SiMSI

Sistema de Monitoreo de Sensores Inalámbricos

Agustín Caldevilla
Carlos Protasi
Rodrigo Sánchez
Guillermo Spiller

Tutores:
Leonardo Steinfeld
Pablo Mazzara

Documento de Proyecto de Grado, Ingeniería Eléctrica

Facultad de Ingeniería
Universidad de la República
Montevideo,URUGUAY

30 de abril de 2009

Prefacio

El presente documento constituye la documentación del Proyecto de Fin de Carrera titulado "Sistema de Monitoreo de Sensores Inalámbricos" (SiMSI), para el Instituto de Ingeniería Eléctrica, Facultad de Ingeniería de la Universidad de la República, Montevideo - Uruguay. Los integrantes del grupo de proyecto son Agustín Caldevilla, Carlos Protasi, Rodrigo Sánchez y Guillermo Spiller.

El proyecto fue desarrollado en el período entre Marzo de 2008 y Mayo de 2009. El Ing. Roque Gagliano fue el tutor en los primeros meses hasta comenzar el año 2009 donde el Ing. Leonardo Steinfeld asumió la tutoría. Durante todo el período el Ing. Pablo Mazzara contribuyó constantemente al desarrollo del mismo y como co-tutor en la última etapa.

El presente proyecto desarrolla un sistema de monitoreo para sensores inalámbricos, donde un usuario remoto es capaz de comunicarse con los sensores y analizar los datos que éstos le envían. La idea original es aplicar este sistema en situaciones donde los sensores estén en condiciones adversas en cuanto a energía eléctrica, clima y otros factores. Un ejemplo sería en aplicaciones agrarias donde el usuario puede instalar los sensores en sus plantaciones y monitorear los resultados desde su casa en el campo o desde cualquier otro dispositivo con conexión a Internet.

El trabajo se divide en cinco partes que describimos brevemente. Primero una descripción e introducción más detallada sobre los objetivos que busca este proyecto y las características del mismo. Luego se ahonda en temas referentes a la red como ser: el envío de datos al servidor, la configuración de red considerada y una breve reseña sobre la red formada por los motes. En la tercera parte se analizan los dispositivos principales sobre los cuales trabajamos en este proyecto: router y servidor. Se explica cómo operamos en cada uno, sus funcionalidades, software utilizado y desarrollado e integración de los mismos a todo el sistema. La cuarta parte describe la interfaz del usuario y cómo son procesados los datos recibidos. Finalmente se formulan conclusiones, aspectos a mejorar y se analizan las oportunidades a futuro que presenta este trabajo.

Agradecimientos

Antes que nada queremos agradecer a Patty, Eve, Carlos, Efe, Alicia, Gonzalo, Karen, Estefa, Beatriz (especialmente por los sandwiches calientes!), Jorge, Javi desde Francia, Juan Carlos, Ivonne, Fiorella, Carolina y Maru por el cariño, la paciencia y el apoyo incondicional a lo largo de todo este período; sin ellos esto no hubiera sido posible.

A nuestros amigos por todas las cervezas que les debemos.

A todos los que nos dieron una mano cuando la precisamos. Entre ellos queremos destacar a Bart, Fiorella Haim, Ricky y Tornado, quienes cada uno en su área nos brindaron apoyo y conocimiento de muchísimo valor.

A Roque Gagliano por proponer la idea y ser nuestro tutor en los primeros meses de trabajo.

A Leonardo Steinfeld, quien como tutor estuvo desde el comienzo hasta el final siempre que lo precisamos. Especialmente por asumir la tutoría en un momento difícil y de forma inesperada para él.

A Pablo Mazzara que realmente es un referente para todos nosotros por su conocimiento, paciencia, dedicación y apoyo constante.

Resumen Ejecutivo

El presente proyecto consiste en el desarrollo de un sistema de monitoreo a distancia de una red de sensores inalámbricos que permitirá a un usuario remoto saber el estado e historial de los sensores en todo momento. El sistema se enfoca en el envío de la información recabada por los sensores hasta un servidor central, la generación de una base de datos y diseño y programación de la interfaz gráfica. Aplicando esta tecnología en el sector agropecuario, se permite satisfacer la creciente necesidad de tener un mayor control por parte del productor/inversor del estado de su producción y tomar las acciones necesarias en el momento requerido. Esto permite maximizar la productividad y tener un mayor control y trazabilidad de los productos o activos monitoreados.

Entrando más en detalle sobre los beneficios que el sistema puede brindar a un productor, se explica brevemente cómo implementarlo y qué información se puede obtener con él.

En primer lugar el productor debe adquirir la cantidad de motes necesaria para distribuir sobre la superficie a controlar y monitorear. Los motes son dispositivos electrónicos que pueden integrar sensores de diferentes tipos e implementar una comunicación inalámbrica, transportando así la información recabada por cada sensor. Una de sus principales ventajas es su bajo consumo, donde con dos pilas AA por mote se puede alcanzar una autonomía mayor a un año. Los sensores integrados pueden variar en distintos tipos pero típicamente los hay de temperatura, humedad y luminiscencia (esto se encuentra en constante desarrollo). Deben colocarse en algún tipo de encapsulado que los proteja de las inclemencias del clima si éstos van a ser colocados a la intemperie. Una vez instalados, los motes formarán una red tipo mesh donde enviarán la información a sus motes vecinos hasta alcanzar al mote central que recolecta los datos del resto. Típicamente el alcance de cada sensor es de entre cincuenta y cien metros, dependiendo de las condiciones del medio. El productor debe tener este punto en consideración a la hora de distribuirlos en la superficie.

Una vez distribuidos los motes, se conecta el mote central vía USB a un router. El router sí debe estar conectado a la corriente eléctrica, detalle no menor en el campo. Sin embargo, dada la flexibilidad y alcance que se puede obtener con la red de motes esto no debería ser un inconveniente. El router debe tener instalado la imagen del sistema operativo (firmware) desarrollado en este proyecto, de forma de poder procesar correctamente la información que recibe y envía el servidor o mote constantemente. El sistema está preparado para soportar más de un router y en ese caso será necesario instalar el firmware en cada uno de ellos. Se deberá hacer una planificación de cuántos routers instalar, dónde y a qué frecuencia operarán, para facilitar al usuario la comprensión de los datos y obtener una red de monitoreo robusta en caso de que algún router tenga un desperfecto.

Finalmente el servidor central deberá estar ubicado en un lugar protegido, como ser el casco de la estancia, y debe tener conexión hacia al menos uno de los routers instalados, dependiendo de la topología de la red. Se instalará un programa en el servidor que no sólo permitirá monitorear los sensores localmente, sino que si se posee una conexión a Internet, será posible monitorear los mismos desde un equipo remoto. El software transforma el servidor central en un servidor web que permitirá al usuario acceder al sistema desde cualquier ubicación a través de cualquier computador o teléfono celular conectado a Internet.

Una vez que esté todo instalado el usuario puede finalmente ingresar en el sistema. Hay tres tipos de usuario: básico, administrador o personalizado. Inicialmente el administrador debe conectarse para configurar el programa y adaptarlo de acuerdo a los requerimientos particulares del productor. Algunas funciones administrativas a destacar son la configuración de la frecuencia con la cual son recolectados los datos, configuración de equipos,

usuarios y visualización de logs. Quizá una de las características más interesantes del sistema es que el usuario puede ver el status de los sensores directamente en un mapa real (descargado gracias a Google Maps). Además de ver el status de cada mote, puede ver alarmas o recibirlas directamente en su correo electrónico o celular si así lo desea. Gracias a esta característica el productor puede tomar acciones a tiempo si considera que su producción lo requiere. Finalmente se pueden configurar reportes donde el usuario puede evaluar los datos recabados a lo largo de diferentes períodos de tiempo como ser ese mismo día o el último año entre otros. Estos reportes configurados por el propio usuario, despliegan gráficas que permiten no sólo ver la evolución de los datos o detectar anomalías, sino también planificar futuros emprendimientos en base a la experiencia ya obtenida.

Cabe destacar la gran flexibilidad que posee el sistema que le permite adaptarse a requerimientos muy diversos. Si bien no es imprescindible, es recomendable que la puesta a punto, instalación y configuración sea realizada por un técnico instalador con experiencia y conocimiento del producto. Esto permitiría al usuario maximizar los beneficios y sacar un rápido provecho de los mismos.

Tabla de Contenidos

Capítulo 1

Introducción del Problema

1.1. Motivación

El presente proyecto nace de la necesidad de monitorear cultivos, plantaciones u otras aplicaciones desde sitios remotos, facilitando las tareas y el control de un productor, que por diversos motivos no puede permanecer constantemente en el lugar donde se encuentra su producción. Particularmente existe un gran potencial para esta aplicación en el Uruguay y resto de los países de América del Sur donde las principales fuentes de ingreso provienen de la generación de alimentos y materias primas. El avance de la tecnología y las telecomunicaciones permiten hoy en día tener un mayor control y seguimiento de los productos, pudiendo registrar la evolución de los mismos desde su creación hasta que llegan a las manos del consumidor final. El sistema implementado puede ser utilizado en aplicaciones agronómicas donde las condiciones climáticas y energéticas generalmente no son las propicias para el uso de la electrónica y las telecomunicaciones. Al día de hoy cada vez es más fácil dotar de ciertas tecnologías a las regiones más remotas y aisladas del país y creemos firmemente que si logramos impulsar estos avances lograremos productos de mayor calidad.

El sistema SiMSI permite a los usuarios monitorear y rastrear sus productos, y en función de ello, tomar acciones en el momento oportuno, evitando pérdidas y logrando su desarrollo en condiciones óptimas. La información puede ser recopilada y podrán correlacionarse los productos obtenidos con la historia relevada por los distintos sensores. Cabe destacar que la interfaz gráfica es altamente amigable e intuitiva y cualquier operador PC básico será capaz de trabajar sin ningún inconveniente con el sistema.

Le ha costado mucho a las telecomunicaciones abrirse camino en el campo, sin embargo cada vez más son reconocidas sus ventajas y utilidades. Esperamos con esto ser parte de este desarrollo donde la ciencia, la tierra y el hombre se unen para lograr beneficios mutuos.

1.2. Objetivos

El objetivo del proyecto es desarrollar un sistema de monitoreo a distancia de una red de sensores inalámbricos que utiliza el protocolo IEEE 802.15.4 LR-WPAN (Low Rate, Wireless Personal Area Network). Los sensores están embebidos en un dispositivo electrónico, llamado mote, que implementa este protocolo de comunicación y logra a pesar de su bajo consumo (dos pilas AA por mote tienen una duración típica mayor a un año) un alcance significativo de entre cincuenta y cien metros dependiendo del medio y para una potencia típica de 1mW (0 dBm) de salida. Específicamente los motes considerados para este proyecto son los motes desarrollados por las empresas Moteiv y Crossbow.

El área de cobertura se divide en pétalos o clusters, dentro de los cuales se realiza la comunicación utilizando el protocolo 802.15.4 hacia un nodo puente. Dichos nodos se comunican entre sí utilizando el protocolo 802.11 a/b/n realizando la interconexión de pétalos y finalmente enviando la información a un servidor central que

procesará y almacenará los datos. Se puede observar un esquema de esta red en la figura 1.1.

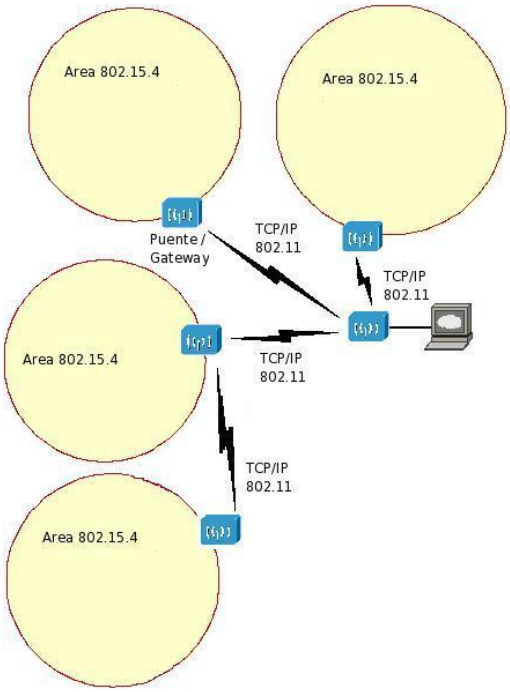

Figura 1.1: Esquema de la red del proyecto

Los nodos puente son routers con soporte DD-WRT, Open-WRT o similar y se analizará cuál es el más indicado usar en este caso (siempre teniendo en cuenta que sea software libre). Se utilizará un router con interfaz USB para colocar un mote central que será el responsable de recolectar la información enviada por el resto de los motes pertenecientes a su pétalo. Deberá desarrollarse un driver que permita obtener los datos del mote central para luego transmitirlos hacia el servidor central.

Se considerarán tres posibles formas para la comunicación entre el puente y el servidor central:

1. SNMP, protocolo de gestión. Tiene la ventaja de ser un estándar conocido y simple, únicamente habría que implementar las variables correspondientes.
2. XML, usan web-services. Se puede simplemente intercambiar información de configuración.
3. Túnel, se podría hacer un túnel sobre TCP/IP.

Se estudiará cuál de las tres opciones es la más conveniente y luego se implementará en el sistema.

Finalmente se trabajará sobre el servidor central que recibe los datos y los almacena en una base de datos con la cual el usuario final podrá interactuar mediante un portal web. Dependiendo de cómo se implemente la comunicación entre router y servidor, se desarrollará además un software (demonio) que hará de interfaz entre la red de sensores y la base de datos. Además, el servidor central funcionará como servidor web permitiendo al usuario conectarse al sistema en forma remota, desde cualquier computador o dispositivo con conexión a Internet.

El proyecto incluye:

- Estudiar las soluciones disponibles para la implementación de una red basada en 802.15.4.

- Estudiar qué Sistema Operativo utilizar en cada nodo puente (router).
- Implementar Driver para comunicación router - mote en sistema operativo escogido.
- Implementar conexión al servidor central mediante una de las tres opciones descriptas.
- Implementar portal WEB, base de datos y demonio para servidor central sobre Windows.

1.3. Resumen del proyecto

Durante el transcurso del proyecto los objetivos se fueron cumpliendo según lo planificado. A medida que el mismo avanzaba, las metas se cumplían y las ideas empezaban a tomar forma, surgieron nuevas iniciativas e ideas que terminaron de modelar los requerimientos planteados al inicio.

Al comienzo se analizaron las diferentes implementaciones de red disponibles para la red 802.15.4. Finalmente optamos por implementar nuestro sistema de acuerdo con la solución propuesta por el grupo RSIS, que trabajó en su proyecto en el mismo período que nosotros. Entre ambos grupos definimos un protocolo y formato de mensajes a intercambiar de modo de tener una solución compatible y que permita implementar el sistema de punta a punta sin inconvenientes. Cualquier otra implementación de red que respete el formato de mensajes definido será cien por ciento compatible con nuestro sistema.

En esta primera etapa, los nodos puentes fueron ocupados por routers Linksys WRT350N debido a su capacidad y a tener un puerto USB disponible donde se conecta el mote central. Se estudió a fondo el equipo, las necesidades y finalmente se decidió instalar un sistema operativo (firmware) específico y adaptado basado en DD-WRT, más adelante en la documentación se explica el por qué de la decisión. Sobre este firmware implementado, se desarrolló el driver requerido para reconocer al mote central conectado vía USB.

Se analizaron las tres opciones de envío y finalmente se optó por la implementación de un túnel, que mediante el módulo “serialforwarder” envía los paquetes hacia el servidor central donde finalmente serán procesados. Los motivos de esta decisión se verán también más adelante.

En este contexto, la inteligencia respecto a cómo intercambiar datos con la red de sensores debió implementarse en el servidor central. Para ello se desarrolló un software (demonio) que hace de interfaz entre dicha red y la base de datos instalada.

En cuanto al desarrollo de la aplicación para el usuario se tuvieron largas discusiones sobre cómo implementarla. Por momentos se habló de dos aplicaciones, una desarrollada en Java que se utilizaría directamente sobre el servidor central, y otra web (con menos funcionalidades) que permitiría al usuario conectarse remotamente a través de un navegador de Internet. Finalmente se optó por implementar una única aplicación implementada con PHP, CSS y Javascript que puede ser accedida tanto desde el servidor central como desde un sitio remoto. Las funcionalidades son las mismas independientemente del origen del acceso, sin embargo, se establecieron tres tipos de usuarios: básico, personalizado y administrador donde los dos primeros tienen capacidades limitadas configurables mientras que el administrador tiene acceso total. Cada usuario ingresado por el administrador puede ser clasificado dentro de una de estas tres opciones. La aplicación tiene diversos casos de uso que se verán más adelante. Al momento del diseño se tuvo especial cuidado en que ésta sea ágil e intuitiva para el usuario. Por esto mismo se eligió un diseño similar al de las páginas webs más populares de hoy en día, de forma que el usuario se sienta familiarizado con el entorno.

1.4. Estructura de la Documentación

A continuación se presenta como está organizado este documento.

