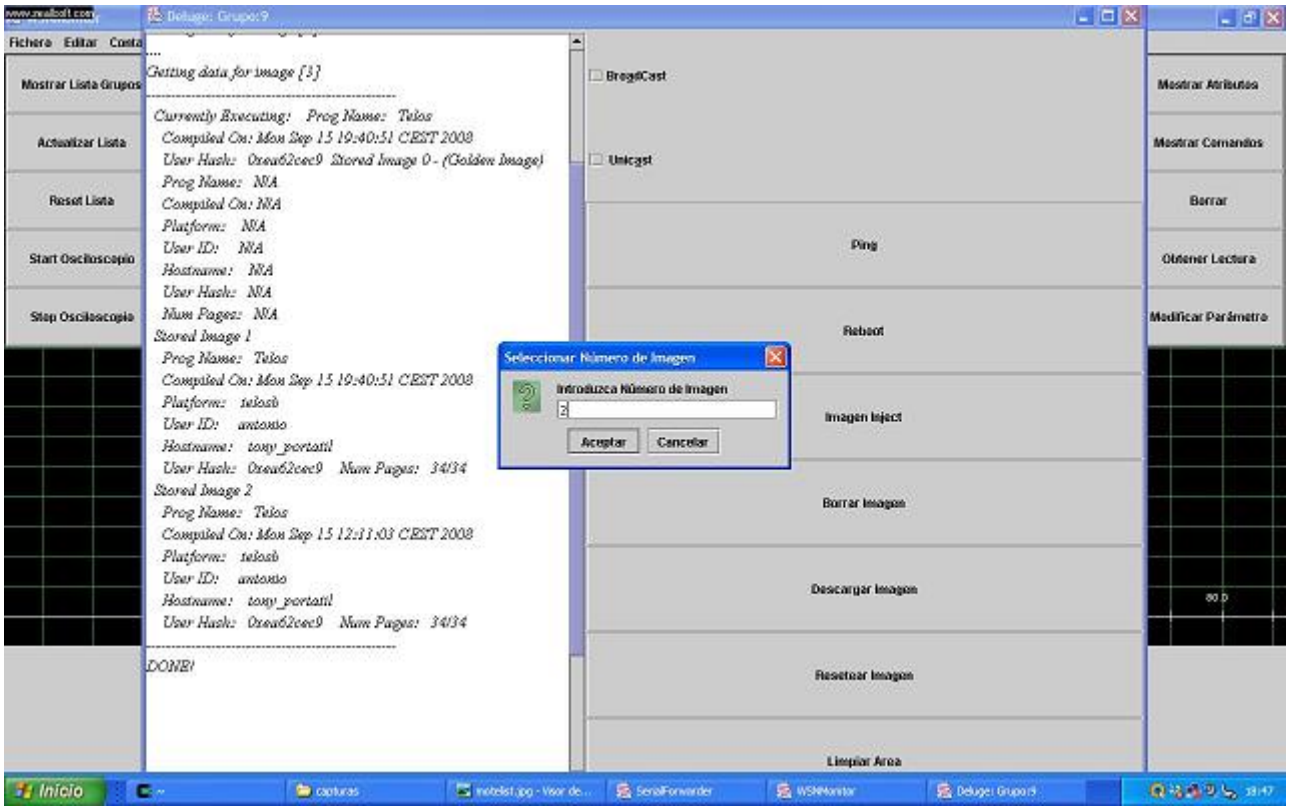


Figura 75. Selecciónar número de imagen.

