Departamento de Informática

PROYECTO FIN DE CARRERA

Diseño e implementación de un módulo de gestión de energía en el simulador Avrora

Autor: Hugo Romo Otero

Tutor: Mª Soledad Escolar Díaz

Leganés, Mayo de 2013

Título: Diseño e implementación de un módulo de gestión de energía en el simulador Avrora

Autor: Hugo Romo Otero

Director: Mª Soledad Escolar Díaz

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

“Nada es permanente a excepción del cambio”

Heráclito

Agradecimientos

A mi familia y amigos por darme todo su apoyo y confianza cuando lo he necesitado.

A Soledad por guiarme y ayudarme durante la desarrollo de todo el proyecto.

Abstract

This final degree work focuses on the development of a new energy management module, which is integrated in a simulator, called Avrora, for Wireless Sensor Networks (WSN).

The WSN are distributed systems composed of a set of independent, autonomous, and robust nodes with the capability of sampling the environment by means of sensors and transducers and that coordinate their actions to achieve an overall goal.

Avrora is one of the most used simulators of WSN. This simulator provides several monitors to manage and control different aspects of the simulation, such as the energy consumed and the packets transmitted/received by each node. The goal of this project is to add to this platform a new functionality that consist in a battery monitor, which allows the users to know at any time the battery status of each node and the residual battery at the end of the simulation.

The new monitor developed is based on the use of a graphical interface that shows the battery level of all nodes involved in the simulation. In addition, the system should provide to the user, once the simulation have finished, a file including the results computed.

The graphics interface of the monitor will show a set of predefined images to represent the state of the batteries of each node as well as the corresponding numerical values. In this interface is defined that it can show a maximum of 512 nodes with the information of the energy of the nodes.

Resumen

Este trabajo de fin de grado se centra en el desarrollo de un nuevo módulo de gestión de energía que se integra con un simulador para redes de sensores inalámbricas denominado Avrora.

Las redes de sensores inalámbricas son sistemas distribuidos formados por un conjunto de nodos independientes, autónomos, robustos, que son capaces de muestrear el entorno a través de sensores o transductores y coordinan sus acciones para alcanzar un objetivo global.

Avrora es uno de los simuladores más usados de redes de sensores inalámbricas. Este simulador dispone de una serie de monitores para gestionar y controlar distintos aspectos de las simulaciones que se realizan (como por ejemplo la energía consumida y los paquetes transmitidos/recibidos por cada nodo). El objetivo de este proyecto es añadir a Avrora una funcionalidad inexistente hasta el momento, y que consiste en implementar un nuevo monitor que permita al usuario conocer en cualquier momento de la simulación el estado de las baterías de cada nodo y la batería residual al final de la simulación.

El nuevo monitor se basa en la utilización de una interfaz gráfica en el que se muestra el nivel de batería de todos los nodos presentes en la simulación. Además, el sistema deberá proporcionar al usuario de manera textual dicha información una vez acabada la simulación mediante un log de resultados.

En la interfaz gráfica el sistema mostrará, mediante una serie de imágenes predefinidas para representar el estado de las baterías de cada nodo así como los valores numéricos asociados. En dicha interfaz se contempla que se pueda mostrar hasta 512 nodos con la información correspondiente al gasto de energía de sus baterías.

Contenido

Agradecimientos 5
Abstract 7
Resumen 8
Contenido 9
Índice de figuras 11
Índice de tablas 12
1. Introducción 13
1.1 Descripción del problema 13
1.2 Motivación 14
1.3 Objetivos 14
1.4 Terminología y conceptos básicos 15
2. Estado de la cuestión 17
2.1. Redes de Sensores 17
2.1.1. Aplicaciones 17
2.1.2. Arquitectura 19
2.1.2.1. Nodos sensores 20
2.1.2.2. Placa de sensores 21
2.1.2.3 Gateway 22
2.1.2.4 Estación base 23
2.2. TinyOS 23
2.2.1 nesC 24
2.3. Java 25
2.3.1 Java Swing 26
2.4. Simuladores para redes de sensores 27
2.4. 1. Avrora 27
2.4. 2. TOSSIM 30
2.4. 3. Castalia 31
2.5. Fuentes de energía 31
2.5.1. Baterías 31
2.5.2. Fuentes alternativas 33

3. Análisis 35
3.1 Requisitos funcionales 35
3.2 Requisitos de apariencia 39
3.3 Requisitos de rendimiento 39
3.4 Requisitos de Sistema 40
3.5 Casos de uso 41
4. Modelo del Sistema 43
5. Diseño 49
5.1 Diseño de la interfaz 49
5.2 Diseño de clases 54
5.3 Integración con Avrora 57
6. Implementación 59
6.1 BatteryMonitor 59
6.1.1 BatteryCheck 60
6.1.2 Logger 62
6.1.3 Monitor 64
6.2 BatteryMonitorPanel 66
7. Evaluación 69
7.1 Validación de la interfaz 69
7.2 Validación del funcionamiento del sistema 72
7.3 Limitaciones 79
8. Conclusiones 81
8.1 Revisión de los objetivos 81
8.2 Líneas futuras de trabajo 81
8.3 Consideraciones legales 82
8.3 Planificación 82
8.4 Presupuesto 83
8.4.1 Recursos humanos 83
8.4.2 Recursos materiales 84
8.4.3 Otros costes directos 84
8.4.3 Coste total 85
8.5 Valoración personal 85
Bibliografía 86

Índice de figuras

Figura 1: Redes de sensores en una frontera 18
Figura 2: Control de incendios 18
Figura 3: Monitorización de pacientes 19
Figura 4: Red de sensores 20
Figura 5: Nodo sensor MicaZ 21
Figura 6: Placa de sensores 22
Figura 7: Gateway MIB520, MIB600 y MIB510, respectivamente 22
Figura 8: TinyOS 23
Figura 9: Java 26
Figura 10: Jerarquía de Swing 27
Figura 11: Pila alcalina 32
Figura 12: Placas fotovoltaicas 33
Figura 13: Efecto fotoeléctrico 33
Figura 14: Motes Eko 34
Figura 15: Casos de uso 41
Figura 16: Diseño inicial de la interfaz 49
Figura 17: Segundo prototipo de diseño de la interfaz gráfica 50
Figura 18: Tercer prototipo de diseño de de la interfaz gráfica 51
Figura 19: Gateway 52
Figura 20: Nivel de baterías 52
Figura 21: Ejemplo del escenario 54
Figura 22: Diagrama de clases 55
Figura 23: Diagrama de flujo de BatteryCheck 56
Figura 24: Integración con Avrora 58
Figura 25: Ventana de 32 nodos 69
Figura 26: Ventana de 512 nodos 70
Figura 27: Ventana de 600 nodos 70
Figura 28: Primera transición 71
Figura 29: Segunda transición 71
Figura 30: Estado final 72
Figura 31: Gráfico de dispersión inicial 74
Figura 32: Grafico de dispersión tras 4 horas 75
Figura 33: Grafico de dispersión final 76
Figura 34: Comparativa del consumo 77
Figura 35: Topología con 100 nodos 78
Figura 36: Diagrama de Gantt 83

Índice de tablas

Tabla 1: UR-F001 36
Tabla 2: UR-F002 36
Tabla 3: UR-F003 36
Tabla 4: UR-F004 36
Tabla 5: UR-F005 37
Tabla 6: UR-F006 37
Tabla 7: UR-F007 37
Tabla 8: UR-F008 37
Tabla 9: UR-F009 38
Tabla 10: UR-F010 38
Tabla 11: UR-F011 38
Tabla 12: UR-F012 38
Tabla 13: UR-F013 39
Tabla 14: UR-F013 39
Tabla 15: UR-A001 39
Tabla 16: UR-A002 39
Tabla 17: UR-R001 40
Tabla 18: UR-S001 40
Tabla 19: UR-S002 40
Tabla 20: UR-S003 40
Tabla 21: CU-01 41
Tabla 22: CU-02 42
Tabla 23: CU-03 42
Tabla 24: CU-04 42
Tabla 25: Componentes Hardware 43
Tabla 26: Componentes hardware y estados 44
Tabla 27: Componentes, estados y consumos 45
Tabla 28: Resultado ejecución 47
Tabla 29: Archivo de topología 50
Tabla 30: Tamaños de la ventana 67
Tabla 31: Parámetros de las pruebas 73
Tabla 32: Energía consumida en 1 hora 73
Tabla 33: Energía consumida en 2 horas 74
Tabla 34: Energía consumida en 4 horas 75
Tabla 35: Tiempos 76
Tabla 36: Simulación con 100 nodos 78
Tabla 37: Recursos humanos 84
Tabla 38: Recursos materiales 84
Tabla 39: Otros costes directos 85
Tabla 40: Coste total 85

1. Introducción

En este capítulo se presenta una breve introducción al proyecto desarrollado. Inicialmente, se realiza una breve descripción del problema y de la motivación que ha originado este proyecto fin de carrera. A continuación se describen los objetivos que se pretenden conseguir. Finalmente, se presenta una lista de abreviaturas y términos comúnmente usados en este documento.

1.1 Descripción del problema

En la actualidad se encuentran en auge el desarrollo e implantación de redes de sensores inalámbricas, debido al aumento de sus usos y aplicaciones en diferentes campos como pueden ser la medicina, domótica o el ámbito militar.

Las redes de sensores inalámbricas utilizan conjuntos de nodos (o motes) interconectados entre sí de forma inalámbrica para, mediante los sistemas de sensores de que disponen, realizar tareas concretas dependiendo de cuál sea el ámbito para el que están diseñados. Al trabajar los nodos de forma autónoma, es necesario que antes del despliegue de la red, se hayan realizado las comprobaciones necesarias para asegurar que la aplicación que van a ejecutar los sensores realiza sus funciones de la manera adecuada y sus resultados son los esperados. Esto implica que se haga necesaria la utilización de simuladores de redes de sensores.

El uso de este tipo de simuladores facilita el trabajo de los desarrolladores para la creación de nuevas aplicaciones y sistemas para este tipo de redes. Estos simuladores presentan una serie de ventajas que facilitan el desarrollo y la evaluación de los nuevos programas que se desarrollan para estas redes, ya que permiten monitorizar de manera constante el funcionamiento de dichos sistemas, y realizar los cambios o comprobaciones necesarias para el perfecto funcionamiento del sistema. Un ejemplo de simulador de redes de sensores es la plataforma Avrora, sobre la que se desarrolla este proyecto.

Pese a todo esto, los simuladores de redes de sensores también implican algunos inconvenientes, como puede ser el que ha condicionado el desarrollo de este proyecto. Dicho inconveniente se basa en que durante una simulación de Avrora el usuario no dispone de ningún método para conocer cuál es el nivel de energía de las baterías de los nodos en cualquier instante de la simulación. Es más, si un nodo agota su batería en un instante determinado de la simulación, el desarrollador no podrá conocer que dicho nodo está inactivo a partir de ese momento, y por tanto, no continuará su ejecución normal.

1.2 Motivación

Este proyecto se plantea como un método para ampliar las funcionalidades de uno de esos simuladores de redes de sensores, en este caso Avrora. El proyecto se centra en este simulador debido a que es uno de los sistemas más populares para este tipo de redes, ya que dispone de gran cantidad de monitores y funcionalidades que facilitan la simulación de las aplicaciones.

Antes de continuar es necesario comentar la importancia de la energía en las redes de sensores inalámbricas. En este tipo de redes es muy importante la gestión eficiente de la energía de la que disponen los nodos dado a que, en general, los nodos utilizan fuentes de alimentación de capacidad limitada (como por ejemplo un par de baterías convencionales) y por tanto, el consumo energético de los nodos debe ser controlado. Para ello, existen numerosas soluciones planteadas para alargar la vida de los nodos lo máximo posible, tanto de tipo hardware como software. Algunas soluciones basadas en hardware, permiten llevar a los distintos componentes físicos a un estado de bajo consumo, mientras no se estén utilizando; algunas soluciones basadas en software permiten hacer un uso más eficiente de los recursos físicos. Sea cuál sea el carácter de las soluciones de ahorro de energía, los simuladores nos van a permiten depurar y evaluar el comportamiento de las aplicaciones escritas para redes de sensores. Así se ha decidido ampliar las funcionalidades del simulador Avrora con la adición de un nuevo monitor que facilite la tarea, a los usuarios de esta plataforma, a la hora de probar sus aplicaciones. Esto se debe a la necesidad de los usuarios de poder conocer el gasto de energía de los sensores en tiempo real durante las simulaciones, ya que con anterioridad no se disponía de ningún medio para conocer dicha información.

Hasta el momento de comenzar con el desarrollo de este proyecto, solo se podía conocer el gasto total de energía una vez finalizadas las simulaciones, pero en ningún caso durante el tiempo de la simulación.

1.3 Objetivos

El objetivo principal de este proyecto es el diseño e implementación de un nuevo monitor que extienda las funcionalidades de Avrora y solvente el problema antes descrito. Este monitor se utilizará para mostrar al usuario la energía disponible en cada momento en las baterías de los nodos utilizados en la simulación.

Para realizar dicha función el monitor generará una interfaz gráfica en la que el usuario pueda localizar de manera rápida y sencilla la información relativa al consumo de energía por parte de los nodos. Además el monitor deberá mostrar al usuario la energía residual de las baterías una vez haya finalizado la simulación.

El nuevo monitor de batería además deberá ser capaz de generar un fichero de resultados con la información del consumo de energía durante la simulación. Este log deberá incluir información relativa a los tiempos de simulación, la energía consumida y la energía residual.

A continuación se enumeran algunos de los sub-objetivos que se pretenden conseguir durante el desarrollo del sistema:

1. Estudiar el funcionamiento de Avrora.
2. Estudiar el modelo de energía de Avrora.
3. Estudio del diseño del interfaz.
4. Integrar el nuevo monitor con la plataforma ya existente.
5. Generar un fichero con los resultados de la simulación.
6. Estudio de los resultados obtenidos de las simulaciones.

1.4 Terminología y conceptos básicos

En este apartado se incluyen todos los términos y conceptos más usados en la memoria y que a su vez pueden resultar confusos.

A

Avrora

Simulador de redes de sensores de código libre. Dispone de un gran número de módulos y parámetros con los que configurar las simulaciones para adaptarse al usuario.

B

Batería

Fuente de energía química de los nodos sensores, normalmente pilas alcalinas o de ion-litio.

D

Domótica

Conjunto de sistemas que automatizan las diferentes instalaciones de una vivienda (1).

E

Estación Base

Elemento de las redes de sensores, normalmente un PC u otro equipo de potencia similar, encargado de almacenar y tratar la información recogida y transmitida por la red de sensores.

G

Gateway

Nodo perteneciente a una red de sensores que actúa como nexo de unión entre la red y la estación base, a la que normalmente se encuentra unido por un cable.

J

Java

Lenguaje de programación orientado a objetos perteneciente a Oracle pero diseñado originalmente por Sun Microsystems. Este lenguaje procede de C y C++ pero con algunas simplificaciones a de bajo nivel.

M

Monitor

Complemento de la plataforma de simulación Avrora que amplía las funcionalidades originales de ésta. Hay diferentes tipos de monitores que realizan acciones específicas como puede ser la transmisión de paquetes (PacketMonitor) o el control del gasto de energía (EnergyMonitor).

Mote

Cada uno de los nodos de una red de sensores, que disponen de unas características hardware limitadas y es capaz de cooperar con otros nodos.

S

Sensor

Dispositivo que detecta una determinada acción externa, temperatura, presión, etc., y la transmite adecuadamente (1).

T

TinyOS

Sistema operativo de código libre desarrollado en la Universidad de Berkeley. Esta especializado en dispositivos con capacidades hardware limitadas como los dispositivos empotrados. Está escrito en nesC.

W

WSN

Las Redes de Sensores Inalámbricas (Wireless Sensor Networks) son sistemas distribuidos compuestos por multitud de nodos sensores o motes que interaccionan entre sí para realizar una tarea conjunta.

2. Estado de la cuestión

2.1. Redes de Sensores

Una red de sensores está compuesta por gran cantidad de nodos sensores interconectados entre sí de forma inalámbrica o cableada, cuyas principales funciones consisten en la recolección, transmisión y procesamiento de datos e información. Este proyecto fin de carrera se centra en las redes de sensores *inalámbricas*, del inglés Wireless Sensor Networks (WSN), es decir, aquellas redes cuyos nodos sensores comunican de manera inalámbrica sin necesitar una infraestructura de red física que la soporte.

Las redes de sensores inalámbricas se constituyen de tres elementos básicos. Una red de sensores se compone de dispositivos denominados *nodos sensores* o *motes*, distribuidos sobre una determinada superficie, interconectados entre sí, y a su vez con un nodo especial denominado *Gateway*, encargado de comunicarse con una *estación base*. La estructura y funcionamiento más detallado de cada uno de estos elementos se verá en el apartado de Arquitectura.

Las redes de sensores inalámbricas inicialmente fueron aplicadas en el ámbito militar, pero sus usos se están extendiendo rápidamente a otro gran número de sectores, los cuales vemos a continuación.

2.1.1. Aplicaciones

Los posibles usos y aplicaciones (2) de las redes de sensores son muy amplios debido a sus características especiales. Algunos de esas aplicaciones son:

- **Militares**: Una de las iniciales y principales aplicaciones cuyo uso está más extendido, es el uso de las redes de sensores en el ámbito militar. Como por ejemplo, detección de ataque químico, biológico y nuclear. Un ejemplo práctico de uso militar de las WSN es en el control de las fronteras entre países o zonas de conflicto, como por ejemplo en la frontera entre Corea del Norte y Corea del Sur. Este tipo de sistemas utiliza sensores de movimiento, magnéticos, de presión, sonoros, etc. La Figura 1 representa gráficamente un ejemplo de este tipo de aplicaciones militares.

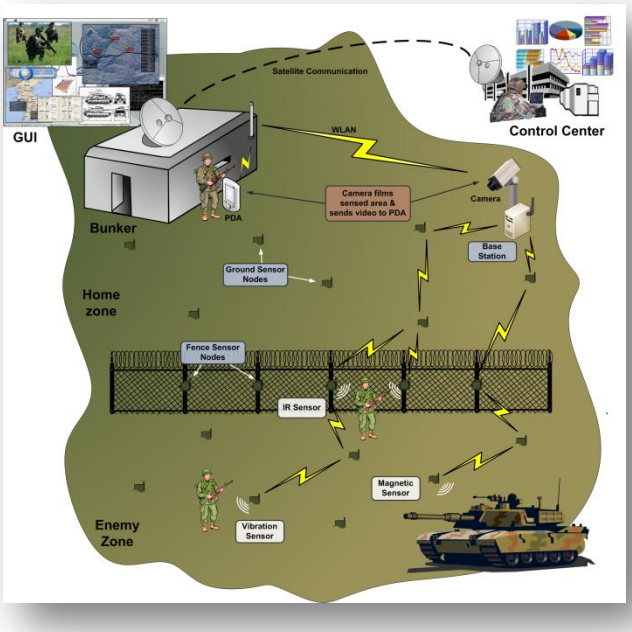

Figura 1: Redes de sensores en una frontera

- **Medioambientales**: en el ámbito medioambiental el uso de este tipo de dispositivos se centra en el control y detección de peligros y desastres naturales, como pueden ser incendios, terremotos o tsunamis. Además también existen otro tipo de aplicaciones más tradicionales como pueden ser la monitorización del medioambiente y de los cultivos agrícolas (3). Un ejemplo del uso de redes de sensores en este ámbito es el despliegue de la red en un incendio mediante aviones, para poder construir un mapa de temperaturas para así estudiar su comportamiento. La Figura 2 representa como se pueden distribuir los nodos en un incendio para controlar su estado y su desarrollo.

Figura 2: Control de incendios

- **Médicas**: Actualmente también se están implementando las redes de sensores para ser usadas en el campo de la medicina, y uno de sus principales usos es la monitorización remota de pacientes y en sistemas de administración de medicamentos mediante control externo. La Figura 3 ilustra cómo se pueden monitorizar a los pacientes en los hospitales o en su vida cotidiana, para así mantener un control constante de su estado físico.

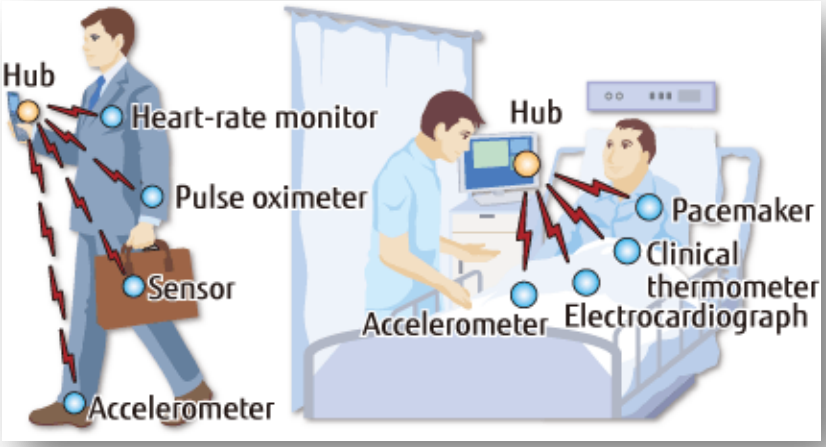

Figura 3: Monitorización de pacientes

- **Domótica**: en el ámbito doméstico el uso de las redes de sensores esta principalmente centrado en la domótica. Por ejemplo, en el desarrollo de entornos inteligentes (*smart environments)* (4), en los que los sistemas de la casa se adaptan a las necesidades de los usuarios mediante la recolección de datos a través de nodos sensores.
- Otras aplicaciones comerciales: control de vehículos, museos interactivos, etc.