En el segundo capítulo se verán aspectos relativos a la red de datos del sistema. Primero se explica cómo se implementó el envío de datos y el porqué de la elección frente a otras opciones. Se describirá la configuración de red considerada, su performance y se analizarán los motes utilizados, sus características y requerimientos para lograr un intercambio de mensajes exitoso.

El tercer capítulo trata sobre los dos dispositivos principales que integran el sistema: router y servidor. Respecto al router utilizado, se analizará su configuración, sistema operativo instalado, driver y script de inicio. Se verá en detalle porqué finalmente se decidió implementar un firmware con DD-WRT en vez de Open-WRT. En lo referente al servidor, se explicará cómo está diseñada la base de datos y su estructura. Luego se verá cómo trabaja el demonio y cómo opera de interfaz entre los motes y la base de datos.

La aplicación para el usuario, el procesamiento de los datos y base de datos implementada son desarrollados en el cuarto capítulo. Primero se verá el diseño del software explicando los casos de uso disponibles. Específicamente se explicarán los principales desarrollos en PHP, Javascript e integración con otros sistemas como *Google Maps*. Este capítulo se cierra con cómo se creó el instalador del software.

En el último capítulo se extraen las conclusiones finales del trabajo y se analiza el trabajo a futuro.

En el apéndice se incluye un análisis de SNMP y XML, resumen de pruebas realizadas para comprobar el funcionamiento del sistema, un interesante análisis de inversión y el manual de usuario que recomendamos leer a quien decida instalar y utilizar este sistema.

Capítulo 2

Red

2.1. Decisión del Envío

El área a monitorear se separa en distintos pétalos o clusters, cada uno intercomunicado con el servidor central por medio de un router o gateway. Éste tiene conectado en el puerto USB un mote central que es el encargado de la recolección de los datos de todos los sensores de su cluster. El router deberá entonces recibir estos datos y asegurar el envío de los mismos al servidor central donde serán procesados y almacenados.

Una de las decisiones estratégicas más importantes fue el lugar donde procesar los datos de los sensores, como procesarlos y como enviarlos al servidor.

Analizando los existentes sistemas de gestión y monitoreo se destacaron SNMP y XML como las opciones de envío más utilizadas. En particular, durante el estudio de XML se tomó conocimiento del desarrollo de un sistema llamado SensorML donde, basado en la codificación XML, se implementan modelos estándar para el monitoreo y recolección de datos. Estos tres sistemas tienen la semejanza de que los datos son procesados en el router y enviados al servidor por medio de un protocolo estándar. En el apéndice A se describirán más en detalle las opciones recién mencionadas.

El primer paso en este análisis fue determinar las características del problema. El sistema a desarrollar deberá ser simple tanto en la utilización cotidiana como al momento de agregar nuevos routers y sensores. Por otro lado, el router deberá ser transparente a la solución, pudiéndose utilizar distintos routers y siendo simple la configuración de los mismos. Con esto se asegura la continuidad de este proyecto ya que no será necesario contar con un router especial ni realizar una gran inversión para continuar el desarrollo del sistema.

El objetivo principal del proyecto es lograr una solución al problema propuesto mediante un sistema escalable, al cual se le puedan agregar funciones, formato de mensajes e inteligencia. Se considera que este proyecto es el comienzo de una rama de desarrollo y del cual partirán distintas iniciativas y aplicaciones tanto en el agro como en otros ambientes.

En base a esto se puede concluir que el router no debe jugar un papel importante. Los mismos constan de baja memoria, procesadores lentos y sistemas operativos reducidos. Esto provoca que realizar modificaciones y desarrollos de aplicaciones generen mucho trabajo, tiempo de implementación y en otros casos ni siquiera sea posible llevarlos a cabo.

Es por esto que el estudio se centró en otra forma de envío de las mencionadas anteriormente, en la tunelización de paquetes.

2.1.1. Tunelización

La tunelización es un método por el cual se utiliza una red intermedia para la transmisión de datos entre dos extremos. Los paquetes a enviar se encapsulan sobre otro encabezado conteniendo la información necesaria para su encaminamiento dentro de la red de transporte. Una vez llegado a destino, se desencapsula y se envía el paquete original al destino final. De esta manera, la red intermedia es transparente para los extremos de la conexión, en este caso, mote central y servidor. El túnel se hará sobre TCP por ser el protocolo líder en transmisión de datos fiable. En la sección 2.2 se detallarán las características de este protocolo y cómo impacta en la red implementada.

Se muestra en la figura 2.1 un esquema del proceso de encapsulamiento Serial sobre TCP.

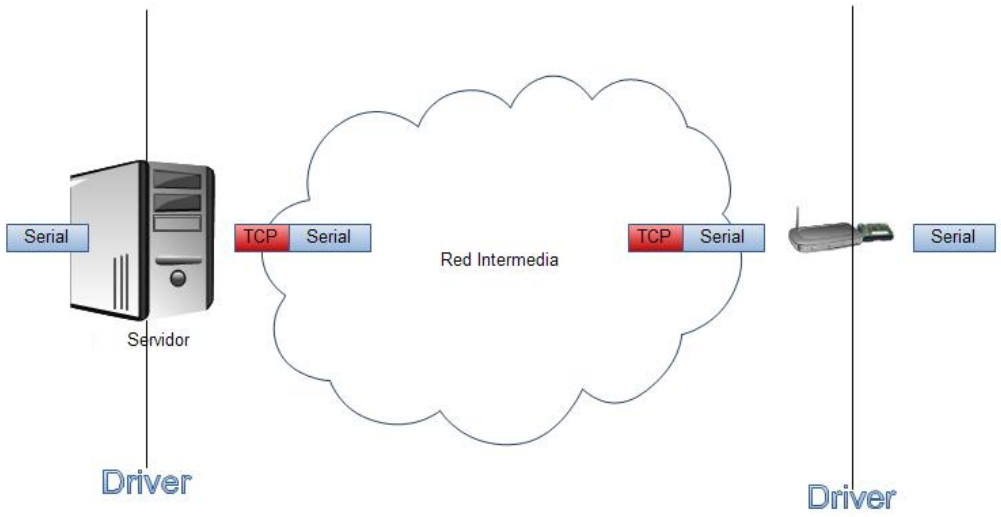

Figura 2.1: Esquema del proceso Serial sobre TCP

Como se observa en el diagrama, el router puente de cada cluster encapsula los datos provenientes del mote central y los envía al servidor. En éste correrá un programa que recibe estos paquetes y entrega el contenido serial al encargado del procesamiento y almacenamiento de las medidas de los sensores. En ambos sentidos de la comunicación, tanto la red intermedia como el router son transparentes para este proceso.

Se logra entonces llevar toda la lógica del procesado y decodificación de los mensajes al servidor donde fácilmente se puede, entre otras cosas, incluir nuevas funciones y agregar o modificar mensajes y protocolos. La única restricción que se impone es la utilización de paquetes con el formato TinyOS 2.x.

Más aún, podrán coexistir en una misma red de monitoreo distintos routers, con distintos sistemas operativos e implementaciones de tunelización sin la necesidad de realizar algún ajuste en el servidor.

El router pasa a ser un simple y tonto dispositivo de red el cual puede ser reemplazado en cualquier momento y no genera ninguna restricción al momento de agregar funcionalidades a nuestro Sistema de Monitoreo de Sensores Inalámbricos.

Por otro lado, estudiando los desarrollos realizados por la comunidad de TinyOS se observa que existen varias aplicaciones para redes de sensores donde el mote central está conectado directamente al servidor. Con la arquitectura aquí propuesta estas aplicaciones podrán ser utilizadas con sensores distantes del servidor. Se deja entonces una posible herramienta para la realización de nuevos sistemas de monitoreo a distancia independientes

de SiMSI pero que podrán utilizar el Router SiMSI para la transmisión de datos.

2.2. Configuración de Red

El proyecto resuelve el monitoreo en situaciones donde la distancia entre el casco de la estancia y los sensores es demasiado grande como para poder conectar los motes directamente al servidor. Una opción sería la utilización de motes con mayor potencia de transmisión pero la performance de la batería se vería muy afectada por este cambio. Otra opción sería utilizar los puntos con electricidad de la plantación para instalar un dispositivo que recolecte los datos de los sensores cercanos y los envíe con una mayor potencia de transmisión hasta el servidor central.

Un paso más lejos, y al que apunta este proyecto, es generar una red mesh entre routers conectados a corriente para poder esparcir aún más los sensores dentro de la plantación y obtener redundancia al momento de algún fallo en el router.

Para esto se diseñó un router simple y transparente de manera que el servidor no percate un problema en este dispositivo, y simple para permitir futuros desarrollos sin la necesidad de engorrosos trabajos dentro del router puente utilizado.

Una clave de este diseño fue la utilización de tunelización (serial over TCP) para el envío de datos desde el router hasta el servidor, donde se quita peso al router gateway, y la red intermedia se vuelve transparente para la comunicación. Dada la gran dependencia de capas que existe en el modelo OSI, y en particular en la implementación de TCP, el túnel realizado sobre la mesh de capa 2 es, en principio, independiente del estado de los enlaces de la red. Esto quiere decir que el túnel podría permanecer abierto cuando en realidad no existe un camino entre las puntas del mismo. En el momento en que se desee enviar un paquete y no haya conexión en capa 2 entre la puntas, el mismo no recibirá respuesta (ACK) y luego de un determinado tiempo se cerrará el túnel. Esto quiere decir que no será necesario realizar un chequeo intensivo del estado de los túneles, ya que mientras que no se intente enviar un paquete a través de él, el mismo seguirá funcionando. En base a esto, se configura el período con el que se realiza el chequeo de estado de túneles igual al período mínimo de envío de datos. De esta manera se perderá como máximo una ráfaga de paquetes.

Como se explicó anteriormente, los routers se instalarán a la intemperie conectados a 220V en medio de la plantación a monitorear. Dependiendo la zona en la que se encuentre la plantación, es posible que ésta no cuente con una corriente eléctrica estable pudiendo tener microcortes, picos de baja y alta tensión, etc. Por lo tanto es necesario que el equipo a utilizar sea muy robusto. Por otro lado, dado que no habrá una persona de soporte en el campo, y seguramente tampoco cerca de él, es necesario contar con routers diseñados especialmente para condiciones exteriores donde ni la temperatura ni la humedad causen fallas en el mismo. Como se verá a continuación en la documentación, todo el desarrollo que se hará sobre el router será considerando que el mismo puede cambiar en la implementación real y debe ser fácil la adaptación del mismo. Luego de un estudio de los posibles routers a utilizar se recomienda hacer pruebas con el MikroTik RB230 ya que es una marca relativamente barata y confiable y en este modelo se puede cargar un linux reducido donde correr el driver SiMSI. Algunas de sus características son disponibilidad de un puerto USB, posibilidad de ser instalado outdoor, posibilidad de instalar distintos tipos de antena para lograr un mejor alcance, etc. El precio de mercado está cerca de los 350 dólares.

Por otro lado, una vez almacenados y procesados los datos en el servidor, se desplegarán mediante una página Web diseñada especialmente para su fácil uso y soportada por varios dispositivos y navegadores.

Se muestra un diagrama con la solución detallada en la figura 2.2.

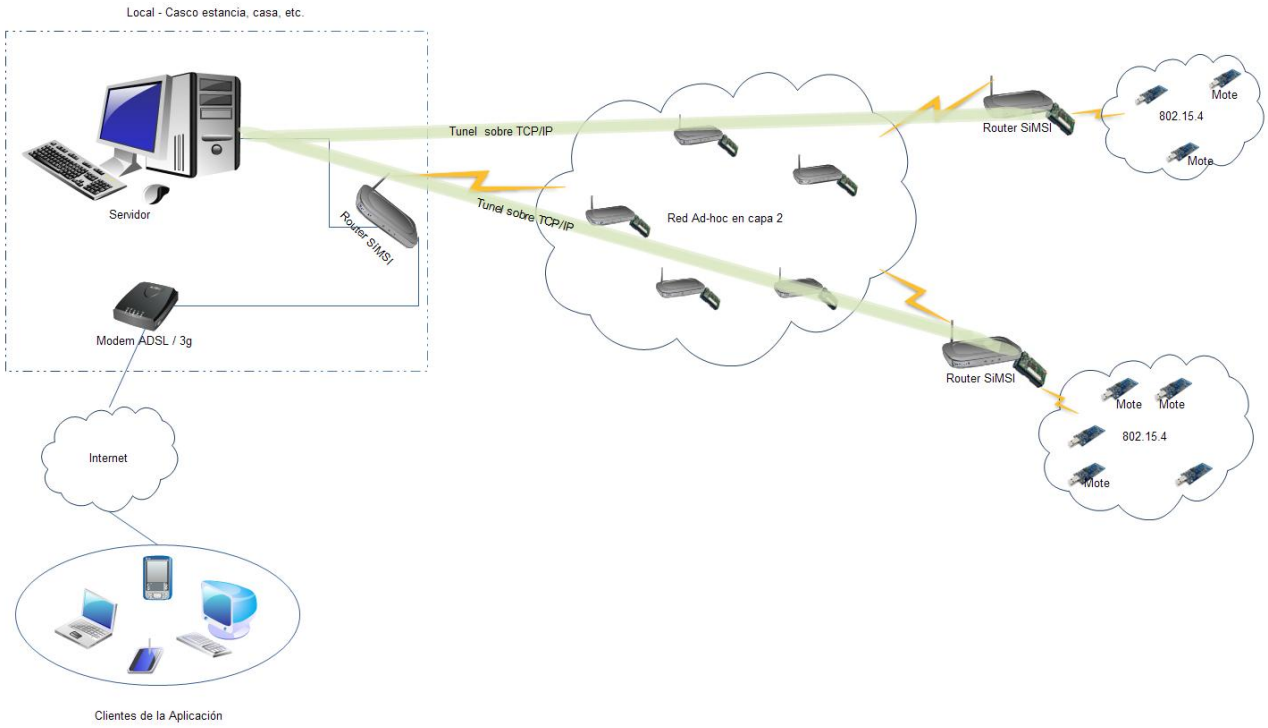

Figura 2.2: Diagrama de la Red SiMSI

2.3. Performance de la Red 802.11

Las características de la red planteada se alejan de lo que es una típica red de datos donde existe un tráfico casi continuo durante gran parte del día y donde se necesita una alta disponibilidad.

En este caso, se enviarán pequeños mensajes periódicamente, cada tiempos mayores a 5 minutos, por lo tanto, para que existan pérdidas de paquetes el enlace tiene que estar caído justo en el momento cuando el router envía la información.

¿Cuáles pueden ser las causas de estas pérdidas?

1. Problemas de Radio Frecuencia
2. Problema en el Router
3. Problemas en el Servidor
4. Problemas eléctricos

Los últimos 3 puntos no tienen rápida solución ya que no hay una persona en el campo capacitada para solucionarlos en el momento por lo que debemos intentar evitarlos pero una vez que ocurren no hay manera de salvar los datos recolectados.

Por el contrario, se podría pensar en la implementación de un buffer en el router que espere el ACK de TCP antes de borrar el dato enviado de la memoria, para lograr así evitar la pérdida de paquetes debido a posibles problemas de radio (fading).

En sistemas en 2,4Ghz ni la atenuación atmosférica ni la lluvia son causantes de fading, pero sí lo son shadowing y multipath.

Los problemas de shadowing, u obstrucciones, se van a despreciar ya que en base a educación se pueden solucionar, explicando que no se deben estacionar o dejar grandes objetos al lado de los routers.

Se concentrará el estudio en evaluar la probabilidad de que se quiera enviar un paquete en el momento en el que el enlace no está disponible.

El modelo más utilizado para el cálculo de esta probabilidad es el de Barnett-Vigants donde especificando datos del terreno como lugar geográfico, rugosidad y distancia, frecuencia y disponibilidad del enlace estima el Margen de Fading necesario para lograr el nivel de servicio deseado.

Se supone una plantación como la mostrada en la figura 2.3 donde el campo tiene un largo de 600m.

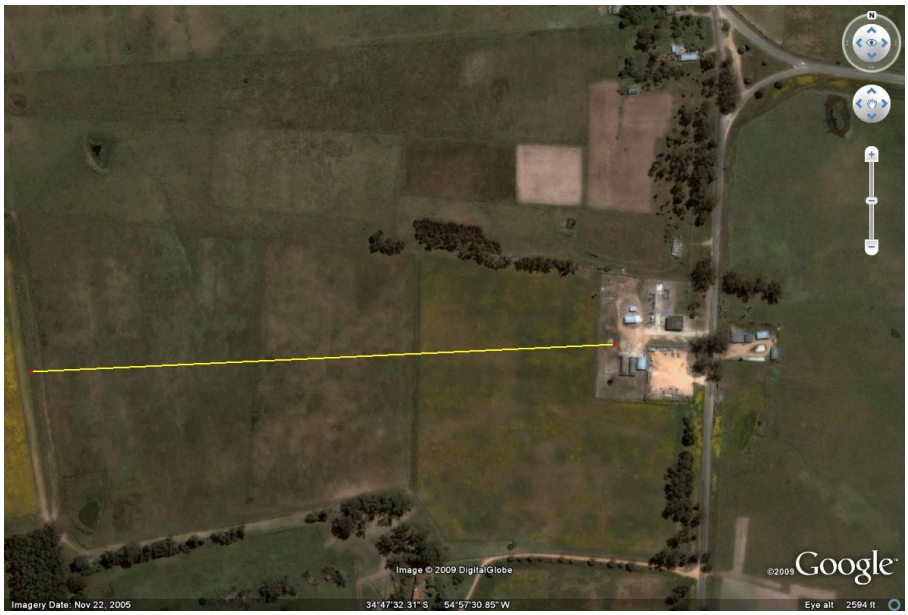

Figura 2.3: Ejemplo de plantación

Considerando datos estándar de equipos Wifi exteriores, se tiene una potencia de transmisión de 18dBm, sensibilidad para 1-2Mbps de -90dB, una antena transmisora omni-direccional con ganancia de 6dBi y el router en el extremo del campo con una omni de 3dBi.