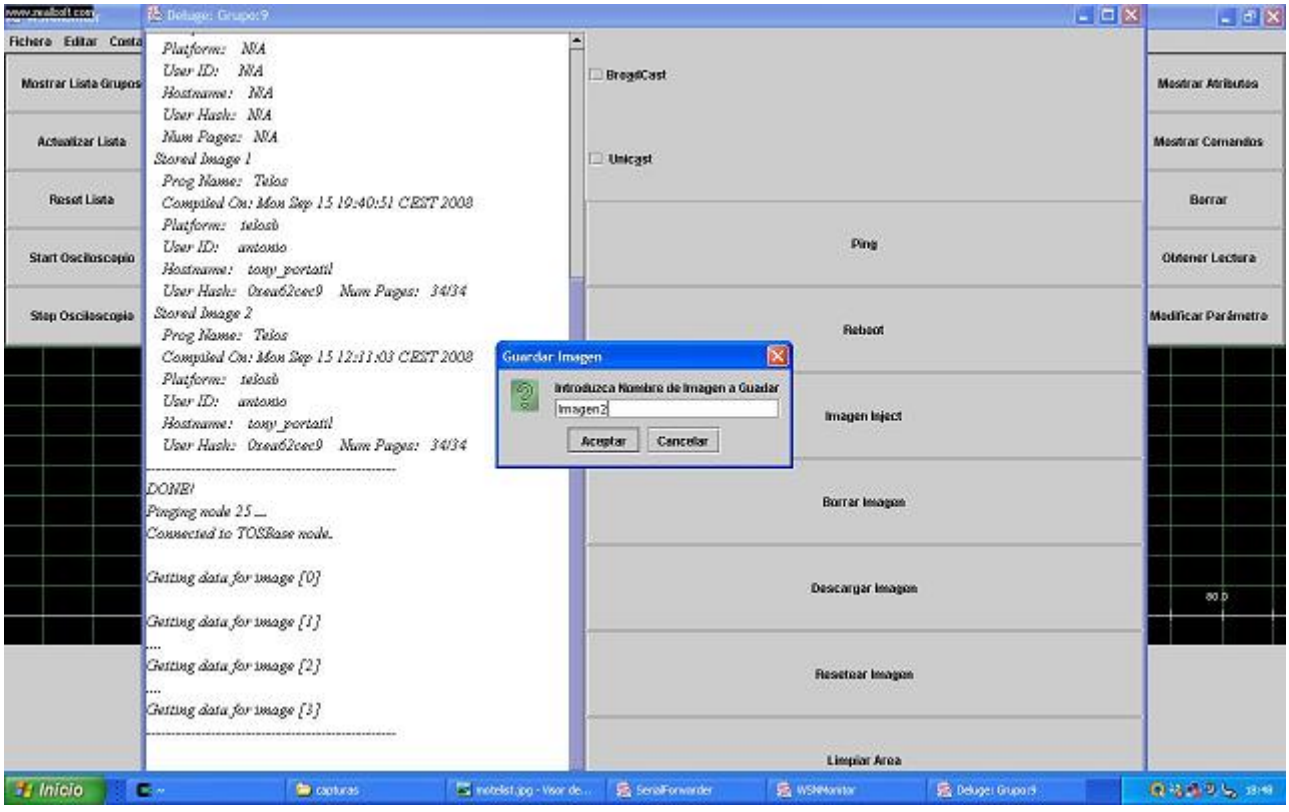


Figura 76. Seleccionar nombre de imagen

Aplicación Desarrollada.