Estos son solo algunos de los ejemplos de las aplicaciones actuales de las redes de sensores, ya que el crecimiento en el uso de este tipo de tecnologías va generando nuevas aplicaciones de manera constante.

2.1.2. Arquitectura

Las redes de sensores, como ya se ha dicho, están compuestas principalmente por tres tipos de elementos: los nodos sensores, el gateway y la estación base.

Dependiendo de cuál sea el uso que se le vaya a dar a la red, la disposición de los nodos puede variar entre una estructura o topología fija, en la que sea determinante la posición de cada nodo, o una distribución aleatoria. Por ejemplo, si se desea monitorizar un viñedo, una distribución posible sería colocar los nodos sensores en posiciones estratégicas a lo largo de los surcos del viñedo, para asegurar la

monitorización completa del área del viñedo. Sin embargo, en aplicaciones de monitorización de fenómenos naturales de difícil acceso, como por ejemplo un glaciar, la única distribución posible es aleatoria, típicamente realizada arrojando los nodos desde un avión que sobrevuela el área a monitorizar.

Un ejemplo de la estructura y disposición de una red de sensores inalámbrica es la que puede verse en la Figura 4.

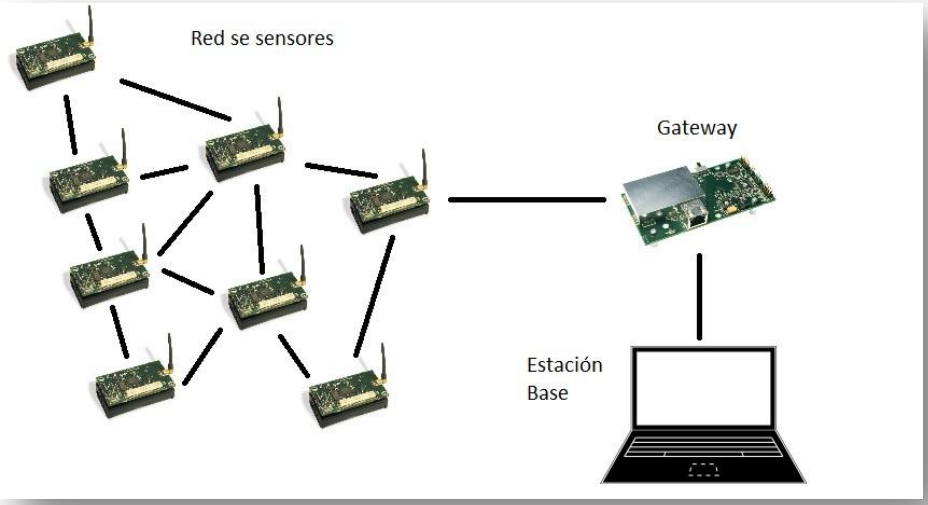

Figura 4: Red de sensores

A continuación se detalla la arquitectura y características de cada uno de los elementos que conforman una red de sensores.

2.1.2.1. Nodos sensores

Constituyen el elemento esencial de una red de sensores ya que son los encargados de realizar mediciones del entorno y transmitir la información. Para la primera tarea, el nodo sensor se conecta a una placa de sensores que incorpora un conjunto de sensores cuya funcionalidad dependerá del uso que se está dando a la red, por ejemplo, medir la temperatura o la presión, entre muchos más usos. Para la segunda tarea, el nodo sensor incorpora una radio de bajo consumo que permitirá transmitir y recibir información.

Típicamente, un nodo sensor está compuesto de los siguientes componentes de hardware:

- Un procesador, normalmente un microcontrolador o una FPGA, que incluye una memoria RAM (datos) y una memoria ROM (código).
- Una memoria flash de tamaño medio.
- Una radio de baja potencia
- Una fuente de energía, normalmente 2 baterías.

- Un bus de expansión que permite transferir los datos capturados por los sensores al microcontrolador.

Un ejemplo de nodo sensor es MicaZ (5), mostrado en la Figura 5. Estos nodos son los usados en este proyecto.

Figura 5: Nodo sensor MicaZ

Los elementos característicos de este tipo de nodo sensores son, por ejemplo:

- El microcrontrolador utilizado es un ATmega128L (6) que utiliza una arquitectura Harvard, es decir, una memoria RAM para datos (4 KBytes) y otra ROM para instrucciones (128 KBytes), y utiliza un conjunto de instrucciones RISC de 8 bits.
- La radio de la que disponen es de baja potencia para reducir el consumo de energía del nodo, el cual es uno de los principales problemas de las WSN. Limitando la potencia, también se limita el radio de acción. La capacidad de transferencia es de 250 Kbps y trabaja en la banda de frecuencias de 2.4 a 2.8 GHz.

2.1.2.2. Placa de sensores

Las placas de sensores son elementos hardware que se utilizan como nexo de unión entre el nodo sensor y los sensores, ya que en la propia placa se encuentran integrados dichos elementos. Ambos elementos, la placa y el nodo sensor se conectan mediante un bus de expansión. La Figura 6 representa una placa de sensores MDA300 compatible con los nodos MicaZ.

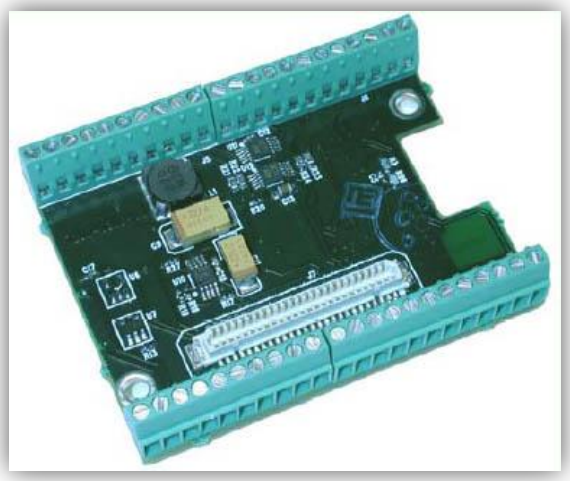

Figura 6: Placa de sensores

2.1.2.3 Gateway

Es un nodo especial encargado de la comunicación entre los nodos sensores y la estación base. Este nodo recibe la información obtenida por el resto de los nodos sensores y la retrasmite a la estación base para su procesamiento, para lo que es necesario que tenga una conexión directa con la estación base. Esa conexión se puede establecer mediante distintos tipos de tecnologías como, por ejemplo, mediante un cable USB o una conexión TCP/IP, siendo estos los más comunes.

Además, el gateway es el nodo encargado de coordinar al resto de nodos sensores (enviando por ejemplo información de control o comandos) y tiene la capacidad para actualizar o modificar el código que ejecutan los nodos a los que está conectado para, por ejemplo, adaptar la red a escenarios cambiantes en las que es necesario realizar reprogramaciones de forma remota.

Algunos de los ejemplos de gateway son los MIB520, MIB600 y MIB510, mostrados en la Figura 7. Como puede observarse la conexión a la estación base es diferente en cada caso, pudiendo ser USB, TCP/IP o un puerto RS-232, lo que proporciona gran versatilidad para conectarse a distintos tipos de plataformas.

Figura 7: Gateway MIB520, MIB600 y MIB510, respectivamente

2.1.2.4 Estación base

La estación base recibe los datos desde el gateway y se encarga de procesarlos y almacenarlos en algún dispositivo con mayor capacidad para su posterior uso. Normalmente suele ser un ordenador (PC o portátil) u otro dispositivo de potencia similar.

Adicionalmente, la estación base también puede ser utilizada para enviar información a la red de sensores mediante algún protocolo pre-establecido. Por ejemplo, la estación base podría modificar la configuración de un subconjunto de nodos, o de la red completa, mediante el envío de una serie de comandos, que el nodo sensor que los recibe ya lleva implementados.

2.2. TinyOS

Una vez vistos los aspectos hardware más importantes de las redes de sensores, es necesario hablar de los componentes software de estas redes. Estos nodos disponen de un sistema operativo de propósito general para la programación de aplicaciones. Uno de los sistemas operativos para nodos sensores más extendidos y populares es TinyOS.

TinyOS (7) es un sistema operativo de código libre, desarrollado inicialmente por la Universidad de Berkeley en Estados Unidos en el 2002. Este sistema operativo está escrito en nesC (8), un lenguaje de programación derivado de C.

Figura 8: TinyOS

TinyOS está especializado y optimizado para ser usado en redes de sensores y sistemas empotrados, debido a que está especialmente pensado para dispositivos con características hardware muy reducidas, es decir capacidad de procesamiento limitada, reducido espacio de almacenamiento, restricciones de batería, etc.

Además está desarrollado para soportar las intensivas operaciones de concurrencia propias de las redes de sensores, con gran cantidad de nodos, por lo que su uso está centrado principalmente en este tipo de sistemas.

Se basa en un modelo de ejecución basado en eventos. En este modelo se establece un sistema de funciones o manejadores, cuya ejecución se da como respuesta a eventos independientes. Estos eventos pueden ser tanto externos como internos. Una

interrupción hardware sería un ejemplo de un evento que provocaría que el manejador asociado ejecutara su código sin interrumpirse, desde el principio al final.

Este modelo de ejecución es el usado en los sistemas empotrados porque reduce la memoria consumida en la creación de los procesos, disminuyendo el espacio de memoria utilizado.

En el caso de TinyOS, se dispone de eventos y tareas. Los eventos tienen mayor prioridad que las tareas, por lo que estas podrán ser interrumpidas por los eventos pero no por otras tareas.

2.2.1 nesC

nesC (8) es un lenguaje derivado de C, diseñado especialmente para trabajar con nodos sensores y concretamente con el sistema operativo TinyOS.

Se basa en un sistema de programación basado en componentes. Esto es así porque las aplicaciones escritas en nesC están formadas por una serie de componentes que interactúan entre sí, y que ofrecen unos servicios por medio de interfaces. Estas interfaces son bidireccionales, es decir, especifican cuales son las funciones a implementar, tanto por el proveedor de la interfaz (comandos), como por el componente que utilizara dicha interfaz (eventos).

Para ilustrar de manera más eficaz los fundamentos básicos de nesC, es necesario ver la estructura y funcionamiento de un programa escrito en dicho lenguaje, como por ejemplo Blink (7). Este programa es uno de los ejemplos más básicos que pueden verse de nesC, ya que su funcionamiento es encender y apagar un LED de una mote.

Este programa está formado por dos archivos diferentes, el archivo de configuración Blink.nc y el archivo con la implementación del programa BlinkM.nc. A continuación podemos ver la estructura del archivo de configuración.

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

Como puede verse en la primera línea se indica que es un archivo de configuración, y en el apartado de implementación se indica qué componentes van a ser usados, y en las subsiguientes líneas se indica cómo van a ser usadas las interfaces de esos componentes. Para ello, mediante el *wiring* BlinkM.Timer -> SingleTimer.Timer, se indica qué interfaces son referenciados por los elementos usados en la aplicación: en

este caso indica que la interfaz Timer usado por BlinkM referencia a la interfaz Timer proporcionado por SingleTimer.

El archivo con la implementación del código para encender y apagar el LED es el siguiente:

```
module BlinkM {
      provides {
            interface StdControl;
      }
      uses {
            interface Timer;
            interface Leds;
      }
}
implementation {
      command result_t StdControl.init() {
            call Leds.init();
            return SUCCESS;
      }
      command result_t StdControl.start() {
            return call Timer.start(TIMER_REPEAT, 1000) ;
      }
      command result_t StdControl.stop() {
            return call Timer.stop();
      }
      event result_t Timer.fired() {
            call Leds.redToggle();
            return SUCCESS;
      }
}
```

En este archivo lo primero que se indica es cuales son las interfaces que se van proporcionar y cuales se van usar. A continuación se implementa el código del programa, que como puede verse, utiliza una sintaxis muy similar a C. Al iniciar la ejecución se establece un Timer que se encargara de encender y apagar el LED cada segundo.

2.3. Java

Este proyecto usará el lenguaje de programación Java para realizar la programación de un modulo de batería que complemente y amplíe la funcionalidad de la plataforma de simulación de redes de sensores Avrora, de la que se hablara en los siguientes apartados. Java es un lenguaje de programación desarrollado originalmente por Sun Microsystems, aunque actualmente pertenece a Oracle, ya que esta empresa adquirió a Sun Microsystems en 2009.

Java es uno de los principales lenguajes de programación usado en la actualidad en el mundo de la informática.

Figura 9: Java

Java tiene una sintaxis parecida a C, aunque con algunas simplificaciones, como en el uso de los punteros a memoria o el recolector de basura, que se ocupa automáticamente de liberar la memoria.

Las principales características de este lenguaje son las siguientes:

- Orientado a objetos. Uno de los principios básicos de Java es que está diseñado para la programación orientada a objetos.
- Multiplataforma: una gran ventaja de java es que es multiplataforma, es decir, la ejecución es independiente de la maquina en la que haya sido creado el programa, ya que las aplicaciones en java se ejecutan sobre una maquina virtual (JVM). Para ello el código fuente se traduce a *bytecode* que es independiente de la máquina de ejecución, ya que es la maquina virtual de java la encargada de ejecutar dicho *bytecode*.

2.3.1 Java Swing

Es una biblioteca desarrollada para sustituir a la biblioteca grafica original de java AWT, y mejorar la portabilidad de las aplicaciones desarrolladas con interfaz grafica, para así adecuarse más a la interoperabilidad ofrecida por Java.

Como ya se ha dicho el uso de Swing aumenta las capacidades multiplataforma de java, pero además consta de elementos y herramientas para mejorar la accesibilidad de las aplicaciones para personas con discapacidades.

Se basa en un sistema de componentes y contenedores muy jerarquizados, es decir a partir de la inicialización de un contenedor general se genera la interfaz gráfica añadiendo los componentes de la interfaz y nuevos contenedores para organizar la estructura general. Esta estructura, fuertemente jerarquizada, se puede ver en la Figura 10.

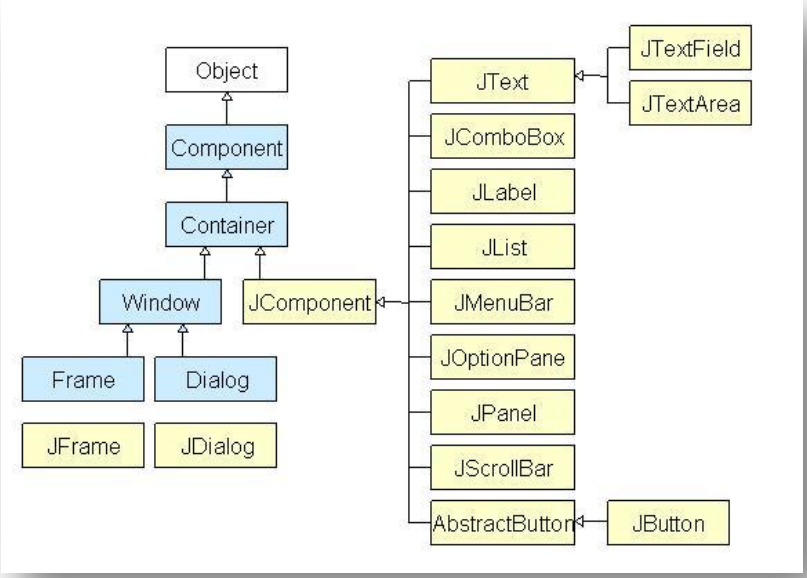

Figura 10: Jerarquía de Swing

2.4. Simuladores para redes de sensores

A continuación se exponen los simuladores de redes de sensores más utilizados en la actualidad.

2.4. 1. Avrora

El último elemento software necesario para el desarrollo de este proyecto es Avrora. Avrora (9) es un conjunto de herramientas, desarrolladas en Java, utilizadas para analizar y simular programas desarrollados en el sistema operativo TinyOS para los microcontroladores AVR de Atmel, en particular para nodos sensores Mica2 (5) y MicaZ.

Las características de este simulador se basan en la facilidad de uso para la realización de las simulaciones y las pruebas de control, ya que Avrora es un simulador capaz de simular de manera muy precisa el tiempo en que los eventos tienen lugar en los nodos sensores. Además, el modelo de energía que utiliza está basado en AEON (10), el cual es considerado uno de los más precisos y utilizados modelos de energía para nodos sensores.

Es un proyecto de código abierto, por lo que se permite que distintos desarrolladores añadan nuevos módulos y funcionalidades para ir completando el proyecto inicial, como es el caso de este Trabajo de Fin de Grado, que añade un nuevo monitor de batería.

Permite monitorizar de manera detallada todos los procesos realizados por los nodos sensores mediante la utilización de monitores. Un programador puede crear los

monitores que desee y añadirlos al repositorio de software. Además permite la posibilidad de ejecutar y simular programas distintos en los diferentes nodos de la red.

Algunos ejemplos de monitores son:

- **Energy Monitor**: se encarga de comprobar el gasto de energía de cada uno de los componentes de los nodos y mostrarle dicha información al usuario.

```
==={ Energy consumption results for node 0}==================
Node lifetime: 637009920000 cycles,  86400.0 seconds
CPU: 1953.3778452066117 Joule
   Active: 1947.115554106376 Joule, 632406621879 cycles
   Idle: 6.262291100235718 Joule, 4603298121 cycles
   ADC Noise Reduction: 0.0 Joule, 0 cycles
   Power Down: 0.0 Joule, 0 cycles
   Power Save: 0.0 Joule, 0 cycles
   RESERVED 1: 0.0 Joule, 0 cycles
   RESERVED 2: 0.0 Joule, 0 cycles
   Standby: 0.0 Joule, 0 cycles
   Extended Standby: 0.0 Joule, 0 cycles
Yellow: 5.371093750000001E-9 Joule
   off: 0.0 Joule, 637009919994 cycles
   on: 5.371093750000001E-9 Joule, 6 cycles
Green: 284.29400536385094 Joule
   off: 0.0 Joule, 319427671099 cycles
   on: 284.29400536385094 Joule, 317582248901 cycles
Red: 284.69998511295574 Joule
   off: 0.0 Joule, 318974154812 cycles
   on: 284.69998511295574 Joule, 318035765188 cycles
Radio: 4872.590852089022 Joule
   Power Off:: 6.532793782552084E-8 Joule, 8027497 cycles
   Power Down:: 1.0951334635416666E-7 Joule, 13457 cycles
   Idle:: 2.0194091796874998E-7 Joule, 1165 cycles
   Receive (Rx):: 4868.767473006674 Joule, 636461858599
cycles
   Transmit (Tx):31::3.823378705566406 Joule, 540019282
cycles
SensorBoard: 181.44 Joule
   on:  : 181.44 Joule, 637009920000 cycles
flash: 0.5184 Joule
   standby: 0.5184 Joule, 637009920000 cycles
   read: 0.0 Joule, 0 cycles
   write: 0.0 Joule, 0 cycles
   load: 0.0 Joule, 0 cycles
```

Como puede verse en el log, este monitor obtiene el consumo de energía de cada uno de los elementos hardware de un nodo, como puede ser la CPU o la radio, a lo largo de todo el tiempo de ejecución. Por ejemplo, en el caso de los LEDs (Yellow, Green y Red) se indica el tiempo que pasan apagados y encendidos, y la energía consumida al estar

encendidos. En otros elementos como la radio también se incluye la energía y los ciclos consumidos en el envío y recepción de paquetes.

- **Packet Monitor**: muestra al usuario todo el proceso del intercambio de mensajes entre los nodos. A continuación se puede ver el informe final tras la ejecución de avrora utilizando este monitor.

```
=={ Packet monitor results }==========================
Node   sent (b/p)    recv (b/p)    corrupted(b)lostinMiddle(p)
------------------------------------------------------------
0  1955728 / 72942     256132 / 8914     0           180
1   251473 / 8922      356715 / 14384    5059        235
2 51394178 / 1253538   233935 / 8344     0           164
3 76120192 / 1872411  1139348 / 30239    2714        692
4  1070840 / 26140   60599106 / 1482723  0           18224
```

Este monitor genera un log con el número de paquetes enviados y recibidos. También se incluye el número de bits corruptos y los paquetes perdidos durante la transmisión. Por ejemplo, de los paquetes enviados o recibidos por el nodo 0, ninguno se ha visto corrupto pero si 180 paquetes se han perdido durante la transmisión.