Se supone también una altura de 1,5m para el receptor y 6m para el transmisor (ya que se encuentra en el casco de la estancia). Como se dijo anteriormente, en este ejemplo la distancia entre ambos será de 600m.

Para lograr una buena comunicación entre dos nodos es necesario tener despejado al menos el 60 % del primer elipsoide de Fresnel. En esta franja se transmite cerca del 95 % de la potencia de la señal, por lo que obstrucciones dentro de esta zona pueden provocar grandes pérdidas de potencia.

En la figura 2.4 se muestra el perfil del enlace (suponiendo terreno plato) con ambas antenas en los extremos donde se puede ver que, por más que el primer radio de Fresnel se ve obstruido, el 60 % del mismo queda despejado por lo que se pueden estimar las pérdidas de propagación como pérdidas en espacio libre.

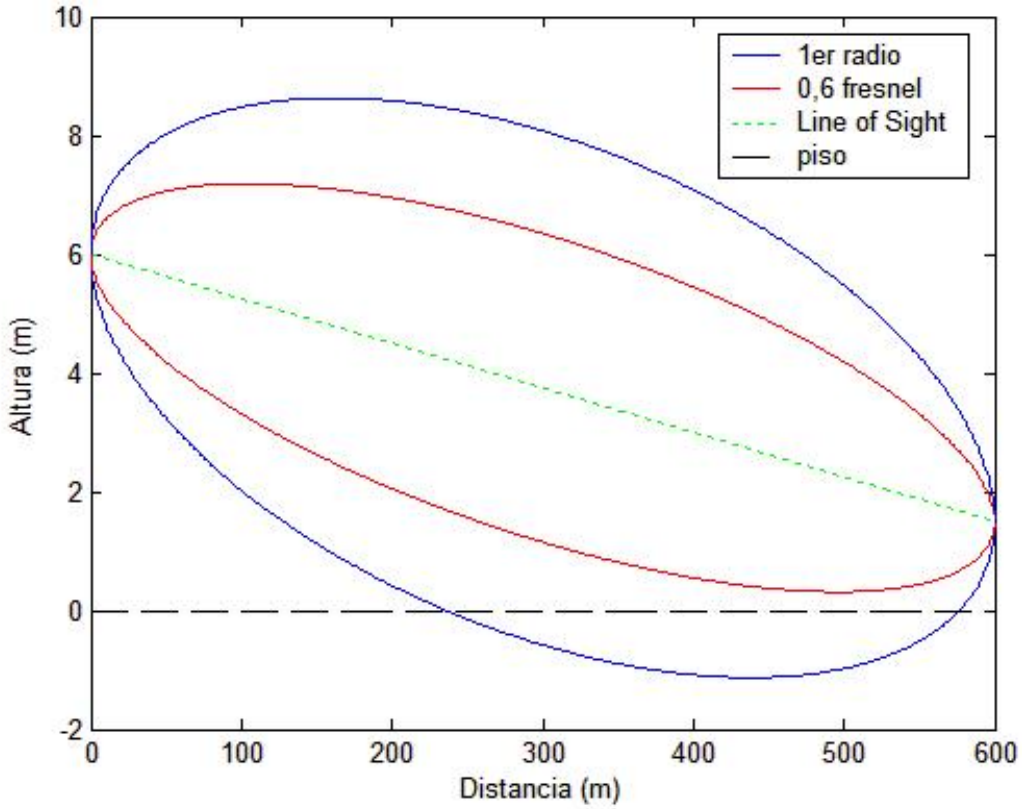

Figura 2.4: Estudio de Radios de Fresnel

Una vez verificado el despeje necesario, se estudia el balance de potencias del enlace. Utilizando Friis como base se utiliza el cuadro Excel mostrado en la figura 2.5 para calcular el margen del sistema.

Balance De Potencias		
Distancia	0,6	km
Frecuencia	2,4	GHz
Pérdida por espacio Libre (dB)	95,61	dB
Pérdidas en Tx (coax., conector...)	1	dB
Pérdidas en Rx (coax., conector...)	1	dB
Potencia transmisor (Ptx)	18	dBm
Ganancia transmisor (Gtx)	6	dB
Ganancia receptor (Grx)	3	dB
Sensibilidad receptor (Srx)	-90	dB
Nivel de la señal en el receptor	-70,61	dB
Margen del sistema	19,39	dB

Figura 2.5: Balance de Potencias

Como se observa, se obtiene un margen de 19dB, por lo que se puede asegurar una disponibilidad mayor a 99,998 %. El enlace estará caído menos de 2 segundos diarios por lo que se puede despreciar la probabilidad de pérdida de paquetes en la red.

Se considera entonces que no es necesaria la implementación del buffer en el router, logrando así realizar la

menor cantidad de tareas en el mismo.

Al momento de diseñar una solución particular, la colocación de los routers en el campo deberá prever posibles obstáculos como árboles, galpones etc. y utilizar las propiedades de la red mesh entre APs para sortearlos.

Por otro lado, no se recomienda la instalación de routers consecuentes a más de 200m de distancia dado que se verá obstruido considerablemente el 60 % del primer fresnel.

En cuanto a la performance de la red, un mensaje promedio enviado por el router es de 10 bytes, que suponiendo una velocidad de 1Mbps (peor caso) el tiempo total de transmisión (DIFS, SIFS, Data, ACK, etc) será alrededor de 600 microsegundos.

Si se supone una red con 30 motes por router, este transmitirá periódicamente 18 milisegundos por lo que la carga impuesta por los mensajes no es significativa como para dedicar un estudio específico a este tema.

Se considera entonces que la red podrá soportar grandes plantaciones con decenas de routers y sensores.

Dado que los routers estarán dispersos en el campo y seguramente varios no estén dentro de la zona de cobertura de otros routers el problema de la estación oculta podría llegar a ser considerable en una red muy grande y con muchos sensores.

De forma explicativa, el problema recién mencionado es cuando un nodo de la red comienza una transmisión que ocasionará una colisión con una transmisión en curso debido a que el sensado del medio resultó libre.

Como se dijo, para las redes que se consideran en este proyecto esto no es un problema pero se plantea para futuros desarrollos. Este problema se puede solucionar “fácilmente” implementando el mecanismo de RTS/CTS que, aunque cada transmisión lleva más tiempo, evita las colisiones ya que todas las estaciones permanecen en silencio, vean el medio ocupado o no, mientras que otra habla.

La implementación de este mecanismo en redes mesh tiene varias consideraciones pero no se entrará en estos detalles ya que no son necesarios para el problema planteado aquí.

2.4. Motes y Sensores

En las secciones anteriores se estudió la transmisión de datos desde el servidor hasta los routers gateways de los clusters y viceversa. Quedó pendiente el análisis del funcionamiento interno de cada cluster.

Cada cluster está formado por un mote central conectado vía USB al Gateway y varios motes distribuidos por toda el área de cobertura de la hoja. Estos últimos contarán con uno o varios sensores que tomarán medidas periódicamente.

La comunicación entre estos motes está basada en el protocolo IEEE 802.15.4 LR-WPAN (Low Rate, Wireless Personal Area Network).

En esta sección se estudiarán las características de los motes utilizados y el protocolo de red implementado para esta aplicación.

2.4.1. Características de los Motes

Existe una tendencia creciente en el desarrollo de distintas aplicaciones para el monitoreo y optimización de recursos en el agro, permitiendo de esta manera la disminución de los efectos climáticos en las inversiones realizadas. Esto conduce al desarrollo de nuevos dispositivos diseñados especialmente para su funcionamiento por un largo período de tiempo en condiciones adversas. Es decir, instalados a la intemperie, alimentados por una batería y con poco o nada de mantenimiento. En especial, en este proyecto se trabajará con los motes de las empresas Moteiv y Crossbow.

Como se dijo anteriormente, un mote es un módulo inalámbrico de baja potencia utilizado en redes de sensores y en aplicaciones de monitoreo.

Tanto los motes de las empresas Moteiv como de Crossbow son compatibles con el protocolo 802.15.4, trabajan en la banda de 2,4GHz, soportan TinyOS, cuentan con un puerto USB en el que brindan una comunicación serial por medio del chip FTDI, son alimentados por dos pilas AA, tienen bajo consumo de batería y tienen un rango de funcionamiento de -40 a 85 grados Celsius aproximadamente.

Se muestra en la figura 2.6 una imagen explicativa del mote de Moteiv

Desde el principio del proyecto se trabajó con los motes de la empresa Moteiv cuyo sistema operativo es TinyOS. TinyOS es un sistema operativo Open Source desarrollado por la Universidad de Berkeley especialmente para redes de sensores inalámbricos. El mismo está diseñado para incorporar nuevas innovaciones fácilmente y para funcionar bajo las fuertes restricciones de memoria que se dan en las redes de sensores [1].

En Octubre de 2007 Sentilla, una compañía dedicada al desarrollo de aplicaciones para el ahorro de energía, compró el sector de Moteiv destinado al desarrollo de motes. Sentilla realiza una migración hacia plataformas Java, y los motes anteriores (que soportan TinyOS) se comercializaron hasta enero de 2008 [2].

Dado que en la Facultad de Ingeniería de la Universidad de la República hay un desarrollo importante basado en TinyOS, se prosiguió el trabajo con los motes de Crossbow que tienen características muy similares a los anteriores motes de Moteiv.

En lo referido a SiMSI, todo el análisis y despliegue de la información se podría re-utilizar en una red de sensores basados en esta nueva implementación de motes. En primer lugar, se debería implementar un nuevo driver en el router. Luego en el demonio se deberán modificar las clases que se encargan de abrir y cerrar la comunicación bidireccional entre el servidor y los routers y sustituir las clases que definen el objeto mensaje que es reconocido y utilizado para mandar información y datos tanto, por el demonio como por la capa de aplicación de los motes.

2.4.2. Detalles de la red 802.15.4

El estándar 802.15.4 define el nivel físico y el control de acceso al medio en redes inalámbricas locales con bajas tasas de transmisión. Existe una especificación de capas superiores llamada ZigBee utilizada en su gran mayoría en aplicaciones de domótica.

Dado que el estándar 802.15.4 no especifica la capa de red, no resuelve el ruteo entre nodos, pero deja la base para la implementación de redes ad-hoc.

Uno de los objetivos del proyecto fue trabajar en conjunto con uno de los grupos que están desarrollando como proyecto de fin de carrera las capas superiores a 802.15.4 optimizadas para redes de sensores.

En un principio se planteó la opción de tomar como punto de partida el proyecto de fin de carrera SIAGRO2 desarrollado en el año 2007. Dado que en este año RSIS, otro grupo de proyecto de fin de carrera de la facultad,

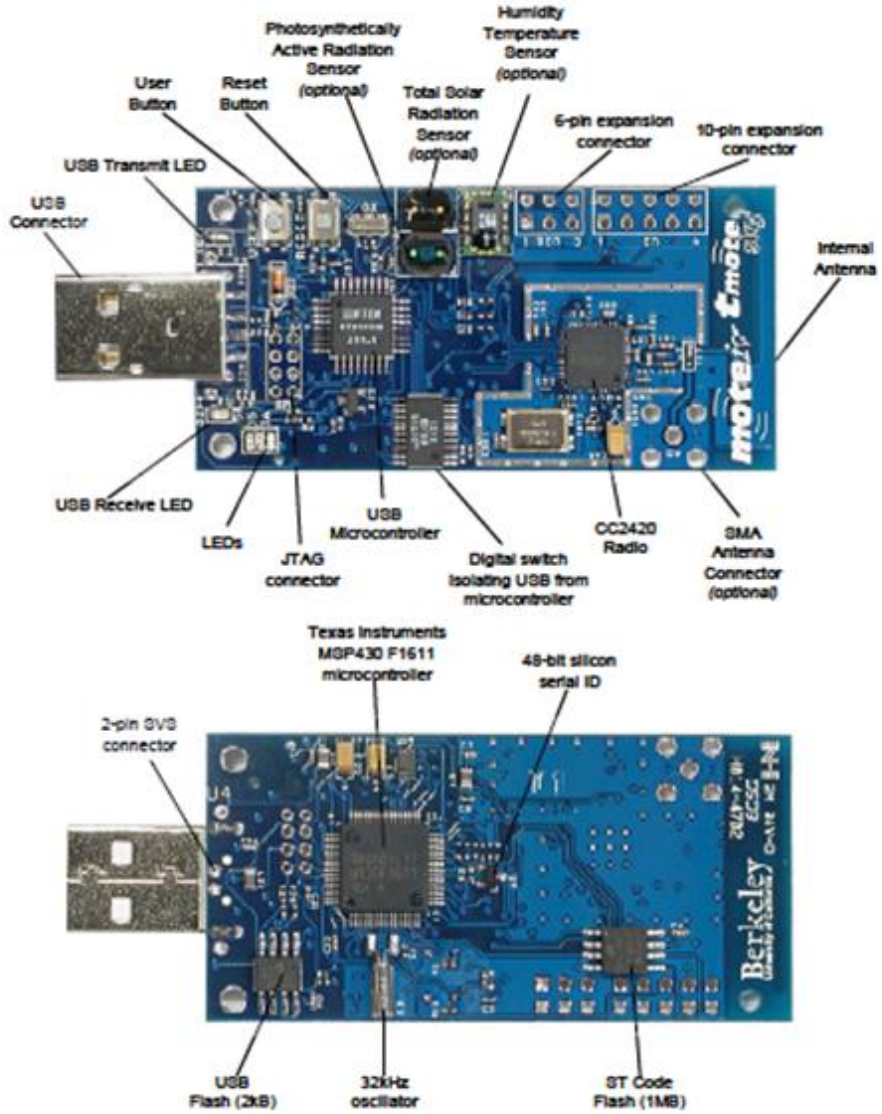

Figura 2.6: Imagen del mote de la empresa Moteiv

comenzó a trabajar en la optimización de este protocolo, se decidió trabajar con ellos. Para esto se fueron definiendo en conjunto los mensajes para la comunicación entre los motes y el servidor central. Se explicarán estos mensajes en detalle en la sección 3.2.5.

Como se dijo anteriormente, cada hoja tendrá un mote central conectado vía USB a un router. Todos los motes del cluster forman una red ad-hoc entre ellos para enviar los datos recolectados al mote central, que se encargará de transmitirlos por el puerto USB al Gateway.

Para aumentar el rendimiento de la batería, los motes "se duermen y se levantan" cada un período de tiempo determinado. Esto hace que la transmisión de un mensaje desde un mote al mote central pueda llevar varios minutos dado que unos de los saltos intermedios podría estar "durmiendo". En la estructura de la red multi-hop que se desarrolla, se establece como "mote padre" al primer salto de cada mote. El mote padre además de encaminar los datos, realiza funciones de control y configuración de sus motes hijos.

La frecuencia con que los motes leen los sensores conectados es variable y se puede configurar desde la

aplicación Web.

Un diagrama explicativo de la red formada por los motes se muestra en la figura 2.7.

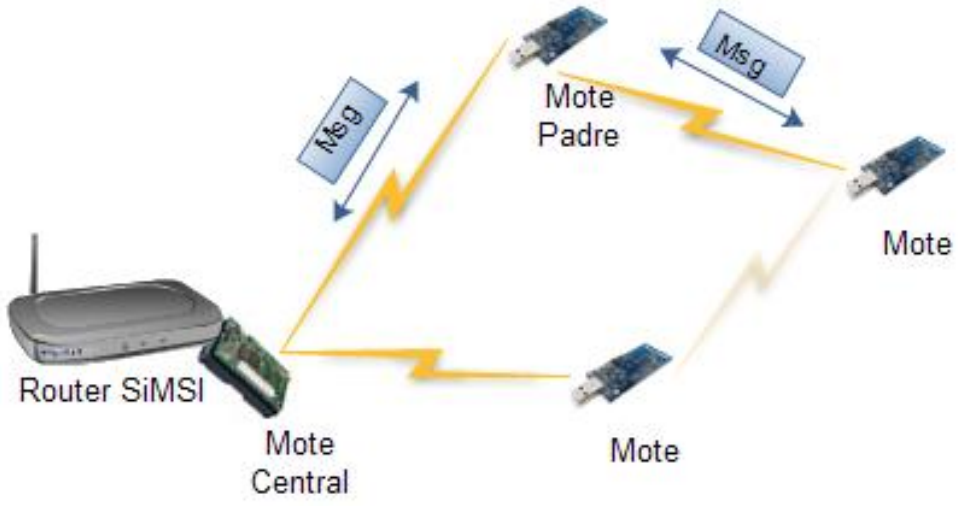

Figura 2.7: Red 802.15.4

Como se explica en tantas otras secciones, es muy importante que la solución completa de monitoreo sea muy simple al momento de instalar, configurar y ampliar.