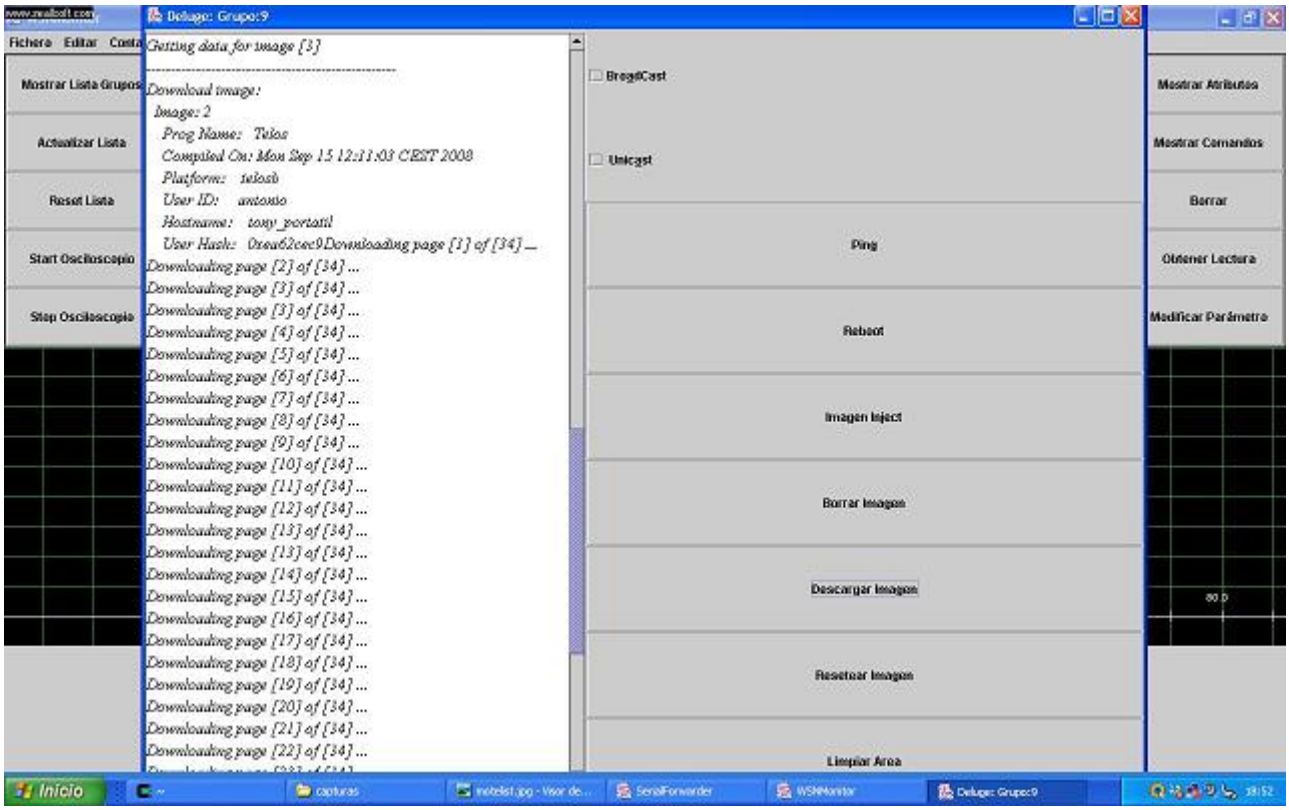


Figura 77. Descarga de imagen.

CONCLUSIONES Y LÍNEAS FUTURAS

4.1 Conclusiones:

Tras la finalización de este proyecto, se ha logrado una interfaz gráfica de usuario genérica para manejar redes de sensores inalámbricas así como una colección de componentes nesC, con los que se pueden añadir de forma sencilla a la aplicación funcionalidades encapsuladas en componentes aislados, los cuales pueden ser añadidos o eliminados de la aplicación. La mayor complejidad que se ha encontrado ha estado en el estudio y aplicación de TinySchema, ya que es parte fundamental en la programación de los nodos sensores.

Se ha conseguido poder trabajar con distintos grupos de red de forma simultanea e independiente, pudiendo realizar las funcionalidades básicas de una red de sensores como son la lectura de los distintos sensores que integran los nodos y la escritura de ciertos parámetros de funcionamiento de una red de sensores como pueden ser la dirección de un nodo, el grupo de red al cual pertenece un nodo, o la potencia de transmisión.

Con la GUI desarrollada se tiene un conocimiento de todos los sensores que están activos en una red de sensores, almacenar lecturas de los sensores de un nodo sensor en un fichero para su posterior procesamiento, representar gráficamente una secuencia de lecturas de cualquier sensor integrado en un nodo sensor a lo largo del tiempo.

Trabajar con medidas del sistema internacional, ha sido una ventaja para que los valores de las lecturas de los sensores puedan ser interpretados sin ninguna ambigüedad por cualquier usuario.