- **Memory Monitor**: recoge información sobre el uso de la memoria por el programa, incluyendo el número de accesos de lectura y escritura.
- **Real-time Monitor**: Ajusta el tiempo de ejecución para que se parezca lo más posible al tiempo real.

Finalmente cabe destacar algunas de las opciones de simulación más importantes de Avrora:

- **Action**: indica la acción a ejecutar, por ejemplo cargar un programa en el simulador.
- **Config-file**: utilizado para cargar un archivo con comandos de configuración adicionales para Avrora.
- **Monitors**: indica cuales son los monitores a ejecutar por el simulador.

Un ejemplo de utilización de estos comandos es el siguiente:

```
java avrora.Main -action=simulate -platform=micaz -simulation=sensor-network -seconds=3 -monitors=leds Blink.elf
```

En él se indica en qué tipo de plataforma se realizará la simulación de una red de sensores, durante cuánto tiempo, los monitores que se utilizaran y que el programa a ejecutar es Blink.elf.

El resultado de ejecutar esos comandos sería el siguiente:

```
Avrora [Beta 1.7.107] - (c) 2003-2007 UCLA Compilers Group
```

```
Loading Blink.elf...[OK: 0.068 seconds]
=={ Simulation events }============================
Node          Time    Event
------------------------------------------------------------
   0       8006682  off off on
   0       8006684  off on  on
   0       8006686  on  on  on
   0       8006688  on  on  off
   0       8006690  on  off off
   0       8006692  off off off
   0       9793689  off off on
   0      11579289  off off off
   0      11579762  off on  off
   0      13364889  off on  on
   0      15150489  off on  off
   0      15150962  off off off
   0      15151490  on  off off
   0      16936089  on  off on
   0      18721689  on  off off
   0      18722162  on  on  off
   0      20507289  on  on  on
============================================================
Simulated time: 22118400 cycles
Time for simulation: 1.337 seconds
Total throughput: 16.543306 mhz
```

Como puede verse, en este caso el resultado muestra el tiempo en el que se produce cada evento, que en este caso es el encendido y apagado de un LED.

2.4. 2. TOSSIM

TOSSIM (11) es un simulador de alta precisión enfocado en las redes de sensores basadas en TinyOS (7), para que los usuarios puedan probar y analizar los algoritmos y programas desarrollados en un entorno controlado. Fue desarrollado por la Universidad de Berkeley de California en Estados Unidos en 2003.

Para poder conseguir ese entorno controlado para las pruebas, TOSSIM realiza algunas simplificaciones respecto al mundo real.

- No proporciona un sistema de consumo de energía, ya que no dispone de un modelo del tiempo de ejecución de la CPU por lo que no es posible calcular el gasto de energía.
- Tampoco dispone de un modelo de radio completo.
- TOSSIM simplifica también el comportamiento de TinyOS, por lo que las aplicaciones que funcionen en el simulador pueden no funcionar en una mote real.

2.4. 3. Castalia

Castalia (12) es un simulador de redes de sensores inalámbricas, redes de área corporal, del inglés Body Area Networks (BAN), y redes de baja potencia. Está basado en la plataforma OMNeT++ (13), y como en el resto de casos, su uso principal es el de simulación y evaluación de aplicaciones en diferentes plataformas para las distintas redes nombradas. Se diferencia de los otros simuladores de redes de sensores en los siguientes aspectos:

- Un nuevo modelo de canales avanzado, adaptado para sistemas WiMAX, WSN y BAN
- Modelo de radio avanzado, que surge de los sistemas de radio de baja potencia reales.
- Sincronización de reloj.
- Disponibilidad inicial de los protocolos MAC y enrutamiento, como *multipathRings* y *bypassRouting*.
- Escalabilidad.

Como ya se ha dicho, Castalia está desarrollada a partir de la plataforma OMNeT++, un framework diseñado para el desarrollo de sistemas controlados por eventos. Además, este sistema le proporciona un gran nivel de modularidad, seguridad y velocidad.

2.5. Fuentes de energía

El último punto a tratar son las fuentes de energía usadas por los nodos sensores de las WSN. En este apartado se consideran tanto las baterías tradicionales como las nuevas fuentes de energía que se están empezando a usar para aumentar la autonomía de los nodos.

Este punto es importante, ya que las baterías son las encargadas de alimentar el nodo sensor para permitir su funcionamiento. En el caso de que la batería caiga por debajo de cierto nivel, el nodo sensor finalizará sus operaciones. Por tanto, la vida de un nodo sensor depende de cómo de rápido un nodo sensor vacía sus baterías, lo cual a su vez depende de los componentes hardware utilizados, su consumo y el tiempo de operación de cada uno de ellos. En general, todos los protocolos y aplicaciones desarrollados para nodos sensores tienen como objetivo hacer un uso eficiente de las baterías.

2.5.1. Baterías

Típicamente, la principal fuente de energía usada en las redes de sensores y en otros dispositivos de características similares son las baterías químicas o pilas. Las más comunes son:

- Pilas alcalinas: son pilas desechables de larga vida y gran densidad energética.
- Baterías de ion de litio: baterías recargables que carecen de efecto memoria, lo que las hace ideales para trabajar con redes de sensores

El funcionamiento de las pilas alcalinas se basa en la reacción química de reducción-oxidación (redox). Para ello se utilizan dos electrodos metálicos (ánodo y cátodo) sumergidos en un líquido conductor (electrolito). Los electrodos reaccionan con el electrolito (hidróxido de potasio – KOH) provocando en el ánodo (zinc - Zn) un exceso de electrones y en el cátodo (dióxido de manganeso – $Mn0_2$) la reacción opuesta, es decir, la falta de electrones. La corriente eléctrica se genera al trasladarse los electrones sobrantes del ánodo hasta el cátodo. La Figura 11 muestra el interior de una pila alcalina con los elementos que participan en el proceso de redox.

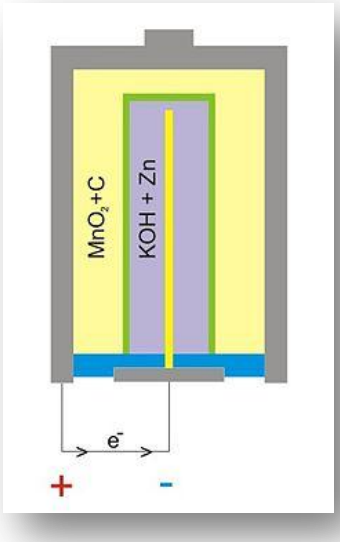

Figura 11: Pila alcalina

La capacidad de las pilas alcalinas depende de la potencia que consuma el dispositivo que la utiliza, en el caso de los nodos sensores esa capacidad varía entre los 2000 mAh y los 3000 mAh.

Las baterías de iones de litio se diferencian de las pilas alcalinas en varios aspectos:

- Como electrolito usan una sal de litio
- El proceso de redox es reversible, lo que permite que las baterías puedan reutilizarse tras haberlas sometido a una corriente eléctrica para revertir el proceso.
- No tiene efecto memoria, por lo que la capacidad de la batería no se verá afectada por las recargas.
- Por norma general, tienen menor capacidad que las pilas alcalinas aunque esto se compensa por su capacidad para ser recargada.

2.5.2. Fuentes alternativas

Dada la limitación que el uso de las baterías imponen sobre el tiempo de vida de los nodos sensores, actualmente se están desarrollando y buscando nuevas fuentes alternativas de energía que permitan alimentar al nodo sensor. Una de las fuentes alternativas que mayor interés ha despertado son las placas fotovoltaicas.

Las placas fotovoltaicas, como puede verse en la imagen, están formadas por un conjunto de celdas o células fotovoltaicas interconectadas entre sí. Estas celdas están fabricadas a partir de materiales semiconductores, principalmente silicio.

Figura 12: Placas fotovoltaicas

El funcionamiento básico de las placas fotovoltaicas se basa en el efecto fotoeléctrico de algunos materiales, es decir en la incidencia de fotones procedentes del sol en materiales semiconductores previamente tratados y preparados, para provocar el movimiento de los electrones. Cuando los fotones chocan con los átomos de dichos materiales se genera una carga eléctrica suficiente para soportar el funcionamiento de los nodos sensores durante periodos de tiempo largos. La Figura 13 representa de forma simplificada el funcionamiento del efecto fotoeléctrico.

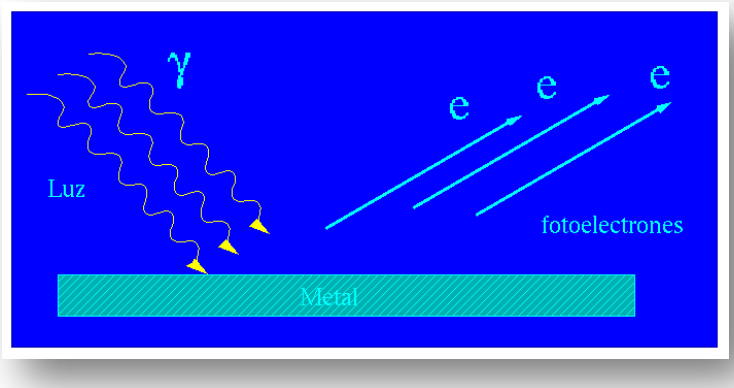

Figura 13: Efecto fotoeléctrico

El uso de las placas fotovoltaicas aumenta la autonomía del nodo sensor, ya que no depende de un sistema que tiene de almacenamiento energético perecedero y de vida

limitada. Por esto también se favorece la aparición de nuevas aplicaciones y usos para las redes de sensores.

Pese a todos los beneficios nombrados, este sistema también tiene sus problemas ya que la generación de la energía no es constante a lo largo del día, por lo que para los momentos en los que la energía generada es insuficiente, es necesario el uso de baterías que complementen a las células solares. Estas baterías se recargaran en las situaciones de mayor generación de energía y se gastaran en los momentos de escasez. Incluso con este sistema mixto, no se puede garantizar el funcionamiento constante de los nodos por lo que serán necesarios buenos algoritmos de gestión de batería para así alargar el ciclo de actividad de los nodos lo máximo posible (14).

Actualmente ya se están usando en algunos dispositivos específicos, como pueden ser los motes Eko (5). Estos motes incluyen una célula solar de la que obtienen la energía, y además están diseñados para un uso específico, la monitorización ambiental y no admiten ningún tipo de modificación, por lo que están listos para su despliegue, como pueden verse en la Figura 14.

Figura 14: Motes Eko

3. Análisis

En este apartado se identifican y se explican cuáles han sido los requisitos iniciales que han condicionado y guiado el desarrollo de este proyecto. Además se incluyen unos casos de uso, para facilitar la comprensión del uso del sistema.

La finalidad de este proyecto es desarrollar, sobre la plataforma Avrora, un módulo para incluir un nuevo monitor que informe al usuario del nivel de la batería de los nodos sensores que se utilizan durante una simulación. Para calcular el nivel de la batería se tendrá en cuenta el uso que hace el nodo de los componentes de hardware, para así, mediante la energía necesaria para alimentar cada componente y el número de ciclos que se ha usado, obtener la energía total consumida.

Para informar al usuario sobre el nivel de la batería de cada nodo sensor, se utilizará una interfaz gráfica y una serie de imágenes que representen la carga de las baterías. Como complemento a esto, tras finalizar la simulación, se deberá mostrar por pantalla la información del uso de las baterías, y además, si el usuario lo desea podrá guardar dicha información en un archivo.

El monitor que se desarrollará se denomina *battery* y deberá especificarse en la lista de monitores que se pasan en el momento de invocar Avrora. El monitor *battery* incluye además tres nuevos parámetros de ejecución:

- El primero de ellos es *-batteryLevel*, mediante el cual el usuario deberá indicar la carga inicial de la batería en julios. Por ejemplo, -batteryLevel=21600.
- El segundo parámetro, *-threshold*, seguido de dos valores numéricos separados por coma, servirá para configurar el valor de los umbrales de control. Estos thresholds son los límites que dividen el nivel de las baterías en tres secciones: nivel alto, medio y bajo. Un ejemplo de su uso, seria: *-threshold*=60,30.
- Finalmente, el último parámetro, *-logbattery*, se utilizará para guardar un fichero de registro de resultados. Por ejemplo, *-logbattery*=battery.log.

Para la especificación de los requisitos y casos de uso se ha seguido el sistema de análisis de requisitos especificado en Métrica v.3 (15). Métrica es una metodología de planificación, desarrollo y mantenimiento de sistemas de información desarrollada por el Ministerio de Hacienda y Administraciones Públicas de España.

3.1 Requisitos funcionales

A continuación se enumeran los requisitos funcionales especificados:

Identificador: UR-F001	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	Se deberá desarrollar un sistema de monitorización de las baterías de nodos sensores, añadiendo un nuevo monitor a la plataforma Avrora.

Tabla 1: UR-F001

Identificador: UR-F002	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El sistema deberá mostrar el nivel de energía de la batería de cada nodo mediante una interfaz gráfica.

Tabla 2: UR-F002

Identificador: UR-F003	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El sistema deberá mostrar el nivel de energía de la batería mediante un porcentaje y una imagen con un código de colores. Dicho código de colores será: • Verde, para la fase entre la carga total y el threshold superior. • Naranja, para la fase entre los threshold superior e inferior. • Rojo, a partir del threshold inferior hasta el fin de la carga de la batería.

Tabla 3: UR-F003

Identificador: UR-F004	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El sistema deberá desconectar o deshabilitar los nodos cuya batería se haya agotado, para ello el sistema deberá extraer el nodo de la simulación.

Tabla 4: UR-F004

Identificador: UR-F005	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta

Descripción	El sistema deberá permitir que el usuario introduzca la cantidad de batería inicial en Julios, mediante el comando *-batteryLevel*. Por defecto, se considerarán baterías cuya carga inicial sea 21800 Julios.

Tabla 5: UR-F005

Identificador: UR-F006	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El sistema deberá permitir que el usuario introduzca los porcentajes de control de los threshold, mediante el comando *–threshold*, seguido de los dos valores que se corresponderán con los threshold superior e inferior. Además deberá contar con unos valores por defecto. Los thresholds son los limites que dividen el nivel de las baterías en tres secciones, nivel alto, medio y bajo.

Tabla 6: UR-F006

Identificador: UR-F007	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El sistema deberá incluir valores de thresholds por defecto. Estos valores serán: • 50% del nivel de energía como threshold superior • 25% de la energía de la batería como threshold inferior

Tabla 7: UR-F007

Identificador: UR-F008	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El sistema deberá considerar que el nodo 0 es el nodo gateway, y por tanto, no consumirá energía, dado que se encuentra conectado a la estación base, la cual le suministra la energía para poder operar.

Tabla 8: UR-F008

Identificador: UR-F009	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El sistema deberá ser capaz de mostrar el estado de hasta 512 nodos sensores en una única pantalla, utilizando únicamente el scroll vertical para desplazarse entre todos los nodos incluidos en la simulación.

Tabla 9: UR-F009

Identificador: UR-F010	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El sistema deberá mostrar por pantalla, una vez finalizado el tiempo de ejecución indicado por el usuario, un informe con la información de la batería residual de cada nodo.

Tabla 10: UR-F010

Identificador: UR-F011	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	La información que se mostrará por pantalla será: -El identificador del nodo, -El número de ciclos y los segundos consumidos en cada una de las fases delimitadas por los thresholds, -El número de ciclos y segundos de ejecución total, -La carga energética de las baterías no utilizada por los nodos en Julios.

Tabla 11: UR-F011

Identificador: UR-F012	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El sistema deberá, si el usuario lo indica, generar un log de registro con la información del uso de la batería de cada nodo. Para ello el usuario deberá emplear el comando –logbattery seguido del nombre del archivo.

Tabla 12: UR-F012

Identificador: UR-F013	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	La información que se guardará en el log para cada nodo será: -El identificador de nodo. -El número de ciclos y los segundos consumidos en cada una de las fases delimitadas por los thresholds.

	-El número de ciclos y segundos de ejecución total. -La carga energética de las baterías no utilizada por los nodos en Julios.

Tabla 13: UR-F013

Identificador: UR-F013	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	Se han de incluir 3 nuevos parámetros de simulación: • -batteryLevel, para indicar la energía total de la batería. • -threshold, para configurar los thresholds de control • -logbattery, indicando donde guardar el log de información.

Tabla 14: UR-F013

3.2 Requisitos de apariencia

En este apartado se detallan los requisitos relacionados con la apariencia de la interfaz gráfica del sistema.

Identificador: UR-A001	
Prioridad: Media	**Fuente:** Cliente
Necesidad: Opcional	
Claridad: Media	**Verificabilidad:** Media
Descripción	Los colores usados en las imágenes deben ser fácilmente distinguibles y ser adecuados para personas con deficiencias visuales, como el daltonismo.

Tabla 15: UR-A001

Identificador: UR-A002	
Prioridad: Media	**Fuente:** Cliente
Necesidad: Opcional	
Claridad: Media	**Verificabilidad:** Media
Descripción	La imagen del nodo 0 (gateway) será distinta a las imágenes del resto de los nodos, para poder identificar claramente que este nodo se encuentra conectado a la estación base.

Tabla 16: UR-A002

3.3 Requisitos de rendimiento

A continuación se encuentran los requisitos relacionados con el rendimiento del sistema.

Identificador: UR-R001	
Prioridad: Media	**Fuente:** Cliente
Necesidad: Opcional	
Claridad: Alta	**Verificabilidad:** Media
Descripción	El sistema deberá poder mostrar la información de gran cantidad de nodos sin que el rendimiento global del sistema se vea afectado de manera significativa respecto al rendimiento de la propia plataforma Avrora.

Tabla 17: UR-R001

3.4 Requisitos de Sistema

Los requisitos de sistema son aquellos que especifican el hardware o software relacionados con la aplicación. A continuación se describen los requisitos del sistema identificados para nuestro proyecto.

Identificador: UR-S001	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El sistema se desarrollará sobre la plataforma Avrora versión 1.7.106.

Tabla 18: UR-S001

Identificador: UR-S002	
Prioridad: Alta	**Fuente:** Cliente
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El lenguaje de programación utilizado será Java.

Tabla 19: UR-S002

Identificador: UR-S003	
Prioridad: Media	**Fuente:** Cliente
Necesidad: Opcional	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El nuevo módulo deberá ser multiplataforma, al igual que Avrora, y deberá funcionar tanto en el sistema operativo Windows como Linux.

Tabla 20: UR-S003

3.5 Casos de uso

En este apartado se incluyen los casos de uso que indican cuales son las posibles interacciones que el usuario podrá tener con el sistema desarrollado. Para facilitar la comprensión de dichos casos de uso se incluye un diagrama en el que se muestran de manera simplificada dichos casos de uso.

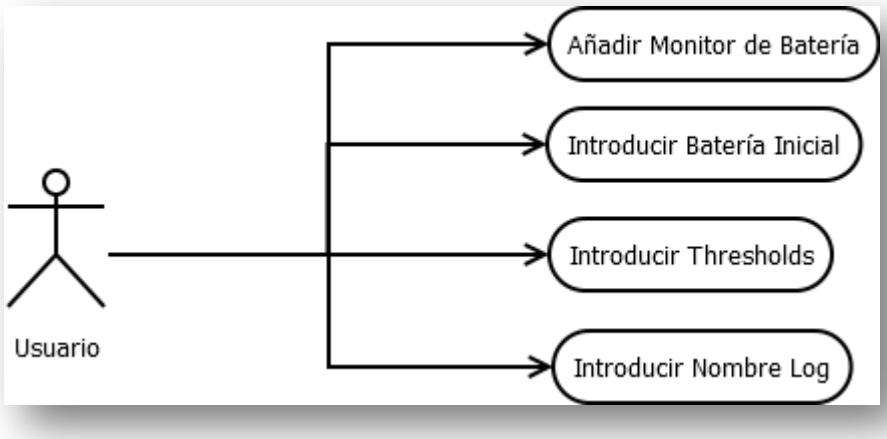

Figura 15: Casos de uso

A continuación se indican de forma más detallada los casos de uso antes mencionados.

CU-01	
Nombre	Añadir monitor de batería
Actor	Usuario
Objetivo	Añadir a la simulación el monitor de batería
Precondiciones	Todos los comandos de simulación necesarios para la simulación básica introducidos
Postcondiciones	Se ejecuta el monitor de energía
Escenario	El usuario añade el monitor de batería a la lista de monitores a ejecutar, incluyendo en el comando de simulación *-monitors* el parámetro *-battery*

Tabla 21: CU-01

CU-02	
Nombre	Introducir batería inicial
Actor	Usuario
Objetivo	Modificar el nivel de batería inicial por defecto por un valor personalizado
Precondiciones	Todos los comandos de simulación necesarios para la simulación básica introducidos y el monitor de batería añadido
Postcondiciones	El nivel de batería inicial se corresponde con el introducido por el usuario
Escenario	El usuario añade el comando de simulación *–batteryLevel:*, seguido por el valor que desee para la batería inicial

Tabla 22: CU-02

CU-03	
Nombre	Introducir thresholds
Actor	Usuario
Objetivo	El usuario introduce los thresholds de control que desee para la simulación
Precondiciones	Todos los comandos de simulación necesarios para la simulación básica introducidos y el monitor de batería añadido
Postcondiciones	Los thresholds usados en la simulación no son los thresholds por defecto sino los introducidos por el usuario
Escenario	El usuario añade el comando de simulación *–threshold*, seguido por los 2 valores, separados por una coma, que se corresponden con los thresholds superior e inferior respectivamente.