En cuanto a la red ad-hoc formada por los motes, esta es auto-configurable. Basta con encender un mote dentro de la zona de cobertura de la hoja y éste ya forma parte de la red enviando y retransmitiendo datos al mote central.

Si bien los detalles de la red interna de los clusters no forman parte estrictamente del proyecto SiMSI, es importante tener en claro los conceptos anteriormente explicados para lograr un conocimiento global del sistema. De alguna manera, estos conceptos son supuestos que se tomaron antes de su desarrollo, en caso de aumentar sus funcionalidades o incorporar nuevos motes y dispositivos, se deberá tener en cuenta si se cumplen dichos supuestos. En caso contrario, se deberá modificar el sistema de forma tal que permita tener total compatibilidad.

Capítulo 3

Dispositivos

3.1. Router

En el problema propuesto por el cliente Roque Gagliano, se especificaba el Linksys WRT350N como router puente a utilizar.

Dado que este proyecto está en una primera etapa, donde se está estudiando su alcance, viabilidad y posible penetración en el mercado se buscó un router con al menos un puerto USB donde poder conectar el mote central, barato, popular y con constante desarrollo para facilitar el proyecto.

El Linksys WRT350N consta con una interfaz de red wireless 802.11b/g/n, 4 puertos Ethernet y un puerto USB 2.0 donde se conectará el mote central.

Como se dijo anteriormente, los motes cuentan con un chip FTDI que a través del puerto USB simulan un puerto serial. Nuestro driver deberá entonces recibir los datos provenientes de una UART y dejarlos disponibles para el envío hacia el servidor central.

A modo de referencia, una UART (Transmisor-Receptor Asíncrono Universal) es un controlador de puertos y dispositivos serie. Es el encargado realizar la conversión paralelo-serie para los datos salientes y entrantes del puerto serial a una velocidad de bit determinada.

Se muestra un diagrama explicativo de la solución a implementar en el router en la figura 3.1.

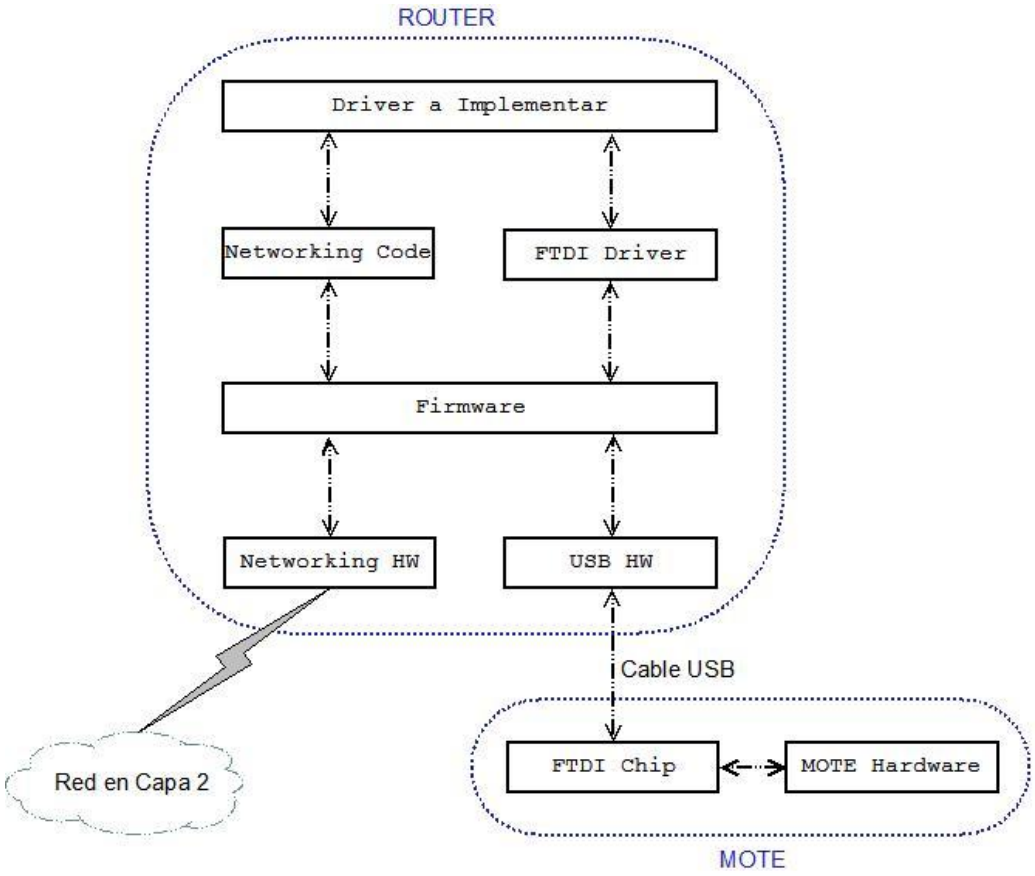

Figura 3.1: Diagrama de la solución para el Router

3.1.1. Firmware

El primer paso a realizar fue la elección del firmware. Se buscó un firmware Open Source con una comunidad activa donde poder encontrar soporte y actualizaciones del firmware instalado.

Dentro de los más populares se encuentran DD-WRT y Open-WRT, en los cuales se centró el estudio. Ambos basan su sistema operativo en una versión de Linux reducida y brindan herramientas para poder instalar y quitar paquetes a esta distribución lo que facilita el desarrollo de nuevas aplicaciones. Al momento de realizar el proyecto, Open-WRT no contaba con una imagen estable para el Linksys WRT350N y eran muchos los casos donde se perdía comunicación con el router al instalar la imagen del firmware.

Por otro lado, en DD-WRT se veía un gran desarrollo de aplicaciones así como personas trabajando en el Linksys WRT350N. En adición a esto, ya existían varias imágenes soportadas por este router y trabajos sobre la habilitación del puerto USB.

Estas razones hicieron que se continuara el trabajo en base al firmware DD-WRT. Luego de unas semanas de estudio se logró contactar con “Tornado” quien domina ampliamente la modificación de firmwares y quien brindó una gran ayuda en esta etapa. El firmware base utilizado es el v24 - build 0616 que viene con el Kernel 2.4.35. Se lo llama “base” ya que es de donde se parte para realizar las modificaciones pertinentes.

3.1.2. Driver

El primer desafío se centró en poder levantar un dispositivo USB en el router. Para ello, fue necesario instalar los módulos básicos del USB ya que no eran soportados en la imagen utilizada. Un módulo se refiere a un

controlador de dispositivos o servicios necesario para interactuar con ellos.

Una vez instalados los módulos, y pudiendo leer y escribir pen-drives se vio que todavía no era posible interactuar con el mote. La causante de esto fue la falta de controladores para el ya mencionado driver FTDI. Una vez encontrados los drivers para la versión de kernel utilizada, se debieron cargar los mismos respetando sus dependencias para un correcto funcionamiento.

Para que el proceso de instalación y carga de módulos sea lo más transparente posible a futuros desarrolladores, se realizó un programa, que se verá mas adelante en esta sección, que ejecuta de manera automática todo este proceso.

Con los módulos cargados, se logró reconocer el driver del mote y recibir y enviar datos a través de una UART.

El driver a implementar deberá crear una comunicación bidireccional entre el mote central y el servidor.

Como se dijo anteriormente, la implementación real de este proyecto podrá ser en otro equipo, distribución o sistema operativo. Es por esto que la implementación del driver deberá ser transportable y escalable para poder aceptar las posibles modificaciones en cuanto a los requerimientos del sistema como también a la masividad del mismo.

Se prosiguió entonces con el estudio del envío para determinar el proceso que deberá realizar el driver.

En la sección 2.1.1 se explica la razón por la cual se decidió realizar un túnel TCP.

Estudiando los avances realizados por la comunidad de TinyOS se vio que existen varias aplicaciones creadas para motes conectados directamente al PC y que basan su comunicación con los mismos en un programa llamado Serial Forwarder.

Para mantener compatibilidad con estas herramientas y aprovechar el continuo estudio y mejoras que realiza la comunidad en los programas desarrollados, se eligió utilizar el Serial Forwarder como driver, más precisamente la version 2.0

El mismo está diseñado para correr en LINUX pero puede ser ejecutado en Windows por medio de un emulador como Cygwin por ejemplo. El SerialForwarder controla el puerto USB y hace de Proxy entre el mote conectado directamente y las aplicaciones clientes. Para ello, envía todos los paquetes recibidos por el USB hacia un puerto TCP y viceversa, especificando previamente la velocidad de transmisión del puerto serial y el puerto TCP destino.

Utilizando como driver el Serial Forwarder, se tiene resuelto el control de la UART y la comunicación con el servidor del lado del router. Al ser un programa estándar desarrollado por TinyOS, a medida que se desarrollen nuevas versiones se podrá actualizar la imagen del router sin tener que realizar ninguna otra modificación en el sistema.

Una vez inicializado el Serial Forwarder, cualquier dispositivo dentro de la red podrá generar una comunicación TCP con el puerto especificado y recibir los paquetes en tiempo real. Se podrán conectar varios dispositivos al mismo tiempo en caso de ser necesario y éstos podrán conectarse y desconectarse en cualquier momento.

3.1.3. Instalación del Driver

Una vez tomada la decisión de utilizar como driver el Serial Forwarder, la instalación en el Linksys fue el siguiente desafío.

TinyOS proporciona un SDK con los códigos fuentes, Makefiles, etc. para poder compilar los programas en distintas arquitecturas de hardware. Dado que el router utilizado tiene una versión de Linux muy reducida no es posible compilar en él por lo que fue necesario utilizar un cross-compiler.

Este tipo de compilador es capaz de crear un archivo ejecutable para una plataforma distinta a la que fue utilizada para compilar. Es muy útil para realizar drivers en sistemas embebidos.

En este caso se utilizó el cross-compiler "mipsel-linux-uclibc-gcc" recomendado en el foro de DD-WRT para la compilación de programas para arquitecturas del tipo Mipsel (arquitectura de nuestro router Linksys).

El proceso de compilación se divide en 4 etapas: procesado, compilación, ensamblado y enlazado. En las primeras tres etapas se prepara el código escrito en alto nivel para ser procesado, se traduce a lenguaje ensamblador y se genera un archivo binario (código objeto) ejecutable por el procesador. En el ensamblado, se insertan al programa objeto los código máquina de las funciones externas utilizadas y se crea el archivo ejecutable.

Dado que el Serial Forwarder cuenta con varios archivos fuentes se debió realizar esta compilación en etapas, ensamblando primero todos los archivos y luego enlazando los archivos objeto obtenidos.

Una vez compilado el driver se debe correr indicando el puerto TCP a utilizar, el puerto donde está conectado el USB y la velocidad de transmisión serial. En este caso las tres opciones serán iguales entre los distintos routers de SiMSI por lo que se dejaron fijas en el driver.

3.1.4. Script de Inicio

En la inicialización del router se deberán insertar todos los módulos necesarios para poder interpretar al mote, ejecutar el Serial Forwarder y comenzar un loop infinito que realiza ciertas acciones en base a chequeos periódicos sobre el estado del túnel.

Estos scripts fueron colocados en /etc/config/ y son llamados IniciSF.startup y usb.startup.

Una de las grandes decisiones del proyecto, explicada en detalle en la sección 2.1.1, fue el utilizar un router lo más tonto posible. En base a esto, las únicas comprobaciones que éste hará será chequear si hay un mote conectado al USB y, en caso de que lo haya, verificar que el Serial Forwarder esté corriendo. Si no lo está, lo ejecutará.

El encargado de estas verificaciones es IniciSF.sh que es ejecutado al iniciarse el Linksys por IniciSF.startup. En caso de querer agregar otras opciones de monitoreo interno del router (como ser memoria y procesador utilizados, etc.) simplemente se deberán agregar al script recién mencionado.

Por otro lado, al insertar los módulos es muy importante respetar las dependencias que éstos tienen. Para ello se creó un script llamado InstMod que recorre los módulos contenidos en la carpeta /etc/modules.d/ y genera una lista con todos los módulos a insertar ordenados según su dependencia. Este archivo es recorrido por usb.startup al iniciarse el router insertando los módulos correspondientes.

De esta manera, se logra simplificar considerablemente el proceso de agregar o quitar módulos en la imagen.

Se utilizaron las herramientas ipkg_install.sh e ipkg_remove.sh para instalar y borrar paquetes IPKG.

La instalación de estos archivos en el router se realizó mediante el Firmware Modification Kit de DD-WRT que permite modificar todos los archivos y carpetas del sistema operativo y generar un nuevo binario con las modificaciones realizadas.

3.2. Servidor

La arquitectura desarrollada en este proyecto se basa en un PC (servidor) en el casco de la estancia, que recolecta, almacena y despliega los datos provenientes de la red de sensores.

Para esto el servidor actúa como servidor Web para la aplicación a nivel de usuario y como servidor de base de datos interactuando con el demonio que almacena los datos recolectados por la red de monitoreo. A su vez, también responde a las consultas que genera el cliente remota o localmente a través de la aplicación web programada en php. Todas estas aplicaciones están desarrolladas bajo Windows como sistema operativo.

3.2.1. Estructura del Servidor

Como se dijo anteriormente, y se ve reflejado en el diagrama de la figura 3.2, se tienen tres entidades principales en el servidor: la base de datos, el demonio y la aplicación Web.

Los dos últimos se comunican entre ellos por medio de la base de datos. En el extremo de la aplicación Web, el usuario es capaz de visualizar y analizar los datos recolectados, posibles alarmas, etc. por medio de consultas a la base de datos. Por otro lado, el usuario puede realizar configuraciones de red, como por ejemplo cambio de la frecuencia de medida de ciertos sensores, a través de la aplicación que almacena estas configuraciones en la base de datos. El demonio, que recorre estas tablas periódicamente en busca de cambios, descubre que hay configuraciones a realizar a nivel de red y se encarga de traducirlas y enviarlas hacia los equipos en cuestión.

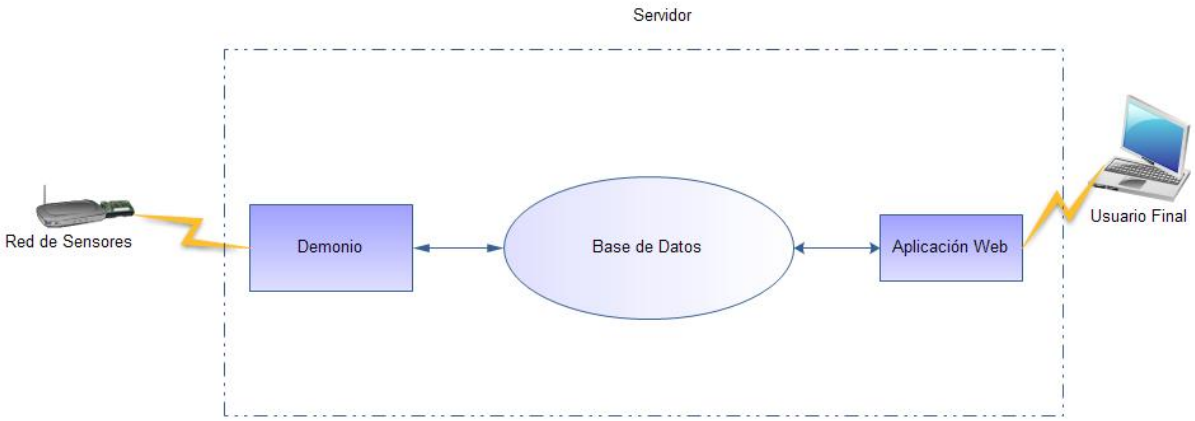

Figura 3.2: Estructura del Servidor

3.2.2. Implementaciones en el Servidor

En esta sección se explicará y argumentará la elección de las herramientas utilizadas para la realización de la aplicación y el almacenamiento de datos. Las tres principales son: Php (lenguaje de programación de la aplicación), MySQL (servidor de base de datos) y Apache (servidor Web).

La implementación de estos tres componentes se realiza básicamente instalando los paquetes correspondientes a un servidor WAMP. La sigla proviene de la conjunción de los principales actores: Windows, Apache, MySQL y PHP.

Elección de Windows

El punto de partida en la implementación del servidor fue la elección del sistema operativo. Desde un principio Roque Gagliano sugirió la utilización de Windows argumentando que se trata de un PC de uso doméstico y que las personas que lo utilizarán no están totalmente capacitadas para manejar otro sistema operativo.

Además existe una amplia aceptación de Windows mundialmente, con una gran diversidad de aplicaciones desarrolladas para la implementación de servidores.

De igual manera, Linux sería otra opción a elegir debido a su gran cantidad de usuarios e implementaciones. Pero este sistema operativo presenta la desventaja de que, generalmente, el usuario básico de PC no se encuentra familiarizado con él.