Con la incorporación de la librería Deluge en la GUI, se puede reprogramar de forma inalámbrica cualquier nodo de una red, además de ofrecer funcionalidades como reiniciar una imagen instalada en un sensor, descargar imágenes almacenadas en un nodo sensor, borrar imágenes almacenadas en un nodo sensor, y obtener toda la información de las imágenes disponibles en un nodo sensor.

Como última conclusión y no menos importante, comentar que debido a que la aplicación nesC es en definitiva un solo componente que ofrece una interfaz para su uso, se deduce que la aplicación desarrollada se puede incluir en cualquier otra aplicación nesC con la sencillez de tan sólo tener que unir la interfaz que ofrece el archivo de configuración Telos.NC.

4.2 Líneas Futuras:

Debido a la gran expansión que un futuro se dará en todos los campos, incluso en nuestra vida cotidiana., una vez acabado este proyecto se pueden abrir muchas líneas futuras como se citan a continuación:

- Optimizar el consumo de energía de los nodos sensores, pudiéndose abrir una línea para la optimización del enrutado y envío de mensajes en las redes de sensores inalámbricas.

-Al igual que en el resto de redes inalámbricas, el cifrado de las comunicaciones en las redes de sensores es muy importarte para la integridad y mantenimiento de los senosres y de la red.

- Incorporación de un comando para poder resetear un nodo cuando por algún motivo durante una toma de una lectura de un sensor o cualquier otro problema deja de funcionar de forma coherente y así pueda volver a su estado inicial y funcionar correctamente.

-Añadir funcionalidades como son poder localizar los vecinos más próximos de los nodos sensores, en definitiva, mostrar la topología de la red de sensores con la que se está trabajando.

-Crear GUIs a partir de archivos de configuración, con un nivel de personalización mucho mas avanzado.

BIBLIOGRAFÍA

- TinyOS Community Forum". An open-source OS for the networked sensor regime, 2007. www.tinyos.net/.

- Project". http://www.eecs.harvard.edu/~konrad/projects/motetrack/

- http://telegraph.cs.berkeley.edu/tinydb/tinyschema_doc/index.html.

- http://www.cs.berkeley.edu/~jwhui/deluge/documentation.html

- Fernando Losilla, Bárbara Álvarez, Pedro Sánchez Palma. "INTRODUCCIÓN A LAS REDES DE SENSORES. PERSPECTIVAS PARA LA INGENIERIA DEL SOFTWARE" Universidad Politécnica de Cartagena.

- I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, Wireless sensor networks: a survey, ComputerNetworks 38 (4) (2002) 393–422

- Berkeley WEBS". Wireless Embedded Systems, 2007.www.berkeley.edu/

- "Grupo de investigación nesC", 2007. http://nescc.sourceforge.net/

- http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/package-summary.html

- http://es.wikipedia.org/wiki/Red_de_sensores

- http://www.radioptica.com/Radio/wsn.asp

- Partridge, K. et al., "Fast intrabody signaling". Demonstration at Wireless and Mobile Computer Systems and Applications (WMCSA) (2000).

- http://w3.iec.csic.es/ursi/articulos_gandia_2005/articulos/SC4/563.pdf.

- Documentación disponible en el directorio /opt/doc/ en TinyOS.
- F. Losilla, "Estado del arte para redes de sensores y perspectivas desde el punto de vista de la ingenieria del software", Universidad Polit´ecnica de Cartagena, Octubre 2005.

- "Grupo de investigación nesC ", 2007. http://nesc.sourceforge.net

- "Crossbow technology Inc". Crossbow: Wireless Sensor Network, 2007. http://www.xbow.com/

- “Zigbee Alliance”. Wireless control that simply works, 2007.
http://www.zigbee.org/

- “MoteIV ”. A leading provider of wireless sensor networking solutions, 2007.
http://www.moteiv.com/

- “Making sense of Sensor Networks”. Crossroads - The ACM Student Magazine, 2007.
http://www.acm.org/crossroads/xrds9-4/sensornetworks.html