Tabla 23: CU-03

CU-04	
Nombre	Introducir nombre del fichero de registro o *log*
Actor	Usuario
Objetivo	Permitir al sistema generar un log de registro con los resultados de la simulación, guardado con el nombre que el usuario desee
Precondiciones	Todos los comandos de simulación necesarios para la simulación básica introducidos y el monitor de batería añadido
Postcondiciones	Finalizada la simulación se ha generado un archivo con el log de registro
Escenario	El usuario añade el comando de simulación *–logbattery:*, seguido por el nombre del archivo donde se guardara el log de registro

Tabla 24: CU-04

4. Modelo del Sistema

A continuación se describe y desarrolla el modelo de energía del sistema desarrollado. Este modelo se utiliza en la aplicación desarrollada para ser capaz de calcular en cualquier momento del tiempo la energía consumida por un determinado nodo, así como la energía residual en la batería.

El propósito de este modelo de energía es poder calcular de forma teórica el gasto de energía total *E* de un nodo Micaz *i (i ∈ 1..n)* en un instante cualquiera de tiempo *t*., Cada nodo dispone de dos baterías, con una carga inicial E_B. El modelo de energía permite calcular para cada nodo la energía consumida *E* y la energía restante o residual del nodo E_n para cualquier instante de tiempo *t* (t≤ tiempo simulación).

Para ello, lo primero que será necesario es describir los elementos o parámetros iniciales a partir de los cuales podremos obtener la energía consumida.

Sea un conjunto de *n* nodos sensores compuestos por una serie de *k* componentes de hardware H= {h_1, h_2, h_3,…, h_k}. En particular, en el caso de que los nodos sensores sean modelo Micaz dichos componentes de hardware son:

Componentes Hardware MicaZ
CPU
Led rojo
Led verde
Led amarillo
Radio
Placa sensora
Memoria Flash

Tabla 25: Componentes Hardware

Cada componente podrá tener una serie de estados distintos de energía según su funcionamiento, S= {s_1, s_2, s_3,…, s_h}. Por ejemplo, el componente de hardware radio podría estar en distintos estados de energía en función de la tarea que esté realizando en un instante de tiempo: On, Off, Transmisión, Recepción. A continuación, se asigna a cada componente de hardware de un nodo MicaZ todos sus posibles estados de energía.

Componente de hardware (H)	Estado de energía (S)
CPU	Active
	Idle
	ADC Noise Reduction
	Power Down
	Power Save
	RESERVED 1
	RESERVED 2
	Standby
	Extended Standby
LED Rojo	On

	Off
LED Verde	On
	Off
LED Amarillo	On
	Off
Radio	Power off
	Power on
	Idle
	Receive
	Transmit
Placa de sensores	On
	Off
Memoria Flash	Standby
	Read
	Write
	load

Tabla 26: Componentes hardware y estados

De estos elementos podemos concluir que $\forall h \in H \; \exists \, s(h) \in S$, por lo que la energía consumida por un componente en la unidad de tiempo (segundo) viene determinado por la tupla formada por el componente y el estado en el que se encuentra, E(h, s). Estos valores son proporcionados por el fabricante en el datasheet del nodo MicaZ (5). La siguiente tabla muestra el consumo energético para cada componente representada en unidades Amperio·hora (Ah).

Componente H	Estado S	Energía E(h,s)
CPU	Active	0,0075667 Ah
	Idle	0,0033433 Ah
	ADC Noise Reduction	9,88E-4 Ah
	Power Down	1,58E-4 Ah
	Power Save	1,237E-4 Ah
	RESERVED 1	0,0 Ah
	RESERVED 2	0,0 Ah
	Standby	2,3456E-4 Ah
	Extended Standby	2,433E-4 Ah
Led Rojo	On	0,0022 Ah
	Off	0,0 Ah
Led Verde	On	0,0022 Ah
	Off	0,0 Ah
Led Amarillo	On	0,0022 Ah
	Off	0,0 Ah
Radio	Power off	2,0E-8 Ah
	Power on	2,0E-5 Ah
	Idle	4,26E-4 Ah
	Receive	0,0188 Ah
	Transmit	0,0174 Ah
Placa Sensora	On	7,0E-4 Ah
	Off	0,0 Ah
Memoria Flash	Standby	2,0E-6 Ah
	Read	0,0040 Ah

	Write	0,0015 Ah
	Load	2,0E-6 Ah

Tabla 27: Componentes, estados y consumos

Estos valores energéticos de consumo de cada componente de hardware son utilizados por el simulador Avrora para calcular en tiempo de ejecución el consumo de los componentes de hardware en la aplicación que está siendo simulada. Avrora usa el modelo de energía AEON (10) para realizar los cálculos. En este modelo es necesario trabajar con los datos en Julios ya que la energía de la batería viene indicada en Julios. El paso de amperios-hora a julios se realiza mediante la siguiente fórmula:

$$E(h,s)_J = E_{Ah} * t * V$$

Siendo t el tiempo en segundos, que en nuestro caso será de 3600 segundos, E_{Ah} es el consumo del componente en una hora, y V es el voltaje del nodo MicaZ que es constante y se corresponde con 3 voltios. Con estos datos la ecuación para transformar de amperios-hora a julios sería la siguiente:

$$E(h,s)_{J/h} = E_{Ah} * t * V = E_{Ah} * 3600 * 3$$

Avrora proporciona todos sus resultados en segundos por lo que es recomendable obtener la energía consumida en un segundo, para así facilitar los cálculos.

$$E(h,s)_{J/s} = \frac{E(h,s)_{J/h}}{3600}$$

Los nodos sensores se encargan de ejecutar un conjunto de aplicaciones, *A*, que se ejecutarán durante un periodo de tiempo *t*, por algunos o todos los componentes hardware que forman parte del nodo, por lo que la energía consumida durante la ejecución de una aplicación dependerá de la energía consumida por unidad de tiempo de cada componente usado y el tiempo total de ejecución de la tarea:

$$E_A = \sum_{i=1\ldots p} E(h,s)_i \cdot t$$

Al tratarse de nodos con distintos componentes que trabajan de forma independiente, es posible que durante el tiempo de ejecución cada componente se ejecute durante un tiempo distinto por lo que para poder calcular la energía consumida por dicho componente primero hay que calcular los segundos de ejecución real. Para ello debemos conocer, en el caso de Avrora, los ciclos totales de ejecución de la aplicación, y los ciclos que ha estado ejecutando el componente, y mediante la siguiente formula obtener el tiempo:

$$t = \frac{Ciclos_{(h,s)} * t_{Total}}{Ciclos_{Totales}}$$

Una vez obtenido el tiempo por cada componente y estado ya es posible obtener la energía consumida total de la aplicación.

$$E_A = \sum_{i=1\ldots p} E(h,s)_i \cdot t_i = \sum_{i=1\ldots p} E(h,s)_i * \frac{Ciclos(h,s)_i * t_{Total}}{Ciclos_{Totales}}$$

Con estas fórmulas sería posible obtener el gasto de energía para uno de los nodos de la simulación en cualquier instante del tiempo de ejecución.

En el caso de este sistema también se pretende calcular la energía restante en la batería de un nodo durante cualquier momento t de la ejecución de A. Para ello contamos con la energía inicial de la batería E_B, con lo que obtendríamos que la energía del nodo en un instante *t* cualquiera será igual a:

$$E_t = E_B - E_A$$

Como complemento también se puede obtener de forma aproximada el tiempo de descarga de la batería, para lo que necesitamos conocer la carga inicial de la batería, E_B y la energía consumida por el nodo en 1 hora, E_A:

$$t_{Descarga} = \frac{E_B}{E_{A/hora}}$$

Para poder comprobar si el modelo de energía desarrollado es fiable para el cálculo de la energía consumida por un nodo, utilizamos los resultados de ejecutar una simulación en Avrora con el monitor de energía.

A continuación se muestra el resultado de ejecutar el programa TestNetworkLPLBS durante 24 horas. Este programa se encarga de la transmisión y enrutamiento de paquetes desde los nodos hasta la estación.

```
=={ Energy consumption results for node 1 }================
Node lifetime: 637009920000 cycles,  86400.0 seconds

CPU: 1269.4259511036198 Joule
   Active: 748.0742811012421 Joule, 242968183387 cycles
   Idle: 520.1979841220401 Joule, 382388228929 cycles
   ADC Noise Reduction: 0.0 Joule, 0 cycles
   Power Down: 0.0 Joule, 0 cycles
   Power Save: 0.0 Joule, 0 cycles
   RESERVED 1: 0.0 Joule, 0 cycles
   RESERVED 2: 0.0 Joule, 0 cycles
   Standby: 0.0 Joule, 0 cycles
   Extended Standby: 1.1536858803374022 Joule, 11653507684
                        cycles

Yellow: 33.13530971451823 Joule
   off: 0.0 Joule, 599994766748 cycles
   on: 33.13530971451823 Joule, 37015153252 cycles

Green: 33.135695538981125 Joule
   off: 0.0 Joule, 599994335747 cycles
   on: 33.135695538981125 Joule, 37015584253 cycles
```

```
Red: 570.2063999217122 Joule
   off: 0.0 Joule, 37534342 cycles
   on: 570.2063999217122 Joule, 636972385658 cycles

Radio: 4386.8667473588785 Joule
   Power Off:    : 1.070042202718099E-4 Joule, 13148678587 cycles
   Power Down:   : 0.042816352465820315 Joule, 5261273391 cycles
   Idle:         : 0.08063010562744141 Joule, 465156215 cycles
   Receive (Rx):: 138.32012070833335 Joule, 18081677056 cycles
   Transmit (Tx):31:: 4248.423073188232 Joule, 600053134751
                  cycles

SensorBoard: 181.44 Joule
   on:  : 181.44 Joule, 637009920000 cycles

flash: 0.5184 Joule
   standby: 0.5184 Joule, 637009920000 cycles
   read: 0.0 Joule, 0 cycles
   write: 0.0 Joule, 0 cycles
   load: 0.0 Joule, 0 cycles
```

Tabla 28: Resultado ejecución

Para comprobar si el modelo desarrollado es válido podemos, a partir de los datos suministrados por Avrora, realizar los cálculos para ver si se corresponden con la realidad. Por ejemplo, se puede comprobar si la energía consumida por la CPU en el estado Active coincide con el modelo de energía, lo que formaría la tupla E(h,s)=(CPU,Active).

Partiendo de la información del gasto de energía suministrada por Avrora para la tupla escogida, E(h,s)=0,0075667 Ah, lo primero que debemos obtener es la energía en Julios que consume dicho componente en una hora.

$$E(h,s)_{J/h} = 0{,}0075667\ Ah = 0{,}0075667 * 3600 * 3 = 81{,}72036\ Jh$$

A continuación se obtiene la energía consumida por segundo.

$$E(h,s)_{J/s} = \frac{81.72036}{3600} = 0.0227001\ Js$$

Una vez obtenidos estos datos ya es posible calcular la energía consumida durante la ejecución.

$$E(h,s)_{Total} = E(h,s)_{j/s} * t = E(h,s)_{j/s} * \frac{Ciclos_{(h,s)} * t_{Total}}{Ciclos_{Totales}}$$
$$= 0.0227001 * \frac{242968183387 * 86400}{637009920000} = 748.0742811J$$

Este resultado como puede verse coincide con el resultado dado por la ejecución de Avrora.

Este proceso habría que repetirlo para cada tupla componente-estado para obtener la energía total consumida por el nodo en 24 horas, que en este caso sería 748.0742811 julios.

Con este dato podríamos calcular de forma teórica cuanto tardaría el nodo en consumir toda su energía hasta agotarla, asumiendo que únicamente está utilizando el componente CPU y siempre en modo Active

$$t_{Descarga} = \frac{E_B}{E_{A/h}} = \frac{21600}{748.0742811/24} = 80.06\ horas$$

Este resultado solo puede considerarse como una aproximación, ya que el gasto de energía depende de muchos factores, algunos aleatorios como puede ser la pérdida de paquetes en las transmisiones de radio que generan la consecuente retransmisión, por lo que la información obtenida no debe considerarse como un resultado invariable y exacto pero sí como un resultado razonablemente aproximado.

5. Diseño

Este capítulo describe el diseño de la aplicación desarrollada. En primer lugar se describirá el diseño de la interfaz gráfica que utiliza la aplicación. A continuación se describirá el diseño de la aplicación desarrollada. Puesto que el objetivo de este proyecto es la implementación de una funcionalidad añadida al simulador Avrora, en este capítulo además se tratará la integración del nuevo monitor con la aplicación Avrora.

5.1 Diseño de la interfaz

En este apartado se muestran los diferentes prototipos considerados para la interfaz gráfica del sistema a desarrollar.

El principal objetivo es que la interfaz creada fuera lo más simple posible para facilitar así la localización y comprensión de la información deseada. La información que nos interesa conocer, para cada nodo, es su nivel de batería en cada momento de la simulación, y además, al final de ésta, la energía residual de cada nodo.

Un primer prototipo puede verse en la Figura 16. En este caso solo se mostraba el número de nodo y la imagen correspondiente al nivel de su batería.

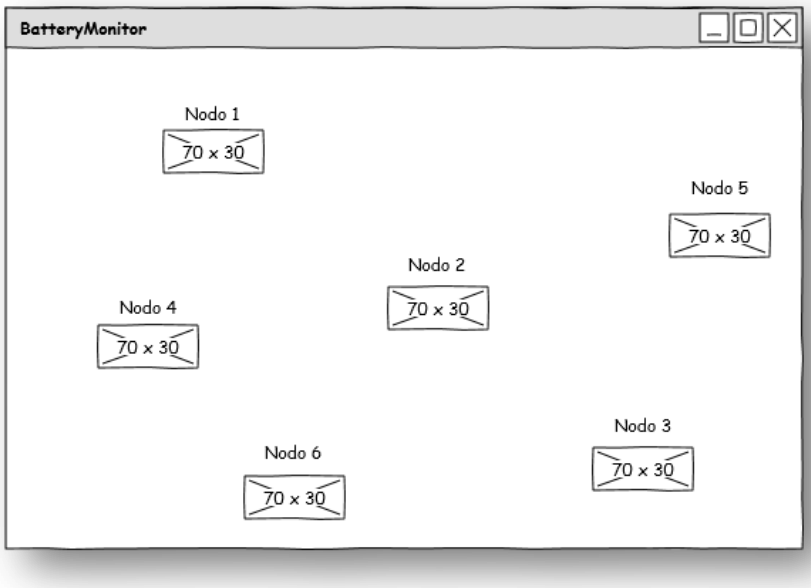

Figura 16: Diseño inicial de la interfaz

La distribución de los nodos se basa en la ubicación dada en el archivo de configuración de los nodos introducidos en los parámetros de simulación de Avrora. La

distribución de los nodos se obtiene del fichero de topología introducido. En dicho fichero viene indicado el número de nodo y su ubicación en un espacio tri-dimensional, dado por las coordenadas x,y,z respectivamente. Un ejemplo de archivo de topología es el siguiente:

```
node0 0 0 0
node1 25 -25 0
node2 50 -50 0
node3 -10 -25 0
node4 75 -10 0
node4 10 -50 0
```

Tabla 29: Archivo de topología

El nodo 0, denominado node0 y localizado en la posición 0,0,0 representa el nodo gateway de la red; este nodo dispondrá de unas características especiales que se ven más adelante.

Este diseño quedó descartado ya que únicamente con las imágenes no se puede conocer el estado real de la batería por lo que fue necesario desarrollar un segundo prototipo que incluya de alguna manera dicha información.

En el segundo diseño, correspondiente con la Figura 17, se añade una caja de texto para cada nodo en el que se muestra de forma numérica el nivel de las baterías.

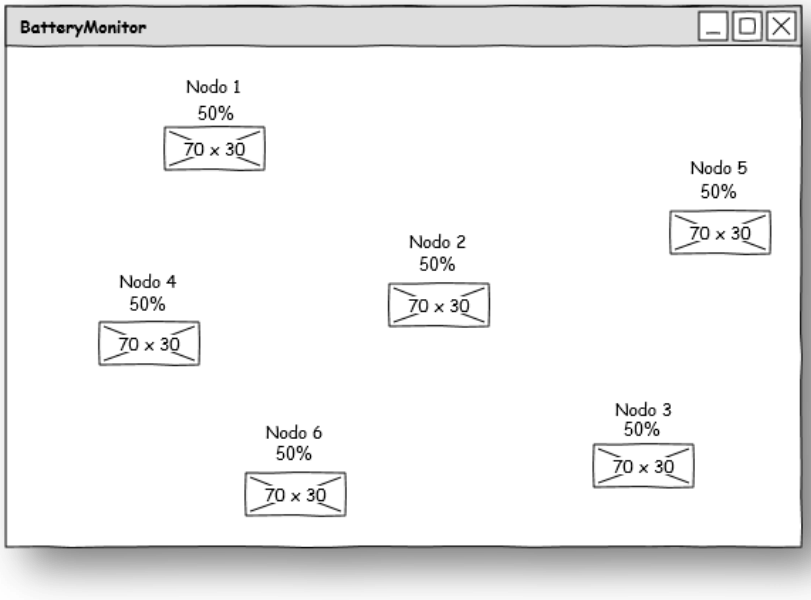

Figura 17: Segundo prototipo de diseño de la interfaz gráfica

La razón por la que fue descartado el primer prototipo queda solventada en este segundo diseño, ya que se añade un porcentaje numérico en el que se indica el nivel real de la batería, con lo que los problemas relacionados con la información de la batería quedan resueltos.

La distribución de los nodos en este segundo prototipo es igual que en el anterior con los nodos distribuidos por toda la ventana según su distribución en la red de acuerdo al fichero de topología. Si bien esta representación especifica fielmente la localización de los nodos en la red, este diseño lleva a problemas cuando el número de nodos a mostrar es muy numeroso, ya que la ventana se llega a sobrecargar de tal manera que encontrar la información de un nodo concreto de forma rápida llega a ser imposible ya que la información mostrada en la pantalla puede llegar a catalogarse como caótica. Es importante tener en cuenta, que el simulador debería poder visualizar el estado de las baterías de hasta 512 nodos (requisito UR-F009). Por esta razón, finalmente este prototipo también quedó descartado.

El siguiente prototipo diseñado es el que puede verse en la Figura 18. En este diseño la información mostrada por cada nodo sigue siendo la misma que en los dos prototipos anteriores, pero se ha modificado la distribución de los nodos para que sea más fácil su localización. En este proyecto no se buscaba una representación fidedigna de la topología de la red, sino un sistema sencillo para conocer la batería de los nodos, por tanto tiene que ser fácil localizar un nodo concreto. Por lo que para conocer la topología del sistema será necesario comprobar el fichero de topología.

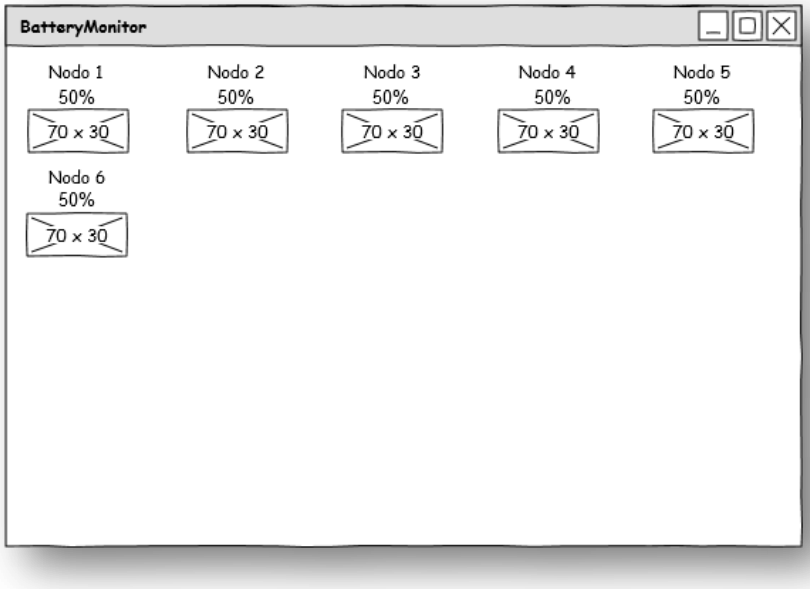

Figura 18: Tercer prototipo de diseño de de la interfaz gráfica

Además, este diseño se adapta mejor a los requisitos iniciales establecidos por el cliente, ya que se pedía que solo fuese necesario utilizar el scroll vertical para la localización de la información de la batería de los nodos (requisito UR-F009).