Lo ideal sería evaluar las necesidades y requerimientos del cliente, para luego utilizar Linux o Windows dependiendo de cuál sea la opción más conveniente.

En este caso compartimos la sugerencia de nuestro tutor, y creemos que lo mejor es desarrollar un sistema que pueda ser operado por cualquier usuario con conocimientos básicos de computación y pueda ser instalado en un PC con requerimientos muy básicos también.

Elección de PHP

Al momento de optar por tener una interfaz web a nivel usuario, se debe elegir el lenguaje para programar la misma. Se eligió PHP por los motivos explicados a continuación.

En primer lugar, PHP es independiente de la plataforma en la que se ejecute, es decir que no depende del sistema operativo en el que se implemente. En el caso de SiMSI, esto no es estrictamente necesario ya que se decidió implementar el servidor web sobre Windows, pero deja la posibilidad de migrarlo en un futuro a otro sistema operativo.

Otro punto clave, y de los más importantes, es que PHP es de desarrollo abierto (open source). En muchas de las decisiones del proyecto se consideró esta característica como fundamental, ya que al trabajar con un lenguaje o programa Open Source, no se depende de una cierta compañía para el soporte y debugging, ni tampoco se deben pagar actualizaciones anuales.

Además de ser Open Source, PHP tiene una gran comunidad activa de desarrolladores que permite encontrar una solución rápida a cualquier traba que se interponga a la hora de programar. Esta comunidad también actualiza y desarrolla nuevas librerías por lo que se puede encontrar una gran variedad de funciones a utilizar facilitando así el desarrollo de la aplicación.

Dado que el lenguaje de programación y el motor de la base de datos están muy ligados, es importante para futuras optimizaciones la posibilidad de poder cambiar de motor ya sea porque la elección no haya sido la adecuada o porque cambiaron los requerimientos del sistema. PHP tiene la particularidad de actuar con muchos motores de base de datos y en particular con MySQL, que es el utilizado en este proyecto.

Por último, considerando que la realización del proyecto de fin de carrera tiene muchos desafíos y puntos de investigación, se intentó buscar entre las opciones posibles, aquélla que genere menor incertidumbre al momento de la implementación. En el caso de PHP, varios integrantes ya estaban familiarizados con el lenguaje, lo que ayuda a acortar tiempos tanto en la etapa previa a la programación así como durante el desarrollo de la aplicación en sí. Se trata además de un lenguaje intuitivo que permite al programador adaptarse fácilmente.

Elección de MySQL

Otra de las decisiones importantes para el buen funcionamiento y escalabilidad del sistema es el servidor de base de datos. Dado que los datos del proyecto se van a alojar en tablas que pueden llegar a tamaños difíciles de manejar, el motor de la base de datos podría llegar a ser un punto crítico en la implementación. A modo de ejemplo, si se toman datos cada 20 minutos y se supone que cada dato es almacenado en una línea de una tabla, por mote se tienen 72 filas nuevas por día. Pensando en la escalabilidad del proyecto, suponiendo que se puede tener 100 motes en un campo, se agregan 7200 filas por día en una tabla.

Investigando los motores utilizados por grandes empresas como Google, Yahoo, Facebook, etc. se destacó MySQL como primer candidato. Este motor, por medio de un diseño óptimo de las tablas, logra manejar rápidamente tablas del orden de 50 millones de líneas, lo que asegura una buena escalabilidad en nuestro sistema.

De igual manera que en las decisiones anteriores, se hizo hincapié en la posibilidad de correr la base bajo cualquier tipo de plataforma, de que posea un manejo intuitivo, Open Source y con una comunidad activa por detrás. Características claves como ya se mencionó anteriormente.

Por otra parte, se puede considerar que otros motores tienen mayores prestaciones que MySQL, que versión tras versión van ampliando las herramientas brindadas, pero esto no es un punto crítico en el proyecto ya que se utilizará un manejo básico de la base de datos centrándose en la velocidad de las consultas.

Otra de las características a tener en cuenta en la implementación del servidor es que el mismo no será exclusivamente un servidor de base de datos, sino que también será un servidor Web y probablemente será utilizado como computador de escritorio por los habitantes del campo. Según lo estudiado, MySQL está diseñado para manejar los recursos de memoria del servidor de manera eficiente sin tener la necesidad de contar con grandes tamaños de disco duro y RAM.

Debido a las razones aquí mencionadas se tomó la decisión de que el servidor de base de datos de SiMSI esté basado en MySQL.

Elección de Apache

Debido a que la interfaz a nivel de usuario es una aplicación Web, es necesario que el PC ubicado en el campo oficie de servidor Web para que el usuario pueda acceder remotamente al sistema. Es necesario entonces decidir qué tipo de servidor Web se implementará.

A continuación se enuncian una serie de ventajas de Apache que llevaron a la elección del mismo.

En primer lugar, y de la misma forma que PHP y MySQL, Apache es de licencia libre y es soportado por cualquier tipo de plataforma, lo que permite que el proyecto SiMSI pueda ser utilizado en distintos sistemas operativos.

También estas tres aplicaciones son 100 % compatibles y funcionan perfectamente en conjunto. Como se dijo anteriormente, todas cuentan con comunidades de desarrolladores activas que permiten la rápida solución a cualquier problema suscitado y la continua mejora de las aplicaciones y librerías.

La alta velocidad de Apache es un punto clave, debido a que una de las prioridades de SiMSI es brindar una navegación simple, intuitiva y rápida al usuario. En especial se necesita un servidor rápido para acciones como graficar y visualizar el mapa, donde la navegación puede resultar un tanto tediosa para el usuario si no se logra una velocidad comparable al de las páginas Web que está acostumbrado.

Por último Apache es de los servidores Web más populares hoy en día por lo que fácilmente se podrán encontrar ejemplos y aplicaciones para futuras ampliaciones en el servidor.

3.2.3. Implementación de la Base de Datos

Al momento de diseñar la base de datos se focalizó en la performance de la misma. Para esto fue necesario, sobre todo en las tablas más pobladas y con una tasa de crecimiento importante, realizar una indexación adecuada, buscando a su vez la menor cantidad posible de campos null.

El proceso de indexación es muy relevante a la hora de hacer una consulta debido a que lo que se busca es evitar recorrer toda la tabla en busca de coincidencias. Usando índices, se logra particionar la tabla para que el motor no tenga que recorrerla en su totalidad en cada consulta. Con esto se obtiene una gran mejora en la velocidad de las consultas, y cuanto más poblada esté la tabla más se aprecia la diferencia. Los distintos índices utilizados en cada tabla serán explicados en su momento cuando se enuncien las tablas implementadas en SiMSI.

Otro parámetro importante a tener en cuenta es la cantidad de campos en las tablas con valores nulos. Cuando un campo no es nulo, el motor procesa la petición de una manera más eficiente, mejorando de esta manera la performance del servidor de base de datos. Por otra parte, tener campos con valor null en exceso genera retrasos en las consultas y ocupa espacio preciado del disco duro que es inutilizado.

Por lo tanto se trabajó con la necesidad de utilizar valores que en la medida de lo posible no sean nulos, buscando que las columnas de las tablas sean lo más precisas posibles entregando la información justa y necesaria para la aplicación.

Para que la base de datos tenga consistencia y coherencia fue necesaria la implementación de claves foráneas que establecen restricciones y relaciones entre los distintos campos de las distintas tablas. Gracias a las claves foráneas se tiene una base de datos dinámica que actualiza todos los parámetros necesarios al momento de un cambio en las tablas.
Esto permite que, gracias a la opción *cascade* de MySQL, se efectúe un cambio en un campo de una tabla, y automáticamente, la base de datos actualice todas las tablas que se encuentren referenciadas al campo de la primera.

A la hora de programar, estas claves son realmente de gran ayuda ya que permiten ahorrar líneas de código (al actualizar un campo, lo se actualiza únicamente en la tabla a la cual hacen referencia las demás y no en todas las tablas donde se encuentra este campo), pero más importante aún, permiten encontrar fallas en el trabajo con la base de datos, ya que restringen cambios en las tablas que luego podrían presentar incoherencias dentro de la base.

A continuación procedemos con el detalle de cada tabla que se encuentra en la base de datos SiMSI.

Alarma_datos

Esta tabla es la principal para el uso de las alarmas en la aplicación. En ella se encuentran los datos principales de las alarmas como son su Nombre (nombre_alarma), el Tipo de Dato que manejan (tipo_dato), los valores límite de las mismas (maximo, minimo) y el campo activada.

Este último campo es el que indica si la alarma se encuentra habilitada o no para hacer el control a la hora de evaluar los datos de los motes y routers.

Si la alarma tiene el campo activada en 0, significa que no se verifican los datos recibidos con esta alarma.

El parámetro primario de esta tabla es el nombre de la alarma (nombre_alarma) ya que es el único parámetro que debe ser único para cada alarma.

Las demás tablas de alarmas se referencian a ella (gracias a la clave foránea que mantienen con el campo nombre_alarma), por lo tanto cuando se edita o elimina una fila de esta tabla (lo que representa una alarma) sucede lo mismo en las tablas subyacentes como son Alarma_usuarios, Alarma_equipos y Alarma_estado.

Alarma_usuarios

Alarma_usuarios presenta, como se puede deducir de su nombre, la conexión entre las alarmas y los usuarios.

En esta tabla se guardan las opciones de notificación en caso de que una alarma se active.

Los campos nombre_alarma y usuario relacionan la alarma con un usuario, y los campos mail y celular indican si el usuario debe ser notificado por mail y/o a su celular cuando se activa una alarma.

Como se vió en la descripción de la tabla anterior, el campo nombre_alarma se encuentra ligado al campo con el mismo nombre de la tabla Alarma_datos. Por su parte, esta tabla presenta otra clave foránea entre el campo usuario y el campo nombre de la tabla Usuarios.

Si se modifica el nombre del usuario que debe ser notificado por la activación de una alarma, en la tabla Alarma_usuarios también se cambiará el campo usuario.

Esta tabla puede tener el mismo valor en el campo nombre_alarma en más de una fila (si existe más de un usuario a notificar para una alarma) o el mismo valor repetido en el campo usuario (un usuario es notificado por más de una alarma). Por lo tanto, para poder indexar, se creó el campo id_alarma2 que es el campo primario. Este índice se autoincrementa a medida que crece la tabla de manera que tenga un valor único por fila.

Alarma_equipos

La tabla Alarma_equipos es una tabla bastante reducida en campos. Tiene solamente 3: nombre_alarma, router_ip y el campo id_alarma3.

En esta tabla se asocia una alarma (nombre_alarma) con la dirección IP de un router de la red (router_ip). Esta asociación se genera para que la alarma actúe sobre todos los datos que llegan de ese router.

El campo nombre_alarma se encuentra ligado al campo del mismo nombre de la tabla Alarma_datos con una clave foránea, y el campo router_ip al campo ip de la tabla routers.

De la misma forma que para Alarma_usuarios, se debió crear un campo que indexara la tabla. Este campo es el llamado id_alarma3 que se autoincrementa cada vez que se agrega una fila a la tabla.

Logs_alarmas

Aquí se encuentra el log de las diferentes alarmas donde se almacenan las activaciones y desactivaciones.

El campo equipo_id se refiere al id del causante de la activación de la alarma. Si se trata de una alarma Estado Enlace, este campo hace referencia a la dirección IP del router. Cuando el nombre_alarma es Sensor Id, almacenamos aquí el sensorID que hace activar la alarma. Y finalmente, existe la opción de guardar el moteID en caso de que la alarma sea de Batería u otra establecida por el usuario de la aplicación Web.

Existen también los campos estado, fecha e id_alarma. Estos campos refieren a si la alarma se activó o desactivó, a la fecha cuando ello ocurrió y a un id para poder indexar la tabla respectivamente.

Se pueden observar las filas guardadas en esta tabla en la sección 4.2.

Alarma_estado

En esta tabla se recaban las activaciones de las alarmas. Cuando un mote o un router sobrepasa un valor límite de alguna alarma que tiene asignada, se guarda una fila en esta tabla. Esa fila se elimina en el momento que el equipo vuelve a sus parámetros normales, dentro de los límites de la alarma.

El campo nombre_alarma hace referencia, obviamente, al nombre de la alarma activada. Los campos mote_id, sensor_id y router_ip muestran la procedencia y el tipo del dato que está violando los parámetros de la alarma.

Luego se encuentran los datos limite (el límite que se violó), dato (el dato que violó dicho límite) y fecha (la fecha del dato).

El campo estado sirve para asegurarse de que la alarma sea verdadera y no una falsa alarma. Este campo es un contador que varía entre 1 y 3.

Cuando se recibe un dato que viola los límites de la alarma se ingresa una fila en esta tabla con el campo estado en 1. Si el siguiente dato sigue sobrepasando los límites de la alarma, se va aumentando el nivel de la alarma, llegando a 3 como nivel máximo y activando el campo enviada. Se verá el funcionamiento de las alarmas y de la escalada de niveles con más detención en la sección 3.2.5.

El campo enviada muestra entonces cuándo la alarma se activó para este equipo.
Existe también el campo contador que contabiliza la cantidad de equipos que se encuetran con la misma alarma activada.
Finalmente, el campo id_alarma sirve como índice para esta tabla utilizando, como ya se vió en otros casos, la función de autoincremento.

Conf_equipos

En esta tabla se guardan los logs de configuración de equipos. Cuando un usuario configura un equipo como puede ser, router o mote, ya sea para agregar, editar, borrar o alterar valores de frecuencia, esta información se guarda en esta tabla.

Es una tabla crucial para el administrador, debido a que puede llevar control de todo tipo de configuraciones que se realicen y saber perfectamente quién las realizó y qué realizó.

Los campos de esta tabla son: id_conf, id_equipo, parametro, dato_ant, dato_nuevo, fecha, usuario.
El índice primario de la tabla es el campo id_conf. El campo id_equipo contiene ya sea el mote_id o router_id del equipo configurado. En el campo parámetro se insertan los distintos tipos de parámetros asociados a cada equipo, como pueden ser ip, id, marca, modelo, número MAC, longitud, latitud, puerto asignado, para un router, y frecuencia, marca, modelo, longitud, latitud, para un mote. Luego, los campos dato_ant y dato_nuevo indican

los datos anteriores y nuevos respectivamente, correspondientes al parámetro ingresado en el campo parámetro. Cuando se ingresa un equipo por primera vez el campo dato_ant queda vacío.

Los datos de fecha y usuario que realizó dicha configuración se encuentran también en esta tabla (campos fecha y usuario respectivamente). Este último campo tiene una restricción impuesta contra el campo nombre de la tabla usuarios. Es la única clave foránea de esta tabla.

Datos

Es la tabla eje del proyecto. La va completando el demonio a medida que van llegando los datos provenientes de los motes y es leída principalmente por la página reportes.php a la hora de graficar y por mapa.php para obtener los últimos datos de los equipos.

Conceptualmente es una tabla muy simple, pero como está creciendo constantemente hay que tratarla con especial cuidado sobre todo a la hora de elegir los índices.

Para leer la tabla eficientemente es necesario que los campos que estén involucrados en las sentencias select y where, sean índices. Es por esto que los índices elegidos fueron los campos tipo, fecha y mote_id, para poder ir particionando la tabla según estos parámetros.

Los campos existentes en Datos son: mote_id, tipo, valor, fecha, id_dato. La clave primaria de esta tabla es el campo id_dato. El campo mote_id es el indicador del id del mote del dato correspondiente a esa fila. Este campo está restringido por el campo mote_id de la tabla motes. Tipo es el tipo del dato como puede ser por ejemplo, Temperatura, humedad, etc.

El demonio inserta el tipo de dato dependiendo del sensor_id proveniente. A cada sensor_id se le asocia un tipo de dato en la tabla tipos_sensores. El campo tipo tiene una restricción impuesta contra el campo tipo_dato de la tabla tipos_datos.

Continuando con los campos de la tabla, el campo valor contiene el dato en sí mismo y el campo fecha, de tipo datetime (fecha y hora), informa el momento en el que el dato llega al servidor. No se trata del momento en el que fue tomado el dato y éste es un aspecto a mejorar a futuro, haciendo cambios tanto a nivel de la programación de los motes como del demonio. Se ve este tema más en profundidad en la sección 3.2.5.

Existen entonces dos claves foráneas que establecen las restricciones y relaciones de esta tabla, con la tabla motes y tipos_datos.

Disponibilidad

La tabla disponibilidad se utiliza para calcular la disponibilidad de cada router, y de esta forma, dar una idea al usuario sobre la calidad del enlace que hay entre el router en cuestión y el servidor central. Una disponibilidad del 100 % indicaría que todas las veces en las que se envió un ping a ese router se obtuvo una respuesta, mientras que una disponibilidad del 0 % indicaría el caso opuesto.

El valor de disponibilidad (%) se ingresa en el campo disp de la tabla y la dirección IP del router correspondiente se ingresa en el campo router_ip. El campo contador se utiliza para guardar una variable auxiliar que el demonio utiliza para realizar el cálculo de la disponibilidad.

En la sección 3.2.5, donde se describe la clase *GatewayConnection*, se puede ver la fórmula utilizada para el cálculo entre otros detalles. Vale destacar que el valor de disponibilidad se resetea semanalmente permitiendo así al usuario ubicar temporalmente el valor obtenido.

El campo router_ip además de ser clave primaria, es una clave foránea que se encuentra ligada al campo con el nombre ip de la tabla routers.

Logs

Es la tabla de logs de acciones de usuarios. Aquí se tiene información sobre todo tipo de movimientos y acciones que cada usuario realiza. También es utilizada en el home para mostrar los favoritos del usuario.

Los campos existentes en logs son: id_log, accion, descripcion, usuario, fecha, link. La clave primaria de esta tabla es el campo id_log. El campo acción (varchar), indica la acción realizada por el usuario. El campo descripción (varchar), describe brevemente la acción anterior. Luego los campos usuario (varchar) y fecha (datetime), marcan el usuario involucrado y la fecha de la acción respectivamente. El campo link (varchar), es utilizado para la página home.php para asociar uno de los favoritos al link asociado.

Hay una sola restricción que vincula al campo usuario, con el campo nombre de la tabla usuarios.

Motes

Es la tabla en la que se guardan las características relevantes de cada mote. Esta tabla se actualiza desde los dos extremos del sistema, es decir, desde el demonio, principalmente cuando recibe datos de los motes, y también desde la interfaz web. Cuando el demonio recibe un dato del mote chequea si éste se encuentra ingresado en la tabla motes, y de no figurar lo agrega.
Por otra parte desde la interfaz web el usuario puede modificar parámetros como ser la frecuencia, marca, modelo, longitud, latitud.

Los campos disponibles en la tabla Motes son: mote_id, motepadre_id, router, lng, lat, marca, modelo, nuevo, frecuencia, ack, frec_deseada. La clave primaria utilizada fue el campo mote_id debido a que es un valor único. Este campo es un identificador único de cada mote.

En el momento en que le llega un mensaje de nacimiento o de topología al demonio, éste recibe además del mote_id del cual proviene, la información sobre cuál fue el primer salto, es decir el mote padre del mote en cuestión. Este id se inserta en el campo motepadre_id.

El demonio también identifica el router desde dónde proviene el dato a través de su dirección IP y verifica y modifica, en caso de ser necesario, la asociación Mote-Router. La misma se guarda en el campo router de la tabla y es dinámica ya que como los motes forman una red mesh entre ellos, pueden optar por distintos caminos y en particular por distintos gateways en cada mensaje enviado. Con la asociación dinámica estos cambios son transparentes para la aplicación.

Luego el usuario inserta los valores de longitud (lng), latitud (lat), marca (marca), y modelo (modelo) asociados a este mote. El campo nuevo es un booleano que marca si este mote ya fue editado por el usuario o no. Es decir, si al demonio le arriba un dato proveniente de un mote que no figura en la tabla motes, el demonio lo inserta fijando el campo nuevo en 1. Luego en la interfaz web, se le indicará al usuario que hay un mote sin configurar, para que de esta forma pueda configurar los valores de longitud, latitud, etc.

El campo frecuencia contiene el valor de la frecuencia actual en la que está configurado el mote. Cuando el usuario configura una nueva frecuencia, dicho valor se inserta en el campo frec_deseada, seteando a la vez el campo ack en 0. Cuando al demonio le arriba la confirmación de configuración de frecuencia, se actualizan el campo ack en 1 y el campo frecuencia con el valor del campo frec_deseada.

En esta tabla no se consideró necesaria la implementación de restricciones hacia otros campos de otras tablas.

Reportes

Es otra de las tablas con más relevancia del proyecto SiMSI debido a que es la encargada de guardar la información relevante a la hora de generar reportes. Se tiene un registro de las opciones elegidas para cada reporte, para luego ser procesadas ya sea para graficar o para generar tablas ilustrativas.

Los campos involucrados son: idreporte, nombre, fechas, grafico, tipo, hora, motes, desde, hasta. La clave primaria es el campo idreporte. Los otros campos son configurados desde la página reportes.php por el usuario. El campo fechas contiene un string que proviene de la elección de las fechas móviles como ser por ejemplo, Última semana, Últimos tres meses, etc. El campo grafico es un entero en el que los posibles valores son 1, 2 o 3 según el tipo de gráfico elegido por el usuario: Evolución temporal de motes, Evolución temporal de routers, Evolución temporal de una hora. El campo hora será distinto de null sólamente en el caso que el valor del campo grafico sea 3. El campo tipo indica el tipo de dato que se va a procesar en dicho reporte. Este campo está restringido por el campo tipo_dato de la tabla tipos_datos. El campo motes contiene un string de todos los motes, separados por ",", que intervienen en el reporte, elegidos por el usuario a través del árbol desplegable de la página reportes.php.

Luego, los campos desde y hasta son parámetros de tipo datetime que se utilizan solamente en el caso que el reporte sea de datos puntuales ya que en ese caso se generan fechas fijas.

Se utilizó entonces una sola clave foránea para relacionar el campo tipo de esta tabla con el campo tipo_dato de la tabla tipos_datos.

Routers

Esta tabla contiene todos los routers instalados junto con sus principales características. Si bien está relacionada con otras tablas como motes, no se evaluó la opción de poner alguna restricción debido a que el router es el equipo de mayor "jerarquía".

Los campos de esta tabla son: router_id, ip, mac, marca, modelo, lng, lat, enlace, puerto. La clave primaria es el campo IP debido a que es un parámetro único y a la vez es el que mejor representa a un router. Luego los otros campos marcan respectivamente el nombre del router (router_id), el número MAC (mac), la marca (marca), el modelo (modelo), longitud router (lng), latitud (lat), puerto por el cual es posible conectarse al router (puerto), y el estado del enlace (enlace).

Se especifica un puerto para poder conectarse al router ya que es posible configurar en el router conectado al ADSL un puerto especial para que todos los mensajes recibidos en el mismo los reenvíe a cierto IP. Se logra de esta manera, a través de Internet, conectarse vía https a cualquiera de los routers del campo.

Tipos_datos

Es una tabla auxiliar para guardar los tipos de datos que el usuario mismo genera. Cuando el usuario configura un nuevo sensor elige qué tipo de dato mide. Al obtener los tipos de datos de una tabla, se evitan errores de tipeo del usuario al momento de, por ejemplo, ingresar un nuevo sensor y tener una coherencia general de los datos.

En la tabla se guarda también la unidad asociada al tipo de dato.

Esta tabla contiene dos campos: tipo_dato que es la clave primaria debido a que tiene que ser un campo único, y unidad.

Tipos_sensores

Esta tabla contiene los distintos sensores junto con sus parámetros relevantes. Los campos que contiene esta tabla son: nombre_sensor, sensor_id, Vref, A, B, tipo_dato, nuevo. La clave primaria es el campo nombre_sensor debido a que es un campo único. El campo sensor_id es el distintivo que se utiliza en el momento de la comunicación del demonio con el mote. El mote posee sensores para obtener valores para distintos tipos de datos, como pueden ser la temperatura, la humedad. Por lo tanto, envía el dato junto con el sensor_id para que luego sea procesado por el demonio de manera exitosa. Esto ofrece escalabilidad debido a que brinda la posibilidad de agregar nuevos sensores a un mismo mote.

Los campos Vref, A, B, son utilizados a la hora de calcular el dato. El demonio aplica una ecuación con estos parámetros para poder determinar el valor del dato arribado. Esta ecuación la provee el fabricante de los motes, y para hallar el valor exacto de los datos obtenidos puede llegar a ser necesario ajustar los parámetros hasta tener el sensor calibrado. El campo nuevo es utilizado por el demonio para identificar si es la primera vez que se recibe un dato de este sensor_id o no. En caso de ser nuevo se notifica a los usuarios seleccionados en la página de alarmas.

Usuarios

Es la tabla que guarda los distintos usuarios de SiMSI, junto con sus principales características personales así como las distintas potestades que tiene en el sistema. Existen distintos perfiles de usuario: Administrador, que tiene todas las potestades, Básico, que tiene potestades limitadas ya pre-defenidas, y usuarios personalizados para los cuales el Administrador define las potestades.

Los campos contenidos son: nombre, password, telefono, codigo_seg, mail, tipo, id_usuario, conf_router, ver_router, conf_mote, ver_mote, conf_frec, dist_frec, conf_usuario, ver_usuario, logs, conf_reporte, ver_reporte, conf_alarma, ver_alarma, conf_sensor, ver_sensor. El índice primario es el campo id_usuario. Luego los campos nombre, password, telefono, mail son tal cual lo indican su nombre. El campo cod_seg es el código de seguridad necesario para enviar alarmas del tipo SMS al celular. El campo tipo indica qué tipo de usuario es (Administrador, Básico, Personalizado).

Los campos siguientes indican las distintas potestades del usuario. Por ejemplo, si el campo conf_alarma tiene el valor 1, entonces el usuario tiene la potestad de configurar alarmas.

3.2.4. Diagrama de Interacción

A modo de conclusión de este capitulo se muestra en la figura 3.3 el diagrama de interacción de la base de datos en el que se especifican los relacionamientos entre las tablas a través de las claves foráneas.

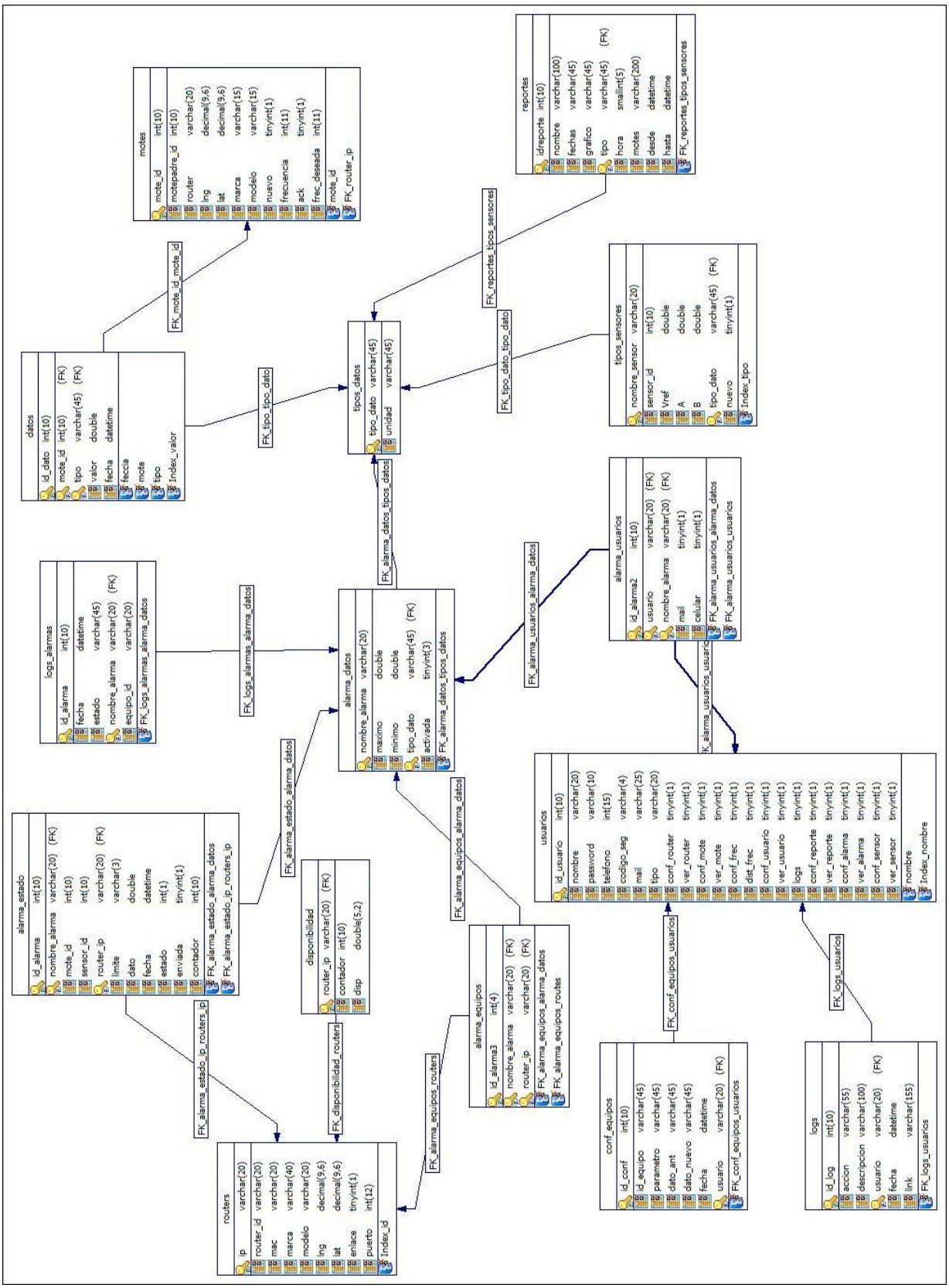

Figura 3.3: Diagrama de Interacción de la Base de Datos

3.2.5. Demonio

Introducción

Un demonio, daemon (de su sigla en inglés Disk And Execution Monitor) es un programa que se ejecuta en segundo plano y no es controlado directamente por el usuario. Generalmente los demonios se inician automáticamente al arranque del sistema operativo, ejecutando procesos de forma continua y que permanecen "invisibles" al usuario [4].

El demonio desarrollado en este proyecto hace de interfaz entre los motes instalados y la base de datos. Está compuesto por diferentes subprocesos internos que se ejecutan en paralelo (threads) y cumplen diferentes funciones como por ejemplo: levantar un túnel entre el servidor y cada mote central o detectar y enviar alarmas por e-mail o sms a los usuarios.

Se eligió Java como lenguaje de programación para el desarrollo. La elección fue principalmente porque ya existe una API (Application Programming Interface) diseñada para la comunicación con los motes. Cabe destacar, que para desarrollar el demonio fue necesario instalar TinyOS en el equipo que se utilizó para programar, además del JDK y un IDE (se utilizó Netbeans). Todas las instalaciones se realizaron según los procedimientos estándar de instalación indicados en las páginas de los desarrolladores.

En las siguientes partes de esta sección se detallan los puntos más importantes referentes al desarrollo del demonio. Primero se hará una breve reseña sobre Java, el por qué de la elección y sus ventajas. Luego se describirá cómo hace de interfaz entre motes y base de datos. Se estudiará su implementación y programación, cada una de sus clases y funcionalidades, incluyendo algunos puntos importantes que fueron tenidos en cuenta para su desarrollo. Se analizará, especialmente las alarmas, cómo se generan, cómo notifican al usuario y su registro en el log. Finalmente se verá cómo es su ejecución e inicio, la salida que puede ver el usuario, su rendimiento y recursos consumidos.

Java

Java es un lenguaje de programación orientado a objetos desarrollado originalmente por James Gosling y sus colegas en la empresa Sun Microsystems a principios de los años 90.

Fue concebido con los siguientes objetivos:

- usar la metodología de programación orientada a objetos
- permitir al mismo programa ejecutar en diferentes plataformas de computación
- soporte interno para el uso en redes de computadoras
- ejecutar en forma segura código residente en máquinas remotas

[5]

¿Por qué programar el demonio en Java?

Como se menciona en la introducción a esta sección, el motivo principal es porque ya existe una interfaz en Java diseñada específicamente para la comunicación con los motes y su sistema operativo TinyOS. Particularmente el paquete utilizado de la API para la comunicación es el net.tinyos.message.

Además de este motivo principal existen otras ventajas adicionales como su portabilidad, que permitiría ejecutar el demonio en servidores con diferentes plataformas y sistemas operativos sin necesidad de reprogramar la aplicación (de todos modos, SiMSI está pensado para correr sobre Windows). Adicionalmente existe una gran cantidad de conocimiento y recursos sobre Java, ya sea en la web, foros, libros o ingenieros conocidos de dónde

aprender y aclarar dudas. Finalmente el curso de Desarrollo de Software para Ingeniería Eléctrica cursado en facultad aportó las herramientas necesarias como para programar sin necesidad de invertir tiempo extra en investigación y aprendizaje.

El Demonio como Interfaz

Como se puede ver en la figura 3.4 el demonio se encuentra entre medio de la comunicación entre la red de sensores y la base de datos. En un sentido (derecha a izquierda viendo la figura) recorre la base de datos para recopilar datos de la estructura de la red, usuarios, cambios en período de muestreo y demás, para luego levantar túneles, solicitar datos o enviar cambios en la configuración del lado de la red. En el otro sentido (izquierda a derecha según figura) recibe diferentes tipos de mensajes donde reconoce a qué tipo pertenecen (datos, ack, sensado batería y otros), luego identifica el router desde donde llegó el dato, desde qué moteId (número identificador del mote) e incluso desde qué sensorId (número identificador del sensor).

Adicionalmente, debido a los mensajes de alarma, el demonio se comunica directamente con el usuario final al enviar mensajes de texto o e-mails en caso de activarse una alarma o para enviar un resumen semanal de las mismas.

Para la comunicación con la red de sensores se utilizó el paquete de la API net.tinyos.message, mientras que para el envío de alarmas y resúmenes por e-mail o sms se utilizó el javax.mail. Finalmente para la comunicación con MySQL se cargó el driver JDBC llamado MySQL Connector/J.