Finalmente se escogió este prototipo como diseño final para la interfaz gráfica de la aplicación ya que, como ya se ha dicho, es una interfaz sencilla e intuitiva y, además, cumple con los requisitos impuestos por el usuario.

Para la imagen usada por el nodo *Gateway* se ha decidido usar la siguiente representación:

Figura 19: Gateway

Dado que se presupone en este proyecto, que el nodo gateway está conectado a la estación base, no necesita una imagen que represente el estado de sus baterías pues estará alimentado directamente desde la estación base.

El diseño usado para las imágenes que mostrarán el nivel de la batería no ha cambiado desde el inicio y se corresponden con las imágenes usadas en el proyecto final. Estas imágenes pueden verse en la Figura 20.

Figura 20: Nivel de baterías

Los colores usados son los característicos para la representación de baterías, lo que facilita al usuario su comprensión. Además dicho código de colores viene especificado en el requisito UR-F003, tal como se explica a continuación:

- Verde, para la fase entre la carga total y el threshold superior.
- Naranja, para la fase entre los threshold superior e inferior.
- Rojo, a partir del threshold inferior hasta el fin de la carga de la batería.
- Blanco, cuando el nodo esta sin energía y desconectado.

A continuación puede verse cómo a partir de un comando sencillo de ejecución de Avrora, se genera la interfaz y qué cambios se producen en la interfaz durante la ejecución.

```
java avrora.Main -action=simulate -simulation=sensor-network -
nodecount=1,9 -platform=micaz -input=elf -seconds=3600 -report-seconds
-monitors=battery -batteryLevel=100 -thresholds=50,25 -radio-range=75
-topology-file=topology.out TestNetworkLPLBS.elf
TestNetworkLPLNodes_50DC.elf
```

Los parámetros –batteryLevel y -thresholds son los parámetros incluidos en Avrora durante el desarrollo de este proyecto. Estos parámetros permiten configurar en el nuevo monitor tanto la carga inicial de las baterías como los thresholds de control.

Una vez iniciada la simulación se generaría una ventana similar a la de la Figura 18, con 10 nodos, de esos 10 nodos el nodo 0 ejecutará la aplicación TestNetworkLPLBS.elf y los otros nueve nodos (segundo valor en el parámetro nodecount=1,9), ejecutan la aplicación TestNetworkLPLNodes_50DC.elf. Estas aplicaciones se encargan de la transmisión y enrutamiento de paquetes desde los nodos hasta la estación base pasando por el Gateway. Las imágenes correspondientes a la batería de los nodos se corresponderían con la primera imagen de la Figura 20 (batería verde), dado que al principio de la simulación las baterías de los nodos parten con la carga completa. Según los parámetros de ejecución introducidos en el comando anterior, se especifica una carga inicial de las baterías de 100 Julios, y los thresholds de control se corresponden con el 50% y el 25% de la carga de dichas baterías. Esto quiere decir, que cuando los nodos consuman 50 Julios (el 50% de la carga) su imagen cambiará desde la batería verde a la batería naranja, y al consumir 75 Julios, en la batería solo quedaran 25 julios o lo que es lo mismo el 25% de la carga restante, la imagen pasará a ser la de la batería roja. Finalmente, cuando los nodos consuman los 100 Julios se realizará el último cambio de imagen, y la imagen mostrada será la de la batería vacía (batería blanca). Este proceso se produciría para todos los nodos de la simulación hasta que finalizase el tiempo de simulación, en este caso, 3600 segundos. Al finalizar la ejecución las baterías podrán estar en cualquiera de los 4 estados antes descritos dependiendo de la carga residual que tengan las baterías.

El tiempo y velocidad con la que los nodos descargarán sus baterías vendrá determinado tanto por la aplicación que se está simulando, como por la topología de la red. Así por ejemplo, en un escenario en el que un determinado nodo actúa como nodo padre de un conjunto de N vecinos, dicho nodo padre agotará antes su batería que el resto de nodos en su vecindario, dado que será el encargado de transmitir sus mensajes y además de reenviar los paquetes procedentes de sus nodos vecinos. Para ilustrar esto, disponemos de la Figura 21, en la que el nodo 0 actúa como nodo Gateway y la comunicación entre dicho nodo y el resto de los nodos debe pasar obligatoriamente por el nodo 1, provocando así que el nodo 1 descargue más rápidamente su batería ya que, el nivel de uso de dicho nodo será mucho mayor que el del resto de nodos.

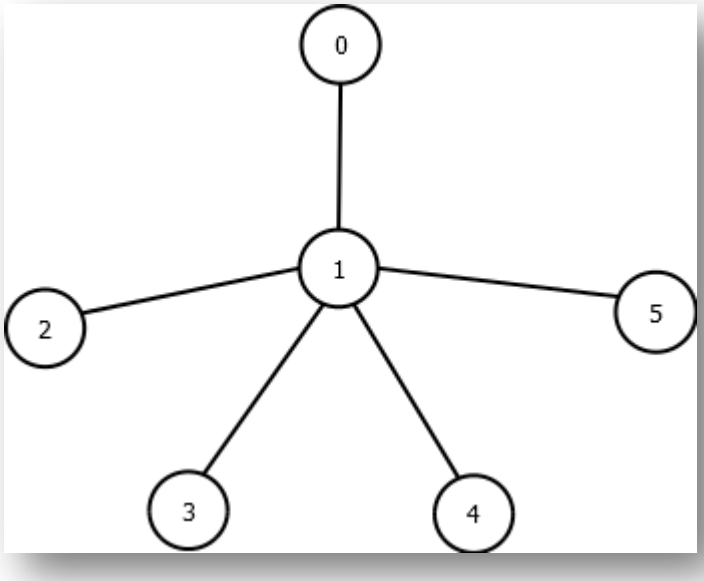

Figura 21: Ejemplo del escenario

5.2 Diseño de clases

En este apartado se muestra el diseño inicial de las clases necesarias para la inclusión del nuevo monitor, el monitor –*battery*, además de las relaciones entre ellas y algunos de los aspectos básicos necesarios para la posterior implementación de este módulo.

El diseño del monitor de batería está basado en el estudio de los otros monitores que vienen incluidos por defecto en Avrora, concretamente del análisis de las clases *EnergyMonitor* y *PacketMonitor*, en las que basa su diseño, es decir, la estructura de algunas de las clases a desarrollar se corresponde con la estructura de estos monitores.

En la Figura 22 se presenta el diagrama de clases diseñado para este nuevo módulo.

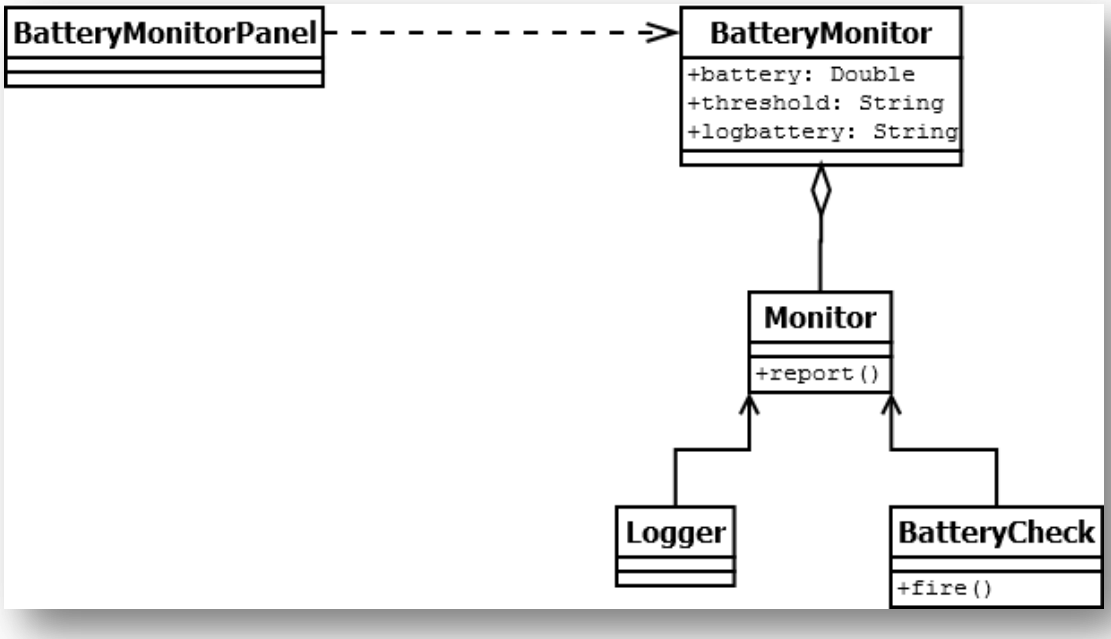

Figura 22: Diagrama de clases

A partir del diagrama de clases, se procederá a explicar el uso y las características de todas las clases a implementar en orden ascendente según el diagrama.

- La clase *BatteryCheck* será la clase encargada de ejecutar las comprobaciones del gasto de energía de los componentes hardware de cada nodo. En cada comprobación, esta clase obtendrá el gasto total de todos los elementos hardware del nodo y comprobará si ese valor es mayor o igual que carga total de la batería. En el caso de que esto ocurra detendrá la simulación para dicho nodo, mientras que si es menor seguirá realizando las comprobaciones. Todas estas comprobaciones se realizaran dentro del método *fire()*. La Figura 23 representa el diagrama de flujo diseñado para la clase *BatteryCheck*.

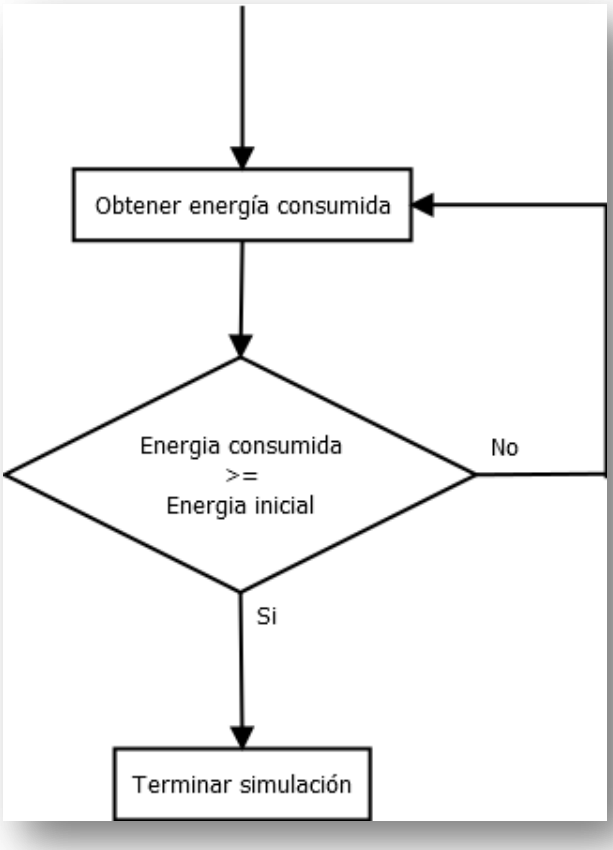

Figura 23: Diagrama de flujo de BatteryCheck

Un ejemplo práctico sería la comprobación en el instante t de la simulación, en el que la energía consumida es 80 y la energía inicial es 100. En este caso, al ser menor la energía consumida que la inicial no ocurriría nada, pero si la energía consumida fuese 100,1 entonces la clase *BatteryCheck* debería detener la simulación de dicho nodo.

- En los requisitos dados por el cliente se pedía que al finalizar la ejecución se pudiese generar un archivo con los resultados de la ejecución. La clase *Logger* será la encargada de generar dicho archivo con toda la información recopilada durante la simulación. El archivo se generará tras la instanciación de la clase mediante su constructor.
- Estas dos primeras clases serán usadas por la clase *Monitor* que, aunque no es la clase principal, sí podría decirse que es la más importante, ya que es la encargada de generar todo el sistema de monitorización con la instanciación de las clases, y además, mostrar el log o registro de información por pantalla tras la simulación mediante el método *report()*. Además, se encarga para cada nodo de añadir a la ventana la información relacionada con el nivel de la batería.
- La siguiente clase es la clase *BatteryMonitor* que es la clase principal del sistema aunque sus únicas tareas serán instanciar a la clase monitor y almacenar los parámetros recibidos de los comandos de simulación. Estos parámetros son el

nivel inicial de la batería (batteryLevel), los dos valores de los threshold de control (threshold) y el nombre del archivo para el log de registro (logbattery).

- Finalmente, solo queda por detallar el funcionamiento de la clase *BatteryMonitorPanel*. Esta clase es la encargada de generar la interfaz gráfica necesaria para el monitor, es decir, creará la ventana donde se mostrará la información de las baterías, y las imágenes que se usarán en dicha ventana.

5.3 Integración con Avrora

A continuación se describe como interacciona el nuevo monitor con la plataforma de simulación Avrora.

El nuevo monitor, al ser un módulo añadido a Avrora y trabajar conjuntamente con esta aplicación, debe intercambiar información con los elementos ya existentes en Avrora. Dicha interacción se produce principalmente con dos de las clases de Avrora, la clase *Simulator* encargada de realizar las simulaciones y la clase *EnergyControl* relacionada con el uso de energía por parte de los componentes hardware de los nodos. En este intercambio de información, el monitor desarrollado deberá poder obtener:

- El número total de nodos de la simulación, obtenido de los parámetros de ejecución iniciales (nodecount).
- Para cada nodo, y cada instante de simulación:
  - Su identificador (ID).
  - El número de ciclos de reloj que lleva ejecutando.
  - La energía consumida por cada componente del nodo, proporcionada por la clase *EnergyControl*.

Además de obtener esos datos, el nuevo monitor también deberá ser capaz de poder modificar los elementos de la simulación, ya sea deteniendo la simulación de los nodos cuya batería se haya agotado o insertando eventos en la simulación para poder realizar las comprobaciones de energía.

A continuación, se incluye un diagrama en el que puede verse el intercambio de datos entre las distintas clases del nuevo monitor y las clases ya incluidas en Avrora.

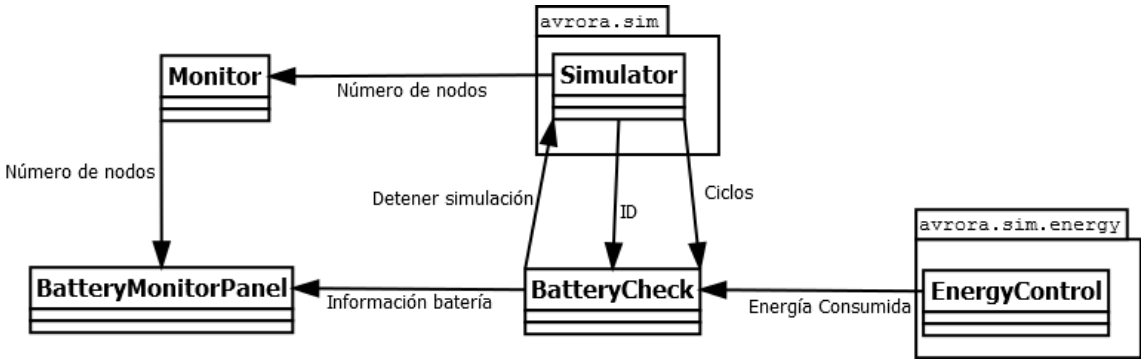

Figura 24: Integración con Avrora

Un ejemplo de este intercambio de energía sería a la hora de generar la ventana de la interfaz al comienzo de la simulación. La clase monitor obtendría el número de nodos totales de la simulación mediante la clase *Simulator*, contenida en el paquete *avrora.sim.* Una vez obtenido este dato se lo envía a la clase *BatteryMonitorPanel* para que genere la nueva ventana.

Otro ejemplo sería cuando un nodo ha agotado toda su energía, entonces la clase *BatteryCheck* deberá detener la simulación de dicho nodo utilizando los métodos de la clase *Simulator*.

Como puede observarse, la interacción del nuevo monitor con las otras clases ya desarrolladas de Avrora es muy simple, ya que la mayoría de la información intercambiada es muy sencilla y dicho intercambio solo se produce en momentos puntuales de la simulación, como es la instanciación del monitor de batería o la detención de la simulación de alguno de los nodos.

Lo anterior no puede aplicarse al intercambio entre la clase *BatteryCheck* y la clase *EnergyControl*, ya que las comprobaciones de gasto de energía se realizan de manera constante por lo que ambas clases interaccionan de manera continua durante toda la simulación.

6. Implementación

En este punto se describen las clases necesarias para la implementación del módulo de batería en Avrora. Para el desarrollo de este proyecto ha sido necesario implementar dos nuevas clases. La primera clase es el propio monitor, denominada *BatteryMonitor*, y la segunda clase es la encargada de generar la ventana de la interfaz gráfica, denominada *BatteryMonitorPanel*

6.1 BatteryMonitor

Como ya se ha comentado, la clase *BatteryMonitor* contiene la implementación del nuevo monitor.

Esta clase está compuesta por un grupo de subclases anidadas que implementan elementos indispensables para el funcionamiento del monitor. La jerarquía de clases es la siguiente:

- BatteryMonitor
  - Monitor
    - BatteryCheck
    - Logger

Es decir, la clase principal *BatteryMonitor* tiene como clase anidada la clase Monitor, y ésta a su vez contiene a las clases *BatteryCheck* y *Logger*.

La clase *BatteryMonitor* hereda de la clase *MonitorFactory* por lo que en ella será necesario sobrecargar algunos de los métodos de dicha clase.

Los primeros elementos importantes son los tres atributos encargados de recoger y almacenar los valores de los parámetros introducidos en los parámetros de ejecución de Avrora específicos de este monitor (-batteryLevel, -thresholds, -logbattery). Para ello se utiliza la función *newOption*, que recibe como parámetros el nombre dado al comando, el valor por defecto y la descripción de dicho comando, y permite incluir y utilizar nuevos parámetros en las opciones de simulación de Avrora. Además esta función permite almacenar los valores de dichos parámetros en las variables indicadas.

```
protected final Option.Double BATTERY = newOption("batteryLevel", 21600.0,
"This option specifies the initial number of joules in each batteries");

protected final Option.Str THRESHOLD = newOption("threshold", "",
"This option specifies the limits of the control sections of the
batteries");

protected final Option.Str LOG = newOption("logbattery", "",
"This option specifies where be save the log archive of the use of the
batteries");
```

El último de los atributos de esta clase es una lista de monitores donde cada nodo añadirá su monitor, para que así puedan acceder a ellos el resto de clases.

Finalmente, esta clase dispone de tres funciones encargadas de generar el nuevo monitor. De esas funciones, dos son constructores que solo realizan la función de llamar al constructor de la clase *MonitorFactory*. La función restante, *newMonitor*, recibe como parámetro una instancia de la simulación actual. Para generar el nuevo monitor llama al constructor de la clase Monitor, el cual se verá con detenimiento más adelante cuando se desarrolle dicha clase.

```
    public BatteryMonitor() {
        super("The \"battery\" is a monitor to show energy consumption.");
    }

    public BatteryMonitor(String s2) {
        super(s2);
    }

    public avrora.monitors.Monitor newMonitor(Simulator s) {
        return new Monitor(s);
    }
```

A continuación se describen todas las clases que componen la clase principal *BatteryMonitor*.

6.1.1 BatteryCheck

La clase *BatteryCheck* se encargará de comprobar el nivel de batería de cada nodo aproximadamente 10 veces por segundo. Esta clase implementa la interfaz *Event* incluida en la clase avrora.sim.Simulator.

Se dispone de tres variables globales en las que se almacena la siguiente información:

- El intervalo de tiempo entre cada comprobación de esta clase, para obtener dicho valor es necesario calcular el número de ciclos de reloj que se corresponden con el tiempo de ejecución deseado. En este caso si queremos que el evento se ejecute cada 10 segundos el número de ciclos introducido debe ser 737280. Este número concreto lo hemos obtenido a partir del numero de ciclos ejecutados en una hora, que concretamente son 637009920000 ciclos, y a partir de ahí obtener los ciclos correspondientes a 0,1 segundos, que son los 737280 ciclos antes indicados.
- La energía total consumida por el nodo en el momento concreto en que se realiza la comprobación.
- El *Iterator* necesario para iterar entre todos los componentes hardware del nodo.