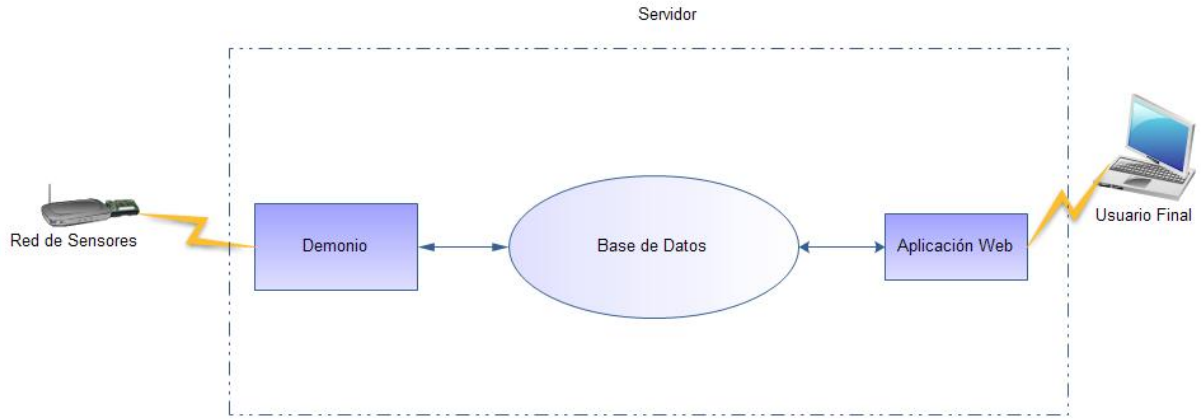

Figura 3.4: Estructura del Servidor

API - net.tinyos.message Debido a su particularidad y aplicación específica para este proyecto se muestra un poco más en detalle el resumen de clases e interfaces de la API utilizada para lograr la comunicación con los motes.

Package net.tinyos.message

Interface Summary	
MessageListener	MessageListener interface (listen to tinyos messages).

Class Summary	
Message	
MoteIF	MoteIF provides an application-level Java interface for receiving messages from, and sending messages to, a mote through a serial port, TCP connection, or some other means of connectivity.
Receiver	Receiver class (receive tinyos messages).
Sender	Sender class (send tinyos messages).
SerialPacket	

[6]

La interfaz MessageListener es utilizada para reconocer y recepcionar mensajes de los motes. Dichos mensajes son definidos por la clase Message y para el demonio se tomaron específicamente los mensajes diseñados por el grupo RSIS llamados: MensajeMote y MensajePC que heredan de la clase Message.

MoteIF es la clase que permite la comunicación con los motes por medio de los mensajes mencionados.

Finalmente las clases Receiver y Sender son utilizadas para recepcionar o enviar mensajes desde o hacia los motes.

La clase SerialPacket no fue utilizada.

Desarrollo y programación del Demonio

La programación y clases del demonio SiMSI se origina de una primera versión de las clases MsgReader y MsgSender que Pablo Mazzara gentilmente nos cedió y explicó.

A partir de allí el demonio fue creciendo en clases, threads y complejidad hasta llegar a lo que es hoy en día. Las diferentes funcionalidades que posee, se fueron agregando gradualmente a medida que surgían ideas de nuestra parte, de los docentes y de la interacción con RSIS.

Creemos que el demonio obtenido posee una estructura sólida, con un excelente rendimiento y buen manejo de excepciones. Debido justamente a que no fue diseñado con una idea clara de su estructura desde el principio, quizá se pueda considerar como debilidad que su diseño no fue planificado desde el comienzo sino que se fueron agregando clases y threads sin considerar en profundidad su modularidad y flexibilidad frente a futuros cambios. Aún así, es sencillo agregar nuevas ejecuciones en paralelo a las ya implementadas y aplicar cambios o modificaciones estructurales a las ya desarrolladas, por lo que el impacto es menor.

El demonio está compuesto por siete threads (hilos de ejecución) principales, otros threads secundarios cuya cantidad depende de la cantidad de enlaces disponibles, y otras clases auxiliares como por ejemplo Alarm.java

que define las alarmas con sus atributos y métodos correspondientes.

La gran ventaja de utilizar threads es que éstos permiten tener varias ejecuciones en paralelo corriendo en forma concurrente.

Además, la idea de los threads es que se ejecuten en loop, repitiendo su tarea cada cierto período de tiempo configurado a través del método sleep. Algunos threads son configurados en loop infinito para que ejecuten sus tareas indefinidamente, mientras que otros conviene se ejecuten en loop mientras se cumpla una condición, como por ejemplo: mientras exista conexión a un determinado router.

Se puede observar en la figura 3.5 un pequeño diagrama de las clases que componen al demonio (faltan las clases MensajeMote.java y MensajePC.java que fueron diseñadas por RSIS que se analizarán más adelante en esta sección).

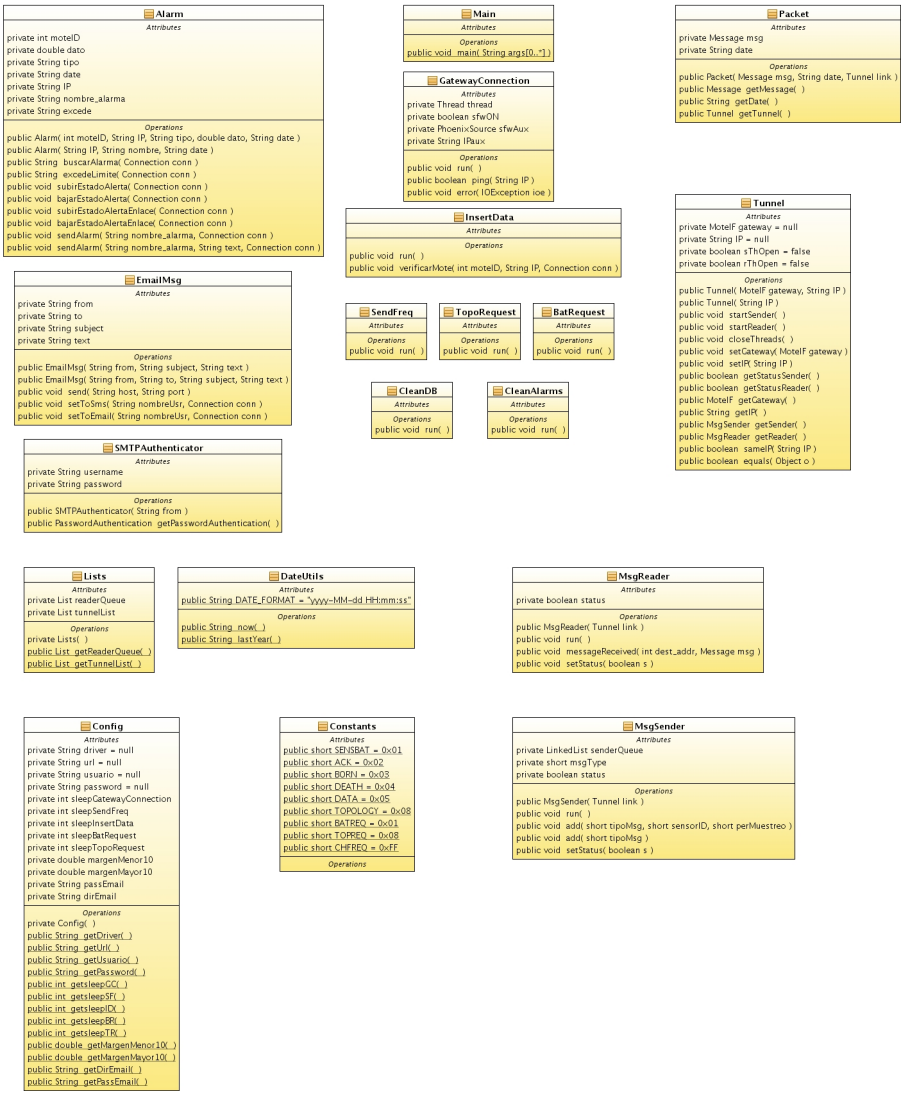

Figura 3.5: Clases de Java del Demonio

El método *main*, que inicializa al demonio, invoca a los siguientes threads que consideramos principales:

GatewayConnection

InsertData
SendFreq
BatRequest
TopoRequest
CleanAlarms
CleanDB

Los threads son clases que implementan la interfaz *Runnable* y luego implementan el método *run()* que es aquél donde se lleva a cabo la ejecución del thread.

Por ejemplo:

```
public class GatewayConnection implements Runnable {
@Override
public void run(){
...
 }
}
```

Para su ejecución se crea la instancia de *Thread* y se ejecuta invocando al método *start()*.

Por ejemplo:

```
...
new Thread(new GatewayConnection()).start();
...
```

Se explica a continuación qué función cumple cada una de las clases:

GatewayConnection Thread que se ocupa de levantar conexiones con routers ingresados por el usuario en la web y los deja a la escucha de nuevos mensajes arribados.

Esta clase es de vital importancia, ya que administrar las conexiones con la red de sensores es el primer paso para luego poder recibir mensajes desde los motes.

Su ejecución puede resumirse en tres etapas:

1. Verifica si hay enlace de red con cada uno de los routers ingresados por el usuario. Para ello se envían pings a cada router, y si más de la mitad de los pings no regresan se considera la conexión como perdida.
2. Se actualiza el valor de disponibilidad. La idea de este valor es dar una noción al usuario sobre la disponibilidad semanal de cada router.

   La fórmula empleada para la disponibilidad es:

   `disp = (((dispAnt/100)*(cont-1) + conex)/cont)*100;`

   donde: `conex` vale 1 o 0 según si en la etapa i) se determinó la conexión como perdida (0) o como activa (1).

   `cont` cuenta la cantidad de veces que se calculó la disponibilidad en la semana

   `dispAnt` es el valor anterior de disponibilidad.
3. Se sincroniza la lista de routers ingresados por el usuario (y con conexión activa) con la lista de túneles ya abiertos. Si aparece un nuevo router, abre un nuevo túnel con el mote central correspondiente o cierra un túnel ya abierto si el usuario lo quitó de la lista de routers.

Para abrir un túnel se levanta primero una conexión con el mote central de cada router y luego se inicia el thread *MsgReader.java*, que queda a la espera de un nuevo mensaje de los motes y lo ingresa en una cola de llegada llamada *readerQueue*.

InsertData Thread que cumple como función tomar mensajes que están en la cola de llegada *readerQueue*, identificar qué tipo de mensaje es (sensado batería, ack, nacimiento, muerte, topología o datos) y tomar las acciones correspondientes según el caso.

A continuación se analiza cada caso:

1. ack (acknowledge o reconocimiento):
   Recibir este mensaje significa recibir una confirmación de cambio de configuración de un mote.
   Al detectar este mensaje se identifica mote origen y se actualiza en la base de datos que ese mote efectivamente ha cambiado su configuración.

2. sensado batería:
   Al llegar este mensaje se identifica el voltaje al que está alimentado cada mote. Se actualizan los datos y en caso de que el voltaje esté por debajo de cierto umbral se dispara una alarma para notificar a los usuarios.

3. nacimiento:
   Aquí se reconoce que ha nacido un mote y particularmente se actualiza quién es su mote padre.

4. muerte:
   Cuando se detecta un mensaje de muerte simplemente se marca que ese mote ahora no tiene padre. No se borra de la base porque se asume que probablemente renacerá más adelante. Si no se especifica en la aplicación que el mote se elimina, este mensaje puede crearse debido a problemas en la conexión entre los motes (por ejemplo, necesidad de recambio de baterías en mote hijo o aumento de la pérdida de ganancia en el aire), lo que lleva a que pueda reconectarse a la red 802.15.4.

5. topología:
   Estos mensajes son generados por el demonio mismo luego de enviar un mensaje de solicitud de topología con el thread *TopoRequest*. Básicamente se toman las mismas acciones que con nacimiento, se actualiza quién es el padre actual del mote.

6. datos:
   En este caso hay varias tareas que se ejecutan:
   Primero se identifica el sensor del cual proviene el dato, y si no está ingresado en la tabla se activa la alarma correspondiente. Es fundamental configurar el sensor, ya que el dato se obtiene operando el valor recibido con ciertos parámetros característicos. Si estos parámetros no están configurados no se obtendría un valor correcto de temperatura por ejemplo. En definitiva, si el sensor no tiene estos parámetros configurados no se ingresa en la base. Además hay que asociar cada sensor a un tipo de dato como ser Temperatura o Humedad, que permite luego, representar los datos según este tipo de dato independientemente de cuál sea su sensorId.

   Una vez configurado el sensor, cuando llega el dato se ingresa en la tabla según los parámetros característicos recién mencionados. Si se identifica que el dato está en condición de alarma, según alguna de las alarmas configuradas por el usuario, la misma se activa.

En todos los casos al llegar el dato primero se verifica si el moteId ya está ingresado en la base de datos y si no, lo ingresa junto con el IP del router desde el cual llega el dato (para saber a qué zona está asociado). Además lo marca como nuevo, de forma que el usuario lo reconozca fácilmente en la aplicación y pueda configurarle manualmente otros datos como ser su ubicación geográfica (que le permitirá verlo en el mapa).

En el caso particular de que el moteId no esté ingresado y sea un mensaje de nacimiento, se configura para esperar un reconocimiento de cambio de configuración con la misma frecuencia que alguno de los otros motes de la misma zona. Esto se hace porque el protocolo diseñado por RSIS configura a un mote nuevo con la misma

configuración que su padre (mantiene configuración de la zona donde nace).

Sendfreq Thread que tiene como objetivo enviar mensajes solicitando el cambio del período de muestreo sobre los sensores de la red.

Para lograrlo, se fija en la tabla motes de que moteId tiene pendiente recibir un ack, si existe alguno, envía el mensaje a través del router (sólo si está en la lista de túneles) que tenga asociado el mote. No es necesario un mensaje por cada mote ya que los mensajes son siempre broadcast y no puede haber más de una frecuencia por cluster para un mismo sensor.

Dado que hasta el momento los motes programados por RSIS sólo trabajan con un mismo sensor de temperatura (sensorId = 0x01) se asume este valor y no se implementó la posibilidad de enviar cambios de configuración para otros sensores. Sería un interesante trabajo a futuro permitir que los motes trabajen con más de un sensorId y enviar un mensaje específico a cada router según sensor a configurar.

Debido a que no se trabaja con números de secuencia para este tipo de mensajes, puede ocurrir de enviar un cambio de configuración antes de haber recibido el ack anterior, y si luego de esto llega un ack, tomarlo como que corresponde a la última frecuencia ingresada.

Este es otro aspecto a mejorar en futuras implementaciones y del punto de vista de SiMSI sería muy sencillo implementar la mejora ya que consistiría en seguir la secuencia de los mensajes. Restaría implementarlo del punto de vista de los motes. De cualquier forma, dado que el tiempo entre cada ejecución del thread SendFreq puede regularse fácilmente, ajustándolo a un tiempo mayor que el tiempo máximo que puede demorar un ack se soluciona el problema. No es una solución rigurosa pero al menos es una rápida solución. Además de que se entiende que el cambio de frecuencia no es una acción que el usuario ejecute frecuentemente.

BatRequest Thread que envía mensaje de solicitud de estado de baterías. Es bastante sencillo, simplemente busca todos los túneles disponibles hasta el momento y envía un mensaje a cada uno (broadcast).

Genera que todos los motes envíen un mensaje indicando el voltaje actual de sus baterías.

TopoRequest Idéntico a BatRequest pero envía mensaje de solicitud de topología.

Genera que los motes envíen un mensaje indicando el id de su padre.

CleanAlarms Thread que ejecuta una limpieza de las alarmas activadas y envía resumen por e-mail con información sobre alarmas de batería, estado de enlace y sensor id.

La idea de este thread es que se ejecute por ejemplo cada una semana. De esta manera se recuerda al usuario las alarmas que hay activadas y se borran alarmas que hayan perdido vigencia y que por algún motivo no hayan sido removidas.

Se genera un resumen para cada uno de los tres tipos de alarma: batería, sensor id y estado de enlace y le llegará al usuario que tenga configurada la notificación vía e-mail en la página de configuración de la respectiva alarma.

Las alarmas creadas por el usuario y que estén activas también se borran, pero no se envía un resumen, simplemente si la alarma sigue vigente ésta se activará nuevamente de forma automática y notificará al usuario según su configuración.

A modo de ejemplo se muestra un resumen de alarma de estado de enlace en la figura 3.6.