Esta clase dispone de un constructor encargado de añadir esta clase al sistema de eventos de Avrora, mediante la función *insertEvent* de la clase *Simulator*. El siguiente código realiza esta operación:

```
public BatteryCheck(){
    simulator.insertEvent(this, interval);
}
```

Dentro de esta clase, el método *fire()* será el método encargado de realizar la comprobación del nivel de batería en cada momento de la ejecución, para lo cual, cada vez que se ejecute, obtendrá la energía consumida por cada componente y calculará la energía total consumida por cada nodo, de la siguiente manera:

```
while(it.hasNext()){

  totalEnergy += ((Energy)it.next()).getTotalConsumedEnergy();

  totalEnergyBattery=totalEnergy;
}
```

Dicho código se encarga de recorrer todos los elementos hardware y sus diferentes estados para así obtener sus gastos de energía individuales, y calcular el gasto global de energía del nodo.

Una vez obtenido dicho resultado obtenemos la energía restante en la batería para así comprobar en cuál de los estados delimitados por los thresholds se encuentra.

```
if((energy-totalEnergy)<(energy*threshold_inf/100)){
…
}else{
   if((energy-totalEnergy)<(energy*threshold_sup/100)){
       …
   }
}
```

Como puede verse, mediante una serie de comprobaciones se comprueba si el nivel de la batería ha cambiado de estado, y en el caso de que haya cambiado de estado se cambia la imagen del nodo que se muestra por pantalla, junto al valor numérico del porcentaje de la batería. Además, se almacena en las variables de la clase *Monitor* el número de ciclos y segundos en el que se produce el cambio de fase.

```
label1.setIcon(BatteryMonitorPanel.icon2);
fase1 = simulator.getClock().getCount();
secf1=simulator.getClock().cyclesToMillis(fase1)/1000;
```

Si se comprueba que la energía restante es igual o menor de 0 se establece la imagen de batería agotada y se interrumpe la simulación de dicho nodo, lo cual se realiza de la siguiente manera:

```
label1.setText("nodo "+simulator.getID()+" 0% bateria agotada");
label1.setIcon(BatteryMonitorPanel.icon4);
simulator.stop();
```

Estas comprobaciones se realizan para todos los nodos excepto el nodo 0, ya que se presupone que es el nodo gateway y por tanto su fuente de energía es distinta a la del resto de nodos.

6.1.2 Logger

La clase *Logger* se encargará de generar un fichero de registro o log de resultados cuando acaba la simulación, e implementa la clase *EnergyObserver*.

Consta de un constructor y una serie de métodos encargados del tratamiento del fichero. El constructor de esta clase se encarga de generar el archivo de log con toda la información necesaria. Para ello realiza los tres siguientes pasos:

- Abrir el archivo con el nombre pasado por el parámetro –logbattery y almacenado en una variable de la clase *BatteryMonitor*.

```
String fileName = LOG.get();
try {
   this.file = new BufferedWriter(new FileWriter(fileName));
} catch (IOException e) {
   throw Util.unexpected(e);
}
```

- Añadir al archivo las líneas iniciales con la explicación de cada dato mostrado en el log.

```
write("Battery monitor results");
newLine();
write("Node  Phase 1(cycles/seg)            Phase 2(cycles/seg)
            Phase 3(cycles/seg)            Total(cycles/seg)
            Residual Energy");
newLine();
write("-------------------------------------------------------------
-----------");
newLine();
```

- Mediante un *Iterator*, recorrer todos los nodos y guardar en cada archivo la información relativa al consumo de energía almacenada en las variables de la clase *Monitor*. Para ello, se ejecutará el siguiente código:

```
write(""+mon.simulator.getID()+"
```

```
        "+mon.fase1+"/"+df.format(mon.secf1)+"
        "+mon.fase2+"/"+df.format(mon.secf2)+"
        "+mon.fase3+"/"+df.format(mon.secf3)+"
        "+mon.total+"/"+df.format(mon.totals)+"
        "+df.format(mon.energiaFinal)+"/"+mon.energy);
newLine();
```

Los siguientes métodos son creados como sustitución de los métodos básicos pertenecientes a la clase *BufferedWriter*, para así adaptarlos al sistema de excepciones de Avrora, y son usados en el constructor antes descrito.

El método *write* recibe como parámetro un String con los datos a escribir en el fichero. Se intenta escribir en el fichero y si no se consigue lanza una excepción.

```
private void write(String text) {
                try {
                    file.write(text);
                } catch (IOException e) {
                    throw Util.unexpected(e);
                }
            }
```

NewLine se encarga, a la hora de gestionar el fichero del log, de generar una nueva línea a continuación del texto existente.

```
private void newLine() {
                try {
                    file.newLine();
                } catch (IOException e) {
                    throw Util.unexpected(e);
                }
            }
```

Finish se encarga de vaciar el buffer y cerrar el fichero del log, en el caso de que se produzca un error lanza una excepción.

```
public void finish() {

               try {
                    file.flush();
                    file.close();
                } catch (IOException e) {
                    throw Util.unexpected(e);
                }
            }
```

También ha sido necesario sobrescribir el método *stateChange*, aunque en este caso, como no es necesario para la aplicación, no tiene ningún uso.

```
@Override
public void stateChange(Energy energy) {
}
```

6.1.3 Monitor

La clase *Monitor* es la responsable de instanciar para cada nodo la clase *BatteryCheck* encargada de las comprobaciones de energía. Además, también es la clase responsable de mostrar por pantalla la información del uso de la batería cuando la simulación finaliza.

Esta clase dispone de un constructor y un método denominado *report*, además de un gran número de variables necesarias para almacenar la información relativa al uso de la batería. Estas variables almacenan el número de ciclos y segundos entre cada cambio de fase, los valores de los thresholds de control, la energía total consumida y otras variables de menor importancia usadas por los distintos métodos y clases del monitor.

El constructor recibe como parámetro una instancia de la clase *Simulator*, y realiza los siguientes pasos:

- Se activa e inicializa el sistema de control de energía. Para ello se obtiene a partir del parámetro *simulator* la instanciación de la clase *EnergyControl*.

```
energyControl = s.getEnergyControl();
energyControl.activate();
```

- Añadir a la ventana generada por la clase *BatteryMonitorPanel*, la información correspondiente a cada nodo. Para el nodo 0 incluye el identificador y la imagen asignada al nodo gateway e instancia la clase *BatteryMonitorPanel*, pasándole como parámetro el número de nodos totales de la simulación. Para el resto de nodos, se incluye su identificador y la imagen de la batería correspondiente a la primera fase del estado de la batería (batería verde).

```
if(s.getID()==0){
  new BatteryMonitorPanel(s.getSimulation().getNumberOfNodes());
  label1=new JLabel("node"+simulator.getID(),BatteryMonitorPanel.base,
JLabel.CENTER);
}else{
  label1=new JLabel("node"+simulator.getID(),BatteryMonitorPanel.icon,
  JLabel.CENTER);
}
```

- Se instancia la clase *BatteryCheck* encargada de realizar las comprobaciones del nivel de la batería, para ello se comprueba que el nivel de batería introducido sea estrictamente mayor de 0, ya que no son validos niveles negativos o nulos.

```
if ( (energy =BATTERY.get()) > 0 ) {
   energy=BATTERY.get();
```

```
    batteryCheck = new BatteryCheck();
}
```

- Se comprueba que se han introducido por parámetros los valores de los thresholds y se almacenan dichos valores en las variables que luego serán usadas por la clase *BatteryCheck*. En el caso de que no se hayan introducido se toman los valores por defecto, que han sido definidos en 50% y 25% para los thresholds superior e inferior respectivamente.

```
if(!THRESHOLD.isBlank()){
   String []div=THRESHOLD.get().split(",");
   int aux1=Integer.parseInt(div[0]);
   int aux2=Integer.parseInt(div[1]);
    if(aux1>aux2){
        threshold_sup=aux1;
        threshold_inf=aux2;
     }else{
        threshold_sup=aux2;
        threshold_inf=aux1;
     }
 }
```

El método *report* se ejecuta cuando termina el tiempo de simulación de la aplicación y muestra. Este método visualiza en el terminal donde se está ejecutando la simulación, la información relativa al uso de las baterías durante la ejecución y la energía residual de cada batería.

Para ello, primero comprobamos si la lista de monitores de la clase *BatteryCheck* no está vacía y se genera un nuevo *Iterator* con el que poder recorrer todos los elementos de cada nodo, para lo que se dispone de un bucle en el que para cada nodo se realiza los siguientes pasos:

- Dependiendo de la fase en el que se encuentre cada nodo en el momento de finalizar la simulación, se almacena la información relativa al final de la simulación. Así por ejemplo si un nodo todavía se encuentra en la fase 1 (batería verde) cuando termine la simulación se almacena para dicha fase el numero de ciclos totales de la simulación:

```
  if(mon.fase1==0){
    mon.fase1=mon.simulator.getClock().getCount();
    mon.secf1=mon.simulator.getClock().cyclesToMillis(mon.fase1)/1000;
  }
```

- Se imprime por pantalla la información de cada fase:

```
Terminal.print(StringUtil.rightJustify(mon.simulator.getID(), 4));
Terminal.print(StringUtil.rightJustify(mon.fase1+"/"+df.format(mon.sec
f1), 23));
Terminal.print(StringUtil.rightJustify(mon.fase2+"/"+df.format(mon.sec
f2), 23));
```

```
Terminal.print(StringUtil.rightJustify(mon.fase3+"/"+df.format(mon.sec
f3), 23));
```

- Se obtiene el número total de ciclos y segundos que el nodo ha ejecutado y a continuación se obtiene la energía residual del nodo, y se muestran por pantalla:

```
mon.total=mon.fase1+mon.fase2+mon.fase3;
mon.totals=mon.secf1+mon.secf2+mon.secf3;
Terminal.print(StringUtil.rightJustify(mon.total+"/"+df.format(mon.tot
als), 23));
mon.energiaFinal=mon.energy-mon.totalEnergyBattery;
if(mon.energiaFinal<0){
      mon.energiaFinal=0;
}
Terminal.print(StringUtil.rightJustify(df.format(mon.energiaFinal)+"/"
+mon.energy, 20));
```

- A continuación se comprueba si en los parámetros de entrada de la simulación se ha indicado si hay que generar el log, y en el caso de que esto sea afirmativo se genera una nueva instancia de la clase *Logger*.

```
if(i.hasNext()==false){
   if ( !LOG.isBlank() ) {
      logger = new Logger();
   }
}
```

Una vez finalizada la ejecución del bucle se eliminan los monitores y se cierra el archivo del log si éste se había generado.

```
if ( logger != null ){
    logger.finish();
}
```

6.2 BatteryMonitorPanel

La clase *BatteryMonitorPanel* es la encargada de generar y configurar la ventana emergente de la interfaz gráfica donde se muestra la información de la batería de los nodos.

Para la creación de la ventana se ha utilizado la biblioteca gráfica de Java Swing. Esta biblioteca, como ya se ha visto en capítulos anteriores, es la más adecuada para realizar interfaces gráficas para aplicaciones multiplataforma, y además dispone de un sistema de contenedores y otros elementos jerarquizados que representan los diferentes elementos visuales utilizados en las interfaces gráficas. En esta clase se realizan las siguientes acciones:

- Inicializar la ventana a partir de un *jframe*, o ventana contenedora, y añadirle un *jpanel*, el contenedor donde se añadirá los diferentes elementos, en el que cada nodo incluirá la imagen e información de su batería
- Se inicializan las cinco imágenes que usarán los nodos. Cuatro imágenes para indicar el nivel de la batería y otra imagen especial para que sea usada por el nodo 0 (gateway).

```
Icon = new ImageIcon(BatteryMonitor.class.getResource ("images/pila-
verde.png") , "Pila verde");
icon2 = new ImageIcon(BatteryMonitor.class.getResource("images/pila-
amarilla.png"), "Pila amarilla");
icon3 = new ImageIcon(BatteryMonitor.class.getResource("images/pila-
roja.png"), "Pila roja");
icon4 = new ImageIcon(BatteryMonitor.class.getResource("images/pila.png"),
"Pila vacia");
base=new    ImageIcon(    BatteryMonitor.class.getResource("images/base.png"),
"Estación base");
```

- Configurar el tamaño de la ventana según el número de nodos de la simulación para así adaptarse a las diferentes simulaciones que se puedan realizar. En esta versión de la aplicación, el número máximo de nodos que pueden mostrarse en la ventana creada (sin tener que usar scroll lateral) es de 512. Las diferentes configuraciones para los distintos números de nodos son las siguientes:

Numero de nodos	Tamaño de la ventana
nodos<=32	850x250
32<nodos<=64	850x600
64<nodos<=128	1100x700
128<nodos<=256	1360 x730
256<nodos<=512	1360x730

Tabla 30: Tamaños de la ventana

Dichos valores han sido escogidos para que al usuario le resulte cómoda la localización de cualquier nodo, y se adapte a los requisitos de usuario establecidos por el cliente. Estos valores además están tomados para adaptarse a una pantalla estándar a partir de 15,6".

Por ejemplo en el siguiente cuadro puede verse como se configura la pantalla para un número de nodos menor de 32:

```
if (nodes <= 32) {
        // Generates the window to less than 32 batteries
        panel = new JPanel(new GridLayout(3, 3));
        frame.add(panel);
        frame.pack();
        frame.setSize(850, 250);
}
```

Para generar la ventana se llama a esta clase desde el monitor de batería del nodo 0 para que se genere una única ventana en la que el resto de nodos podrán añadir la información correspondiente a su batería.

7. Evaluación

En este capítulo se describe la evaluación de la aplicación desarrollada que permite comprobar el funcionamiento correcto del sistema desarrollado de acuerdo a los requisitos establecidos por el usuario.

7.1 Validación de la interfaz

Las primeras pruebas a realizar son las relacionadas con el propio diseño de la interfaz. El objetivo principal de estas pruebas es:

- Comprobar la distribución de los nodos en la ventana generada.
- Verificar el uso o no de únicamente el scroll vertical en los momentos indicados.
- Comprobar la transición entre las diferentes imágenes que muestran la batería de los nodos.

En las pruebas realizadas se ha comprobado que se cumplen los requisitos establecidos por el cliente. Algunos ejemplos de la interfaz pueden verse en las siguientes imágenes tomadas de la aplicación durante su ejecución.

La Figura 25 representa la distribución de los nodos en la ventana cuando el número de nodos no excede de 32.

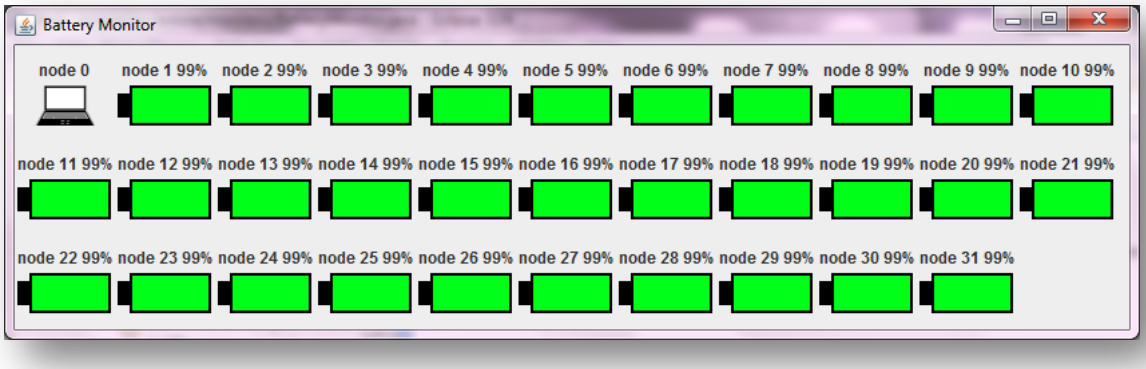

Figura 25: Ventana de 32 nodos

La Figura 26 se corresponde con el número máximo de nodos para los que está garantizado el único uso del scroll vertical para la localización de cualquiera de los nodos. Según lo establecido en los requisitos el número de nodos máximo es 512.

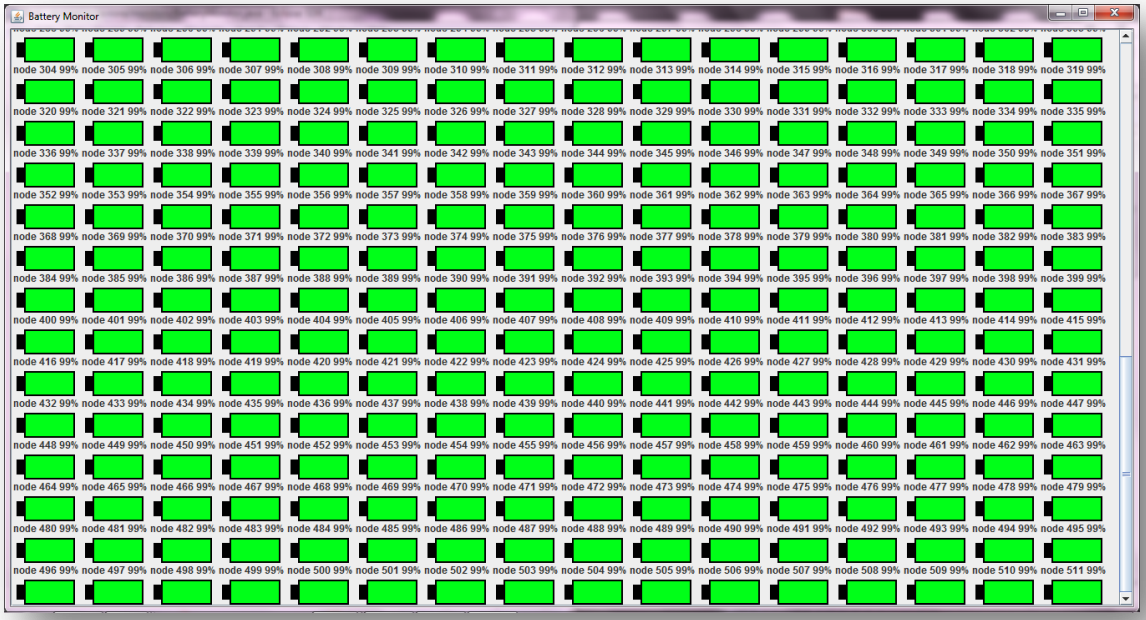

Figura 26: Ventana de 512 nodos

Finalmente, la Figura 27 representa como se vería la interfaz de la aplicación para un número mayor de nodos, en este caso 600. Como puede verse, en este caso es necesario utilizar tanto el scroll vertical como el horizontal para la localización por parte del usuario de cualquiera de los nodos.

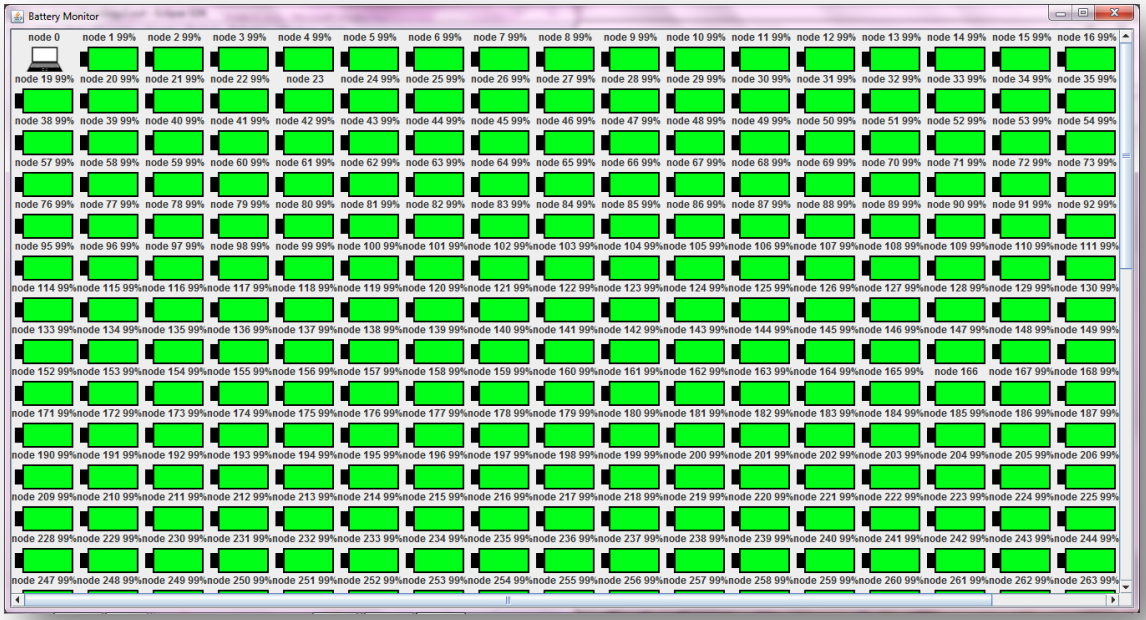

Figura 27: Ventana de 600 nodos

Una vez vistas todas estas ventanas puede verse que el sistema cumple con los requisitos especificados ya que, es capaz de mostrar de forma ordenada hasta 512 nodos utilizando únicamente el scroll vertical, tal y como se indicaba en los requisitos.

A continuación se incluyen diferentes imágenes en las que puede verse el cambio de las imágenes según el nivel de la batería. En la prueba realizada para estas imágenes los thresholds de control han sido establecidos en el 60% y el 30% del total de la batería.

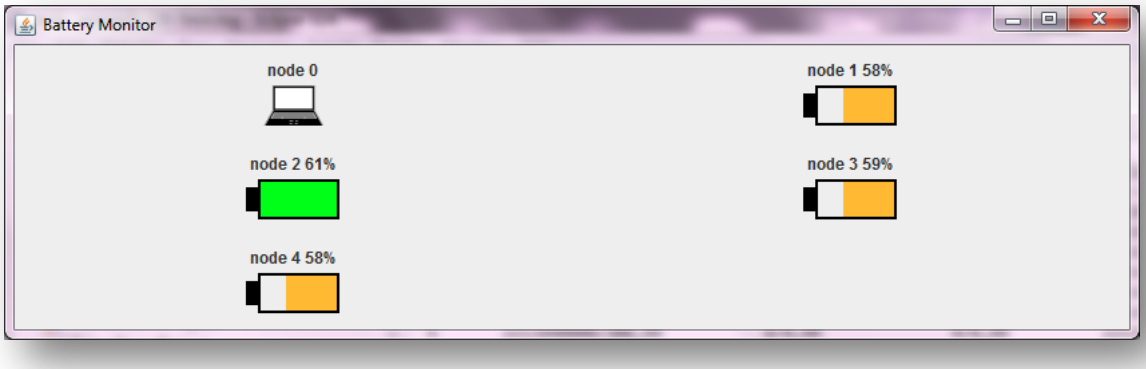

Figura 28: Primera transición

En la Figura 28 se ve la primera de las transiciones entre imágenes, al llegar el nivel de la batería al threshold superior, en los nodos 1, 3 y 4.

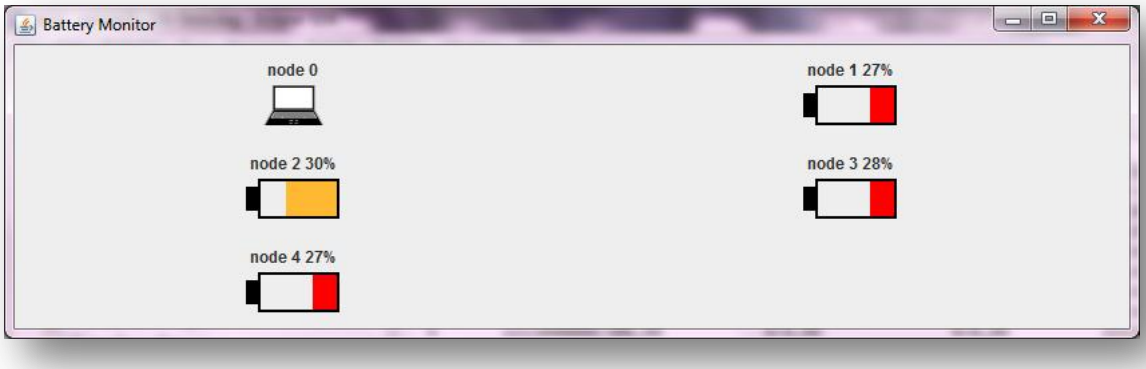

Figura 29: Segunda transición

Una vez que el nivel de la batería sea menor que el threshold inferior se realiza un nuevo cambio de imágenes representado en la Figura 29. Como se puede observar, las baterías de los nodos 1, 3 y 4 se muestran en rojo, mientras que la batería del nodo dos se muestra en ámbar.

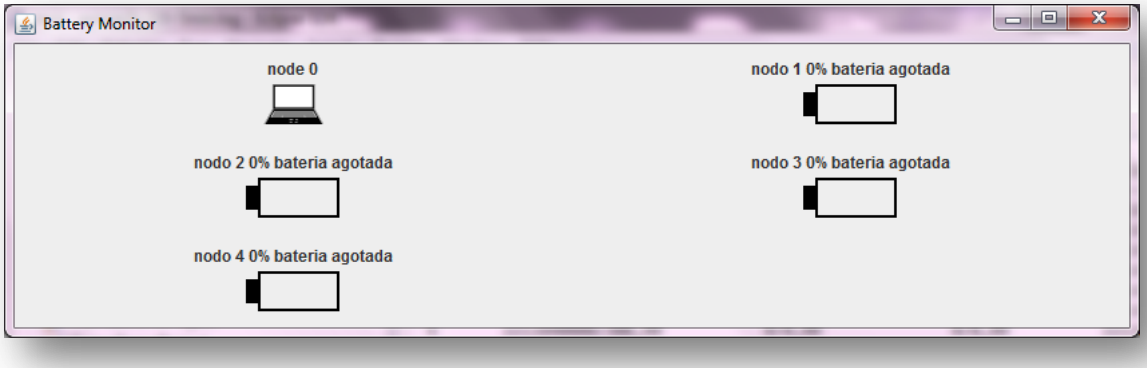

Figura 30: Estado final

Finalmente la Figura 30 representa el último cambio de imágenes cuando las baterías de los nodos ya están agotadas.

Puede comprobarse que las imágenes de las baterías van cambiando según lo establecido en los thresholds por lo que esta parte del sistema funciona correctamente.

7.2 Validación del funcionamiento del sistema

El siguiente aspecto a tratar y probar es el propio funcionamiento del sistema. Para ello será necesario comprobar que el gasto de energía de cada nodo durante cualquier momento de la simulación es el correcto. Para ello se realizarán varias simulaciones con diferentes tiempos y con los siguientes parámetros de simulación:

```
java avrora.Main -action=simulate -simulation=sensor-network -nodecount=1,4 -platform=micaz -input=elf -seconds=345600 -report-seconds -monitors=battery -batteryLevel=21600 -radio-range=75 -topology=static -topology-file=topology.out TestNetworkLPLBS.elf TestNetworkLPLNodes_50DC.elf
```

En la Tabla 31 se muestran los diferentes parámetros de simulación usados para facilitar la localización de los diferentes valores

Parámetro	Valor
Action	Simulate
Simulation	Sensor-network
Nodecount	1,4
Platform	Micaz
Input	Elf
Report	Seconds
Monitors	Battery
BatteryLevel	21600
Radio-range	75
Topology	Static
Topology-file	Topology.out

Tabla 31: Parámetros de las pruebas

El fichero de topología topology.out contiene los siguientes nodos y su localización:

```
node0 0   0   0
node1 50  0   0
node2 0   50  0
node3 -50 0   0
node4 0  -50  0
```

En esta topología el nodo gateway se encuentra en el centro (0,0,0) y el resto de nodos se ubican a su alrededor a la misma distancia pero en diferentes posiciones de los ejes.

Las pruebas realizadas han sido condicionadas por la propia aplicación Avrora, ya que para simulaciones con gran cantidad de nodos el rendimiento global de la aplicación depende del rendimiento de Avrora, el cual desciende en gran medida cuando el número de nodos aumenta de forma considerable. Es decir, para realizar una simulación de una sola hora, el tiempo real necesitado puede llegar a superar las 5 horas. Por lo que para las pruebas realizadas se han utilizado únicamente 5 nodos, ya que permiten obtener un rendimiento aceptable del sistema y sirven como simplificación del sistema global con más nodos. Sin embargo, el sistema funcionaría de manera correcta con un número mayor de nodos, si bien el tiempo de simulación aumentaría con el número de nodos.

A continuación se muestran los resultados de realizar las pruebas de simulación de 1 hora de duración con 5 nodos (4 nodos sensores y el nodo gateway). Los datos se refieren a 4 simulaciones distintas.

Nodo	Energía consumida en 1 hora			
	Simulación 1	Simulación 2	Simulación 3	Simulación 4
1	237.73 Julios	240,68 Julios	229.28 Julios	237.73 Julios
2	237.19 Julios	240.43 Julios	226.9 Julios	237.19 Julios
3	240,81 Julios	244,91 Julios	227,29 Julios	240,81 Julios
4	234,79 Julios	237,58 Julios	227,02 Julios	234,79 Julios

Tabla 32: Energía consumida en 1 hora

Como puede verse en la Tabla 32 el consumo de energía de cada nodo y en cada simulación son relativamente parecidos con una desviación muy pequeña, por lo que a simple vista, parece que la aplicación para simulaciones cortas funciona de una manera determinista.

Para seguir comprobando el funcionamiento de la aplicación se han repetido las pruebas pero esta vez durante dos horas. Los resultados obtenidos se muestran en la Tabla 33:

Nodo	Energía consumida en 2 horas			
	Simulación 1	Simulación 2	Simulación 3	Simulación 4

1	510 Julios	462,31 Julios	458,58 Julios	515,26 Julios
2	516,76 Julios	458,37 Julios	453,97 Julios	522,52 Julios
3	535,47 Julios	460,08 Julios	454,55 Julios	543,3 Julios
4	501,85 Julios	457,36 Julios	453,97 Julios	506,92 Julios

Tabla 33: Energía consumida en 2 horas

Combinando los resultados de las 2 primeras pruebas obtenemos el siguiente gráfico de dispersión en el que a partir de los datos obtenidos en las dos primeras pruebas obtenemos la función de la tendencia de los datos de forma aproximada, ya que como puede verse el consumo es prácticamente lineal con pequeñas variaciones. Esto quiere decir, que dada una misma topología y un conjunto de parámetros de simulación, la energía consumida es proporcional al tiempo de simulación.

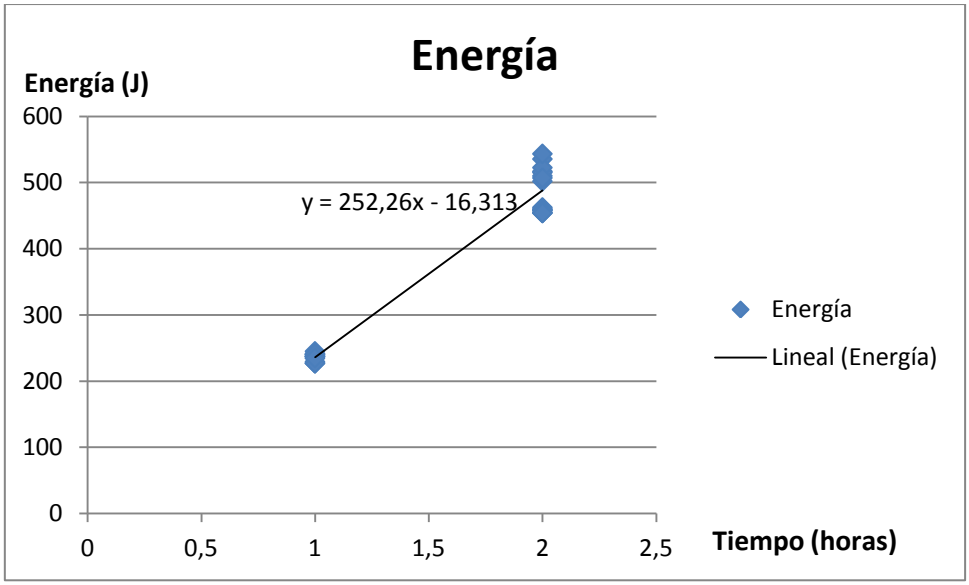

Figura 31: Gráfico de dispersión inicial

La ecuación obtenida representa de forma lineal el consumo medio de batería de todos los nodos. Es importante señalar que en esta ecuación el valor independiente representa las desviaciones o errores producidos y utilizados para ajustar la ecuación. Del gráfico, se obtiene que la ecuación para el cálculo de energía de un nodo en un tiempo x es:

$$y = 252{,}26 * x - 16{,}313$$

Con esta ecuación podemos calcular de forma teórica y aproximada la energía consumida por un nodo en cualquier instante de tiempo *t*. Además, sería posible calcular cuál es el tiempo necesario para que un nodo consuma toda su energía que representaría su tiempo de vida.

Así por ejemplo el tiempo total de vida de la batería sería:

$$y = 252{,}26 * x - 16{,}313 => x = \frac{y + 16{,}313}{252{,}26} = \frac{21600 + 16{,}313}{252{,}26} = 85{,}69\ horas$$
$$= 3{,}56\ dias$$

Para poder probar que esta ecuación es la adecuada y además el sistema desarrollado es correcto, es necesario realizar un número suficientemente elevado de evaluaciones que utilizan los mismos y distintos parámetros. Para evaluar este proyecto, en particular, hemos realizado un subconjunto de las pruebas necesarias pero razonables para poder validar el modelo. Así, a continuación presentamos los resultados de Avrora de otra serie de pruebas de 4 horas así como los resultados teóricos del gasto de energía de un nodo cualquiera. En primer lugar, calculamos de manera teórica el consumo energético de un nodo ejecutando en las mismas condiciones durante cuatro horas.

$$y = 252{,}26 * x - 16{,}313 = 252{,}26 * 4 - 16{,}313 = 992{,}73\ julios$$

Una vez realizadas las pruebas de 4 horas en Avrora obtenemos los resultados mostrados en la Tabla 34.

Nodos	Energía consumida en 4 horas			
	Simulación 1	Simulación 2	Simulación 3	Simulación 4
1	916,88 Julios	998,27 Julios	1023,19 Julios	992,65 Julios
2	908,04 Julios	1006,28 Julios	1036,95 Julios	999,09 Julios
3	908,98 Julios	1036,99 Julios	1076,86 Julios	1027,37 Julios
4	907,71 Julios	983,48 Julios	1007,34 Julios	978,42 Julios

Tabla 34: Energía consumida en 4 horas

Como puede verse estos datos se aproximan al valor teórico estimado que habíamos obtenido anteriormente, si bien es cierto que en algunos casos se ha observado que el valor de la simulación y el valor calculado tienen una desviación más alta. Por tanto, para intentar obtener una ecuación más precisa, se combinan los nuevos datos con los datos anteriores en una nueva gráfica de dispersión.

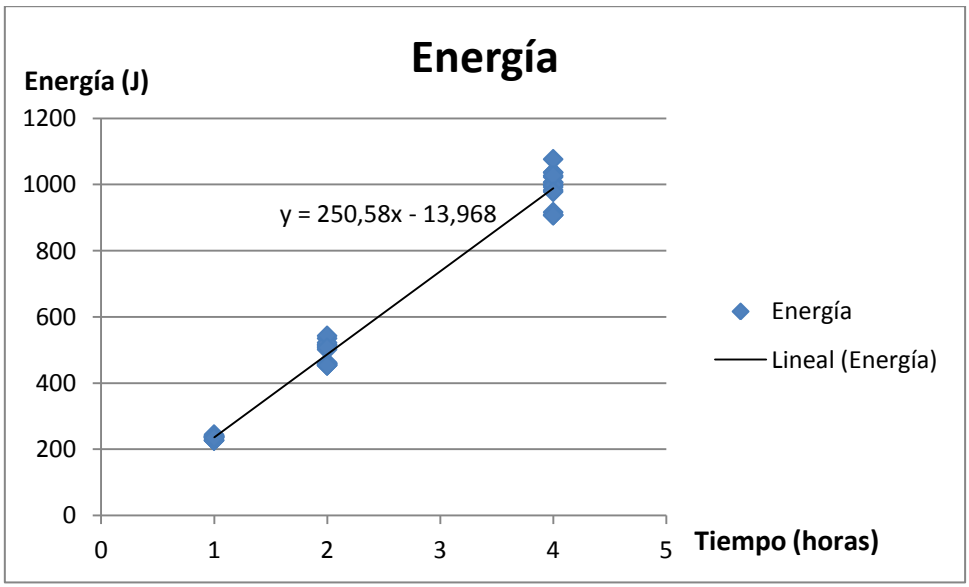

Figura 32: Grafico de dispersión tras 4 horas

Desde la Figura 32, se deduce que la nueva ecuación de energía es:

$$y = 250{,}58 * x - 13{,}968$$

Con esta nueva ecuación volvemos a obtener el tiempo total necesario para gastar la batería de un nodo.

$$y = 250{,}58 * x - 13{,}968 => x = \frac{y + 13{,}968}{250{,}58} = \frac{21600 + 13{,}968}{250{,}58} = 86{,}25\ horas$$
$$= 3{,}59\ dias$$

Este nuevo cálculo no difiere mucho del cálculo anterior, por lo que podemos determinar que el tiempo total necesario rondará aproximadamente las 86 horas. Con este dato podemos realizar una nueva simulación en la que el tiempo de ejecución sea por ejemplo 96 horas. Durante la ejecución de esta prueba la idea es que se consuma totalmente la energía de todas las baterías y además se dispone de un margen de tiempo en el caso de que en algún punto de la simulación el gasto de energía se ralentice.

Los tiempos obtenidos en esta nueva simulación son los mostrados en la Tabla 35:

Nodos	Tiempo
1	342674,10 seg
2	342653,00 seg
3	342651,80 seg
4	342670,70 seg

Tabla 35: Tiempos

Obtenemos el último gráfico de dispersión con los datos de las 4 series de pruebas.

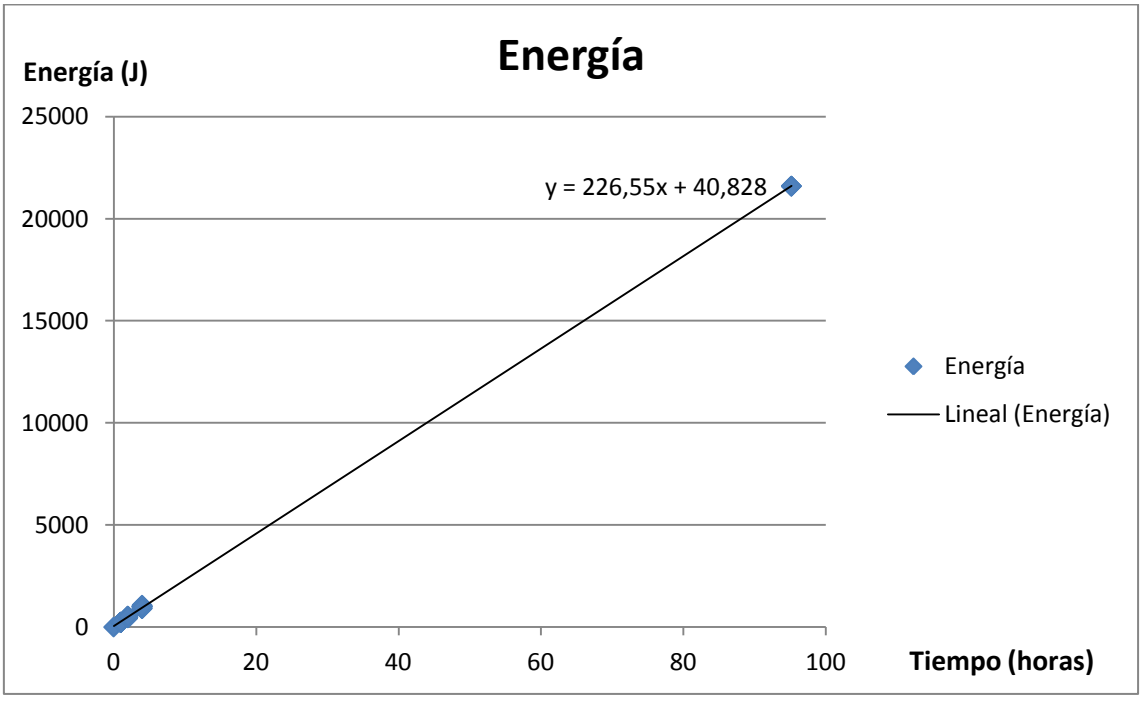

Figura 33: Grafico de dispersión final

Desde la Figura 33, podemos concluir que la última ecuación para obtener el valor de la energía es:

$$y = 226{,}55 * x + 40{,}828$$

Con todos los datos recopilados en las diferentes pruebas podemos realizar una gráfica comparativa entre:

- La estimación hecha con los datos de las simulaciones de 1, 2 y 4 horas (y = 250,58x - 13,968).
- Los resultados de las simulaciones con Avrora (y = 226,55x + 40,828).
- Gasto de energía lineal considerando el mínimo gasto de energía de la simulación de 1 hora (235,94 julios).

En el eje X se representa el tiempo de la simulación en horas y el eje Y se corresponde con el consumo de energía en Julios.