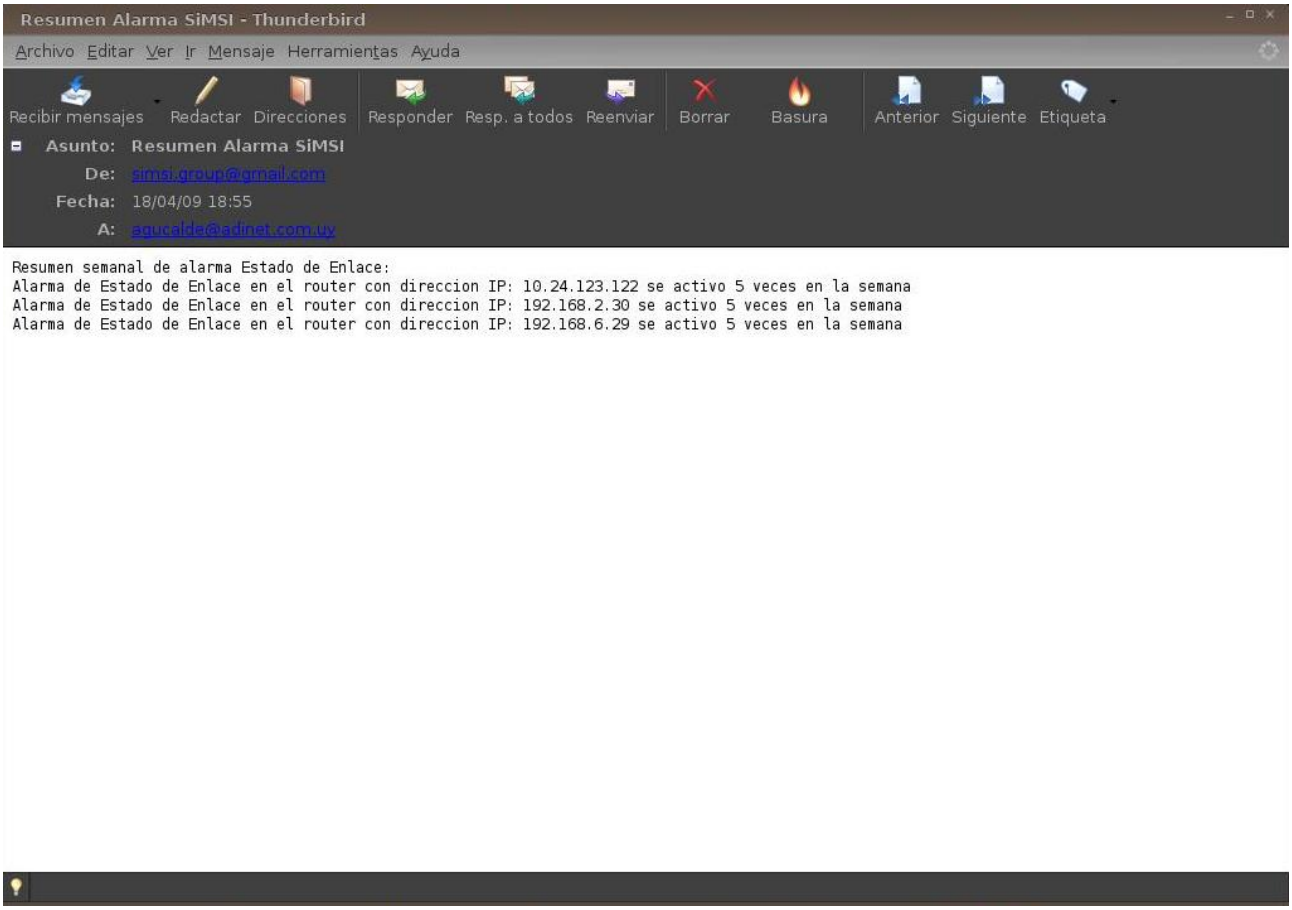

Figura 3.6: Correo electrónico de Resumen de Alarmas

CleanDB Thread que realiza limpieza de datos de la base anteriores a 1 año.

La lógica de esta ejecución es obtener todos los datos anteriores a un año, agruparlos en períodos de quince días y promediar su valor obteniendo solamente un dato promediado cada quince días. De esta forma se logra un mantenimiento de la base de datos, se controla el espacio en disco, el tamaño de la tabla datos y su eficiencia.

Este thread está configurado para ejecutarse cada un mes.

Existen también otros dos threads utilizados para recibir mensajes de los motes o enviar mensajes:

MesageReader Thread que está siempre escuchando y recibiendo mensajes del mote. Básicamente lo que hace este thread es mantenerse en loop ejecutando un "listener". Cuando recibe un mensaje se ejecuta el método *messageReceived(int dest_addr, Message msg)* que ingresa el mensaje en la cola *readerQueue*. Para que esto sea posible es necesario que la clase implemente la interface *MessageListener*, perteneciente al paquete net.tinyos.message.

Un punto interesante a tener en cuenta es que habrá un thread *MsgReader* por cada enlace levantado, es decir, un *MsgReader* por cada mote central. Además es posible cerrar el thread, y de hecho se hace, cada vez

que se cierra un túnel.

MsgSender Thread utilizado para enviar mensajes al mote. Se mantiene en loop recorriendo la cola *senderQueue* y cuando encuentra un mensaje lo envía. Posee dos métodos add para ingresar mensajes en la cola que se utilizan según el tipo de mensaje a enviar.

Al igual que con *MsgReader* hay un thread por cada enlace y se cierra el thread una vez cerrado el enlace.

Además de los threads hay clases auxiliares que complementan al demonio:

Tunnel Clase que define el objeto Tunnel que posee atributos relevantes a la conexión con cada cluster de la red de sensores. El atributo *gateway* que es del tipo MoteIF guarda el enlace creado con cada mote central y el atributo *IP* guarda la dirección IP del router al que está conectado dicho mote. La clase posee los métodos que permiten abrir y cerrar threads *MsgReader* y *MsgSender*, además de algunos otros métodos auxiliares.

Lists Clase que define listas (arreglos o colas) que serán usadas por diferentes threads.

En todo el demonio hay dos listas que pueden ser accedidas concurrentemente por varios threads, estas listas son *readerQueue* y *tunnelList*. La primera guarda los mensajes que llegan desde los diferentes motes centrales instalados y la segunda almacena instancias de la clase *Tunnel*.

El thread *InsertData* está constantemente leyendo la cola *readerQueue* a la espera de algún mensaje, mientras que los diferentes threads abiertos según cada enlace (*MsgReader*) guardan los mensajes que reciben en la misma cola.

Por otra parte, el thread *GatewayConnection* sincroniza la lista de routers ingresados por el usuario con los que realmente hay conexión y los guarda en la cola *tunnelList*. Al mismo tiempo los threads *SendFreq*, *BatRequest* y *TopoRequest* obtienen los túneles de esta lista para enviar mensajes.

En ambos casos el hecho de que dos o más threads accedan o cambien el tamaño de la misma lista simultáneamente podría generar un problema grave de sincronización. Es por ello que es necesario utilizar una lista sincronizada.

Las opciones más adecuadas para solucionar este asunto es o utilizar un Vector o utilizar ArrayList sincronizado. Existen algunas diferencias entre ambas opciones que se verán a continuación.

La clase Vector implementa arreglos ya sincronizados mientras que la clase ArrayList implementa arreglos no sincronizados a menos que se utilice la siguiente línea de código:

```
List list = Collections.synchronizedList(new ArrayList(...));
```

De esta forma se obtiene un arreglo (instancia de la clase List) sincronizado. Adicionalmente es necesario antes de iterar sobre la lista en algún thread, incluirlas en un bloque que las sincronice:

```
...

synchronized(list) {
Iterator i = list.iterator();
while (i.hasNext())
```

...

En cuanto a performance, trabajar sobre instancias de Vector es algo más rápido que sobre instancias de ArrayList sincronizadas.

Todo esto en principio daría la pauta de usar Vector, ya que no sólo ahorraría algo de trabajo a la hora de escribir el código sino que se tendría una mejor performance, sin embargo se optó por usar ArrayList. La decisión se debe a que la clase Vector es algo antigua y hay una tendencia a que sea sustituida por clases como ArrayList. De hecho, la idea de que ArrayList no sea necesariamente sincronizada es a propósito para darle a elegir al programador si sincronizar o no según lo que necesite (ArrayList sin sincronizar tiene una mejor performance que Vector) y se ve como un avance respecto a Vector. En este caso, dado que el tamaño de las listas será bastante pequeño creemos que la performance no será muy diferente entre las dos opciones. Tampoco era demasiado trabajo escribir los bloques de sincronización, así que se prefirió utilizar ArrayList.

Hay varios foros en Internet donde programadores de todo el mundo discuten sobre este tema [7].

Finalmente y cerrando el tema de las listas sincronizadas, si una lista es accedida por más de un thread simultáneamente se lanzará una excepción *ConcurrentModificationException*. Si bien es importante tratar esta excepción, no debe tomarse como una verdadera referencia ya que podría lanzarse aunque no haya ocurrido un acceso simultáneo realmente (fail-fast behavior).

DateUtils Clase utilizada que permite obtener y operar con el día y la hora a través de sus métodos.

Packet Clase que define objeto Packet que contiene atributos relevantes a un mensaje recibido desde un mote como ser: el mensaje arribado en sí, fecha y el túnel por el cual llegó (IP e interfaz moteIF). Instancias de esta clase se guardan en la cola *readerQueue* y permiten luego, en el thread *InsertData*, obtener la información de cada paquete.

EmailMsg Esta clase permite crear los e-mails o sms que serán enviados al usuario como notificación de alarma. Contiene métodos que permiten enviar el mensaje y completar el destinatario según sea e-mail o sms (en sms varía según proveedor de telefonía).

SMTPAuthenticator Clase que permite crear instancias de autenticación y autenticar al usuario en el servidor de correo.

Alarm La clase *Alarm* permite crear instancias de alarma con los atributos que la definen. Además posee varios métodos con los cuales se puede modificar el status de la alarma, activarla, desactivarla y registrar eventos en un log. Dado que el tema de las Alarmas es bastante complejo y una de las funcionalidades más importantes e interesantes de SiMSI, los métodos implementados y todo el tema en general se analizarán en profundidad más adelante en esta documentación.

Constants Clase donde se declaran constantes a utilizar en distintas otras clases del demonio. Varias de estas constantes fueron definidas por el grupo RSIS de forma de tener un protocolo común a la hora de intercambiar mensajes con los motes.

Config Clase que se utiliza para capturar ciertas constantes cuyos valores serán tomados de un archivo txt *"simsi.properties"*.

simsi.properties Es un archivo de texto donde se pueden modificar ciertas constantes que serán utilizadas por el demonio. La ventaja de usar este archivo es que se puede modificar y no será necesario recompilar el software. Por ejemplo, se puede cambiar fácilmente el usuario y contraseña utilizados para conectar a la base de datos. También pueden cambiarse los tiempos que permanecen dormidos distintos threads del demonio entre otras constantes.

MensajeMote y MensajePC Ambas clases fueron provistas por el grupo RSIS y definen el formato de mensajes que intercambia el demonio con la red de sensores. Son el vínculo entre RSIS y SiMSI.

Para finalizar esta descripción de las clases y funcionalidades del demonio, es importante aclarar que actualmente la fecha obtenida de los mensajes arribados es la fecha en la que son recibidos los mensajes por el demonio, y no la fecha de cuando fueron tomadas las medidas. La idea para obtener este dato, que por cierto es relevante, es crear un thread donde se envíe (una vez por día por ejemplo) día y hora actual a la red de sensores de forma de sincronizar y corregir posibles defasajes. Luego habría que agregar un línea de la forma: *msg.getDate()*, siendo *msg* el mensaje recibido, y obtendríamos día y hora de realizada la medida.

Por el lado de nuestro sistema implementar esta funcionalidad no sería muy complejo y creemos en uno o dos días de trabajo estaría pronto. Sin embargo, antes habría que implementarlo sobre los motes, de forma que puedan procesar el mensaje de sincronización recibido, guardar fecha en que se toma cada medida y proveer una función para obtener esta información a través del mensaje recibido por el demonio. Durante el transcurso del proyecto se habló de este tema con el grupo RSIS y nuestros tutores pero finalmente no dieron los tiempos para implementarlo por parte de RSIS y se decidió no implementarlo hasta que esté bien definido el formato de mensajes referentes a este tema.

Alarmas

Existen cuatro tipos de Alarmas que pueden ser generadas en SiMSI: Datos, Batería, SensorId y Estado de Enlace. A continuación se describe cada una ellas y su relación con el demonio.

Datos Este tipo de alarmas son ingresadas por el usuario desde la página Web. En ella se ingresan los valores límites que definen el grupo de valores bajo el cual se considera que un dato está en condición de alarma.

El usuario puede ingresar varias alarmas de este tipo, sin embargo, no puede configurar más de una alarma de un mismo tipo de dato por cluster o zona. Esto es porque no se encontró necesidad de configurar por ejemplo, dos alarmas de temperatura diferentes para una misma zona, sobretodo teniendo en cuenta que en una alarma se puede configurar tanto el mínimo como el máximo aceptado.

Cuando el demonio recibe un dato, verifica si existe una alarma en esa zona para ese tipo de dato mediante el método *buscarAlarma(Connection conn)*. Si existe, chequea si el dato recibido cumple condición de alarma o no con el método *excedeLimite(Connection conn)*. Luego en caso de cumplir alguna condición sube el estado de alerta mediante el método *subirEstadoAlerta(Connection conn)*.

El estado de alerta se define como un estado intermedio donde se han detectado uno o dos datos consecutivos que cumplen con la condición de alarma. La idea de este estado es filtrar la posibilidad de que se active una

alarma si un dato o dos llegan cumpliendo la condición de alarma por error o por algún evento transitorio. Al tercer dato consecutivo arribado que cumpla la condición sí se activa la alarma y notifica al usuario. De esta manera, con el estado de alerta evitamos alertar al usuario sobre una alarma inexistente o esporádica. Si la alarma está activada y el dato arribado no cumple condición de alarma, se baja un punto el nivel en estado de alerta utilizando el método *bajarEstadoAlerta(Connection conn)* pero sin desactivar la alarma.

El estado de alerta se controla con una variable llamada *estado* que se incrementa o disminuye si llega un valor en condición de alarma o no respectivamente. Cuando *estado* alcanza el valor tres se activa la alarma, cuando llega al valor uno y disminuye una vez más, se desactiva si es que estaba activada. Esta disminución del contador de tres a cero se utiliza ya que podría ocurrir que el valor fluctúe por encima y por debajo de la cota en cada mensaje y se mantenga siempre en estado de alerta sin activar o desactivar la alarma.

Cuando el período de muestreo es mayor a treinta minutos se decidió no utilizar el paso intermedio de estado de alerta ya que para que se active la alarma sería necesaria al menos una hora y media y esto podría repercutir gravemente en el usuario si necesita tomar una acción en el corto plazo.

Una vez que está activada la alarma se envía notificación vía e-mail o sms a los usuarios seleccionados en la página de la alarma.

Otro punto interesante a tener en cuenta es que al bajar el estado de alerta hay dos casos posibles:

1. El valor arribado cae dentro de un margen por debajo de la cota superior o por encima de la cota inferior.
2. El valor cae fuera del rango de valores definidos por un margen.

En el caso *1* sube o baja el estado de alerta, mientras que en el segundo caso se desactiva la alarma directamente. Por ejemplo, si se tiene una alarma de temperatura para temperaturas mayores a 30°C activada y llega un valor de 28°C se decrementa la variable *estado* al nivel dos, y si bien la alarma sigue activa entramos en estado de alerta, de forma de que si recibimos dos valores más de 28°C se desactiva la alarma. Sin embargo, si se recibe un dato de 25°C, la alarma se desactiva directamente salteando el estado de alerta. Esto es porque para valores mayores a 10 se definió un margen del 10 % sobre la cota configurada en la alarma. En este caso, como la cota superior es de 30°C se tiene un rango de valores de entre 27°C y 30°C donde la alarma se mantiene activa pero pasa a estado de alerta. Para valores menores a 27°C la alarma se desactiva directamente sin pasar por el estado de alerta. Para valores límite de valor absoluto menor que 10 se toma un margen fijo de 1 debido a que un 10 % podría ser poco representativo. De cualquier forma ambos márgenes pueden ser configurados en el archivo *simsi.properties* modificando la variables *margenMayDiez* (en %) y *margenMenDiez* (en valor absoluto).

Este comportamiento de las alarmas y estado de alerta se puede entender fácilmente haciendo una analogía con el funcionamiento de un circuito Trigger Schmitt.

Alarma Batería La alarma batería tiene un funcionamiento bastante simple. Si el valor arribado correspondiente a la carga de la batería es menor a cierto valor configurado, se activa la alarma para ese mote, se notifica a los usuarios correspondientes y se guarda un registro en el log.

Si la alarma estaba activada para determinado mote y llega un valor por encima del umbral configurado (por ejemplo porque se cambiaron las baterías de ese mote), se desactiva la alarma y registra el evento en el log.

Alarma SensorId La razón por la que existe esta alarma es como se explicó al describir el thread *InsertData*, porque cada sensor debe tener ciertos parámetros característicos configurados. Esta alarma se activa, notifica

a los usuarios y guarda un evento en el log ni bien llega el dato con un sensorID no configurado. Sólo se envía notificación una vez, o si vuelve a llegar un dato de un sensorID no configurado luego de ejecutarse el thread *CleanAlarms*. Una vez que el sensor se configura, cuando llega un dato con ese sensorID se desactiva la alarma y guarda un registro en el log.

Alarma Estado de Enlace Esta alarma se activa cuando se detecta que