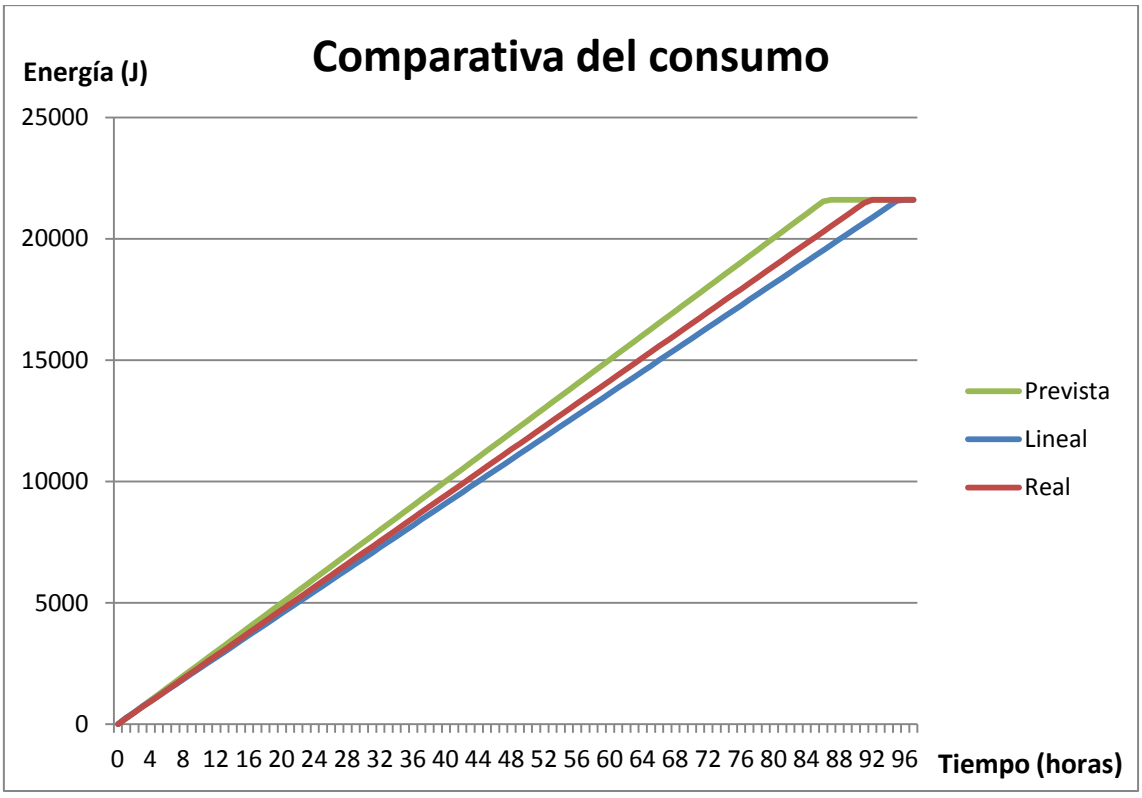

Figura 34: Comparativa del consumo

Una vez realizadas todas estas pruebas puede verse que el funcionamiento del sistema desarrollado es el deseado ya que los resultados obtenidos se ajustan a los datos esperados. Como ya se ha dicho estas pruebas han sido realizadas con un número pequeño de nodos para mostrar la validez y el correcto funcionamiento del sistema desarrollado. Sin embargo, el sistema puede aceptar cualquier número de nodos hasta un valor de 512. En este caso, los resultados de energía obtenidos (aunque estén correlacionados con los datos ya obtenidos) serán mayores, ya que al haber un mayor número de nodos el intercambio de mensajes será mayor, con el consiguiente aumento del gasto de energía y la reducción del tiempo de vida de las baterías.

Como complemento se han realizado pruebas con 100 nodos durante únicamente 20 minutos, con un tiempo real de simulación cercano a 7 horas. En la Tabla 36 se muestran los resultados de energía obtenidos para algunos nodos:

Nodo	Posición	Energía consumida(J)
1	(50,0,0)	117,46
18	(0,-100,0)	116,11
55	(-175,0,0)	114,5
74	(-100,60,0)	115,32
94	(190,-180,0)	114,23

Tabla 36: Simulación con 100 nodos

La Figura 35 representa la topología usada en esta simulación. Los puntos rojos representan los nodos indicados en la Tabla 36, y el nodo gateway se representa como el cuadrado central ubicado en (0,0,0).

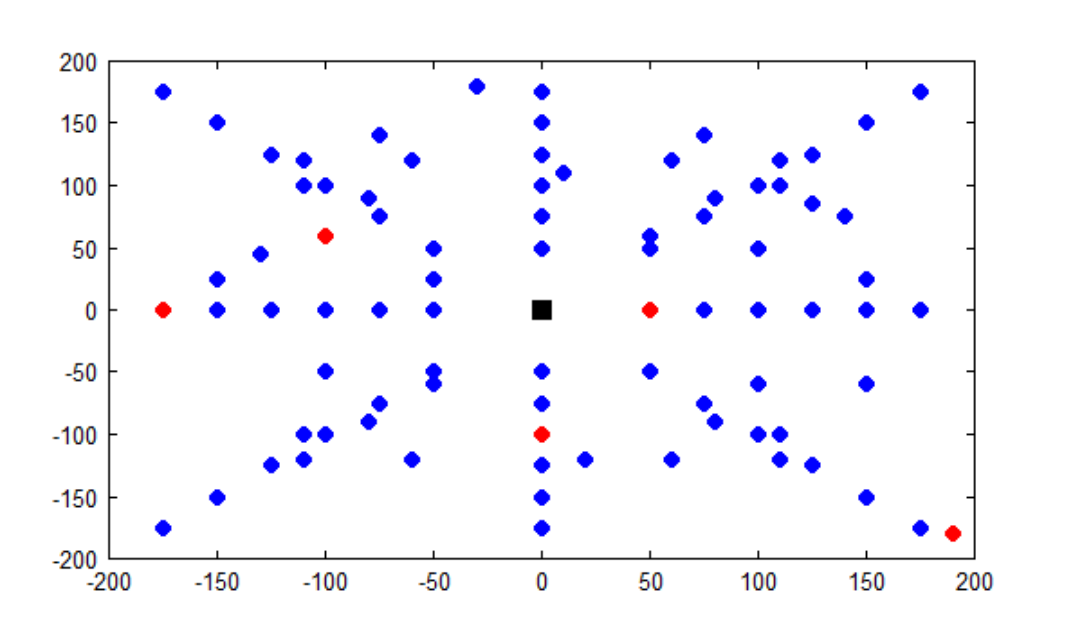

Figura 35: Topología con 100 nodos

Como puede observarse de los resultados de energía, no se llega a poder apreciar de forma clara las diferencias entre los consumos de los distintos nodos. Esto es debido fundamentalmente a que el tiempo de simulación de esta prueba no es muy grande. Si aumentamos el tiempo de simulación las diferencias de consumo entre los nodos se harían más evidentes. Lo que sí se puede comprobar es que normalmente los nodos hoja (nodo 55 y nodo 94) tienen un consumo menor que los nodos interiores (nodo 1, nodo 18 y nodo 64), debido a que no realizarán la retransmisión de los mensajes de sus nodos vecinos, y por tanto el consumo de los nodos hoja puede ajustarse más a un consumo lineal que los nodos interiores, que será más impredecible.

Finalmente, es importante destacar que aunque a primera vista pueda parecer que el gasto de energía en este tipo de dispositivos es lineal y constante, realmente no es así, ya que el gasto de energía depende de muchos factores, como pueda ser el uso de la radio. Predecir el consumo de energía de una red de sensores, o incluso de un único mote, es un problema altamente complejo, ya que las variaciones que se producen se pueden considerar realmente aleatorias e imposibles de predecir. Además en las redes con un número mayor de motes, esas variaciones son todavía mayores por lo que el gasto de energía será más difícil de predecir, aunque siempre se podrán usar modelos lineales que aproximen el gasto de energía, suponiendo siempre el peor caso.

7.3 Limitaciones

En este apartado se incluye cuales han sido las principales limitaciones que han condicionado el desarrollo de las pruebas necesarias para validar el sistema desarrollado.

El primer punto a tener en cuenta en la realización de la evaluación es que las pruebas han venido condicionadas por un bug presente en Avrora (16). Este bug es poco frecuente y durante la realización del proyecto solo se ha dado en casos muy puntuales y bajo unas condiciones concretas. Dicho error se basa en un problema de la radio que produce un desbordamiento de los bytes a transmitir provocando que se lance una excepción que detiene la simulación del nodo en el que se haya producido el error.

Otro elemento importante a tener en cuenta es la topología de la red y el número de nodos, ya que como se ha dicho el gasto de energía de la red es dependiente de estos factores, y por tanto para las pruebas en las que era necesario calcular el tiempo de simulación para comprobar si se llega a agotar la batería de los nodos son factores que pueden determinar si la prueba es válida o no.

El último punto es la incompatibilidad en ciertas situaciones especiales, del nuevo monitor de batería (battery) con el monitor de energía incluido en Avrora (energy). Esto se debe, a que tanto el monitor de batería como el monitor de energía comparten algunas funciones, como puede ser desconectar los nodos que han agotado su batería. Además ambos disponen de comandos para asignar el nivel de batería a los nodos, el monitor de batería usa el comando –batteryLevel y el monitor de energía usa el comando -battery. La incompatibilidad mencionada se produce cuando se ejecutan ambos monitores juntos (battery y energy), y en el monitor de energía se usa el comando –battery para asignar la energía de los nodos mediante ese monitor. El problema reside en que el monitor de batería que hemos desarrollado (battery) asume que el nodo 0 es el nodo gateway, y que dispone de un sistema de alimentación energético infinito. Sin embargo, en el monitor de energía de Avrora no se realiza tal asunción, y por tanto cuando se consume la energía establecida mediante el comando –battery se desconecta el nodo (al igual que ocurre con el resto de los nodos), lo que provoca un conflicto de funcionalidades. Pese a todo esto, si se usan ambos monitores

juntos sin el uso de de dicho parámetro (–battery) por parte del monitor de energía de Avrora no se produce ningún error y los monitores son totalmente compatibles.

8. Conclusiones

En este capítulo se tratan varios aspectos relacionados con la finalización del proyecto como es la creación de un presupuesto para el proyecto o las futuras líneas de desarrollo.

8.1 Revisión de los objetivos

En este apartado se comprobará si se han cumplido los objetivos especificados en los primeros capítulos de este documento.

Este proyecto buscaba ampliar las funcionalidades de la plataforma Avrora, mediante la inclusión de un nuevo monitor, con el que poder comprobar en cualquier momento a lo largo de una simulación, cual es el nivel de las baterías de todos los nodos implicados en la simulación.

Para ello era necesario desarrollar una nueva interfaz gráfica en la que mediante texto e imágenes se muestre el nivel de energía de las baterías de los nodos de la simulación de Avrora. Además se debe generar un fichero de registro o *log* con los datos del uso de las baterías por parte de los nodos.

Los objetivos descritos se han conseguido realizar con éxito, ya que el nuevo monitor de batería desarrollado cubre estas necesidades de manera satisfactoria. El monitor, además de realizar las tareas pedidas, muestra la información de manera cómoda y sencilla para el usuario, lo que le facilita la localización de la información de forma rápida.

Todo esto, además, se ha hecho sin sobrecargar de manera perceptible el rendimiento de la propia plataforma Avrora, consiguiendo otro de los objetivos propuestos al comienzo del proyecto.

8.2 Líneas futuras de trabajo

En esta sección, se enumeran las posibles mejoras o complementos que se podrían desarrollar como añadidos al proyecto, para aumentar o mejorar sus funcionalidades, y que no están incluidas en el proyecto pedido.

- El principal elemento a desarrollar en el futuro, y que ampliaría en gran medida las funcionalidades del nuevo monitor, consiste en añadir soporte para nodos sensores con fuentes de energía alternativas como placas solares. De esta forma se ampliaría la capacidad de Avrora para simular los elementos más actuales de las redes de sensores como los motes Eko

(5), que disponen de placas solares para recargar las baterías que utilizan. Este sistema deberá ser capaz de hacer una distinción entre las horas nocturnas y diurnas de la simulación, con el objetivo de poder ajustar el funcionamiento de la batería para ser recargada durante las horas de sol (e.g. desde las 6 de la mañana hasta las 7 de la tarde, dependiendo de la posición geográfica entre otros factores), y durante las horas de baja producción hacer uso únicamente de la batería (e.g. desde las 7 de la tarde hasta las 6 de la mañana).

- Ampliar el número de nodos que la aplicación puede mostrar de forma ordenada. Actualmente se pueden mostrar como máximo 512 nodos haciendo uso únicamente del scroll vertical, pero se podría intentar ampliar dicho numero, de forma que se puedan intentar simular las redes de sensores más extensas.
- Al aumentar el número de nodos de las simulaciones, también aumentan el número de operaciones a realizar por cada nodo, y unido esto a los cálculos necesarios para la interfaz gráfica, el rendimiento global del sistema puede disminuir, por lo que se podría intentar aumentar el rendimiento del sistema. Este problema, como ya se ha visto en el capítulo Evaluación, es intrínseco de Avrora, por lo que los cambios que se podrían realizar, serían solo en el nuevo monitor desarrollado para que el rendimiento global se vea aún menos afectado.

8.3 Consideraciones legales

A continuación se presentan los aspectos legales relacionados con el proyecto:

El proyecto desarrollado es un complemento añadido a la plataforma Avrora, por lo que al igual que esta estará bajo una licencia libre Open Source (17).

El sistema desarrollado no dispone de mucha interacción directa con el usuario, y al no almacenar ningún tipo de información no es necesario que cumpla las restricciones básicas respecto a la protección de datos como puede ser la normativa de la Agencia Española de Protección de Datos (18).

8.3 Planificación

En este apartado se incluye la planificación desarrollada para realización de este proyecto. La Figura 36 representa el diagrama de Gantt en el que se muestra como se han distribuido las tareas a lo largo del desarrollo del proyecto, desde septiembre de 2012 hasta mayo de 2013.

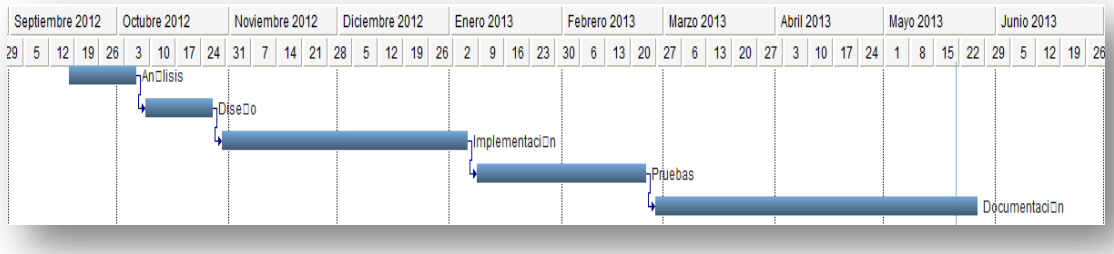

Figura 36: Diagrama de Gantt

La metodología utilizada es una metodología en cascada, por lo que para empezar una nueva fase es necesario haber concluido totalmente la fase anterior. A continuación se describen las fases en las que se ha dividido el proyecto y el tiempo asignado a cada una:

- Análisis: en esta fase se procede al estudio del problema propuesto y sus posibles soluciones. La duración de esta fase es de 3 semanas.
- Diseño: aquí se presentan diferentes propuestas y se escoge la solución más adecuada. Esta fase tiene una duración de unas 3 semanas.
- Implementación: Desarrollo e implantación del sistema escogido en la fase de diseño, por lo que su duración es larga, unas 10 semanas.
- Pruebas: en la fase de pruebas se evalúa que el sistema desarrollado responde al funcionamiento deseado y cumple los objetivos buscados. Tiene asignada una duración de 7 semanas.
- Documentación: Creación de todos los documentos necesarios en el desarrollo del proyecto. Esta fase es la más larga con una duración de 13 semanas.

8.4 Presupuesto

En esta sección se muestra el presupuesto necesario para el desarrollo de este proyecto. El presupuesto se ha dividido en recursos humanos y recursos materiales y en cada uno se analizarán los costes necesarios.

8.4.1 Recursos humanos

Hay que tener en cuenta algunos elementos a la hora de calcular los costes humanos:

- Para el desarrollo del proyecto se ha seguido una metodología en cascada compuesta por las siguientes fases:
  - Análisis.
  - Diseño.
  - Implementación.
  - Pruebas.

- Documentación.

- Como el proyecto ha sido desarrollado por una única persona se ha establecido un precio/hora medio igual a 30€/hora.
- Se ha necesitado unas 36 semanas y un trabajo medio de 20 horas semanales para la finalización del proyecto, lo que suma un total de 720 horas de trabajo.

A continuación se muestran los gastos del proyecto relacionados con los recursos humanos.

	Precio/Hora	Horas	Coste (€)
Análisis	30	70	2100
Diseño	30	70	2100
Implementación	30	190	5700
Pruebas	30	210	6300
Documentación	30	180	5400
Total		720	21600

Tabla 37: Recursos humanos

8.4.2 Recursos materiales

Los gastos de recursos materiales se muestran en la Tabla 38. En este apartado consideramos tanto el ordenador en el que ha sido desarrollado el proyecto como la conexión a internet necesaria para buscar recursos o comunicarse con el cliente.

	Cantidad	Coste	% Uso dedicado	Dedicación	Periodo de depreciación	Coste Imputable
Ordenador portátil	1	750 €	100	9 meses	60 meses	112,50 €
Total						112,50 €

Tabla 38: Recursos materiales

El coste imputable se obtiene mediante la siguiente fórmula:

$$\frac{A}{B} * C * D$$

Donde:

- A= nº de meses desde la fecha de facturación en que el equipo es utilizado
- B= periodo de depreciación
- C= Coste del equipo (sin IVA)
- D= % del uso que se dedica al proyecto

8.4.3 Otros costes directos

En la Tabla 39 se incluyen los otros gastos relacionados con el proyecto:

	Coste	Cantidad	Subtotal

Conexión Internet	45 €/mes	9 meses	405 €
Total			405€

Tabla 39: Otros costes directos

8.4.3 Coste total

En la Tabla 40 se muestra el coste total de la realización del proyecto, obtenido a partir de los costes antes calculados.

	Coste
Recursos Humanos	21600€
Recursos Materiales	112,50€
Costes Directos	405 €
Total	22117,50€

Tabla 40: Coste total

8.5 Valoración personal

Una vez finalizado el proyecto y revisado el trabajo realizado, puedo decir que éste ha sido una gran oportunidad para poder conocer y estudiar nuevas tecnologías con las que no había tenido la oportunidad de trabajar.

La primera de ellas son las redes de sensores inalámbricas. Esta tecnología era desconocida para mí, y el desarrollo de este proyecto me ha mostrado el funcionamiento y aplicaciones de este tipo de sistemas, cada vez más utilizados debido a sus grandes posibilidades.

Además he tenido la oportunidad de trabajar con Java Swing, un API que ya conocía pero con el que nunca había realizado ningún proyecto.

Personalmente, ésta ha sido una gran oportunidad académica, ya que me ha permitido poner en práctica gran cantidad de los conocimientos adquiridos durante la carrera, y prepararme en cierto grado para el mundo laboral.

Bibliografía

1. Real Academia Española. [En línea] http://www.rae.es/.

2. *Wireless sensor networks: a survey.* **Akyildiz, I.F., y otros.** s.l. : Elsevier, March de 2002, Computer Networks, Vol. 38, págs. 393-422. (http://www.sciencedirect.com/science/article/pii/S1389128601003024).

3. *A novel methodology for the monitoring of the agricultural production process based on wireless sensor networks.* **Escolar Díaz, Soledad, y otros.** s.l. : Elsevier, May de 2011, Computers and Electronics in Agriculture, Vol. 76, págs. 252-265. (http://www.sciencedirect.com/science/article/pii/S0168169911000548).

4. *Component-based software systems for smart environments.* **Herring, C. y Kaplan, S.** Octubre de 2000, Personal Communications, Vol. 7, págs. 60-61. (http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=878541&isnumber=19016).

5. Memsic. [En línea] http://www.memsic.com/.

6. Atmel. [En línea] http://www.atmel.com.

7. TinyOS Home Page. [En línea] http://www.tinyos.net/.

8. nesC. [En línea] http://nescc.sourceforge.net/.

9. Avrora. [En línea] http://compilers.cs.ucla.edu/avrora/.

10. AEON energy analysis. [En línea] http://ds.informatik.rwth-aachen.de/research/projects/aeon/energyAnalysis.htm.

11. TOSSIM: A Simulator for TinyOS Networks. [En línea] http://www.cs.berkeley.edu/~pal/pubs/nido.pdf.

12. Castalia. A simulator for WSNs. [En línea] http://castalia.npc.nicta.com.au/.

13. OMNeT++. [En línea] http://www.omnetpp.org/.

14. *Optimization of Quality of service in Wireless Sensor Networks Powered by Solar Cells.* **Escolar Díaz, Soledad, Chessa, Stefano y Carretero Pérez, Jesús.**

15. Métrica v.3. [En línea] http://administracionelectronica.gob.es.

16. Avrora CC2420 and USART bugfixes and new problems. [En línea] http://lists.ucla.edu/pipermail/avrora/2009-September/001208.html.

17. Open source. [En línea] http://opensource.org/.

18. Agencia Española de Protección de Datos. [En línea] http://www.agpd.es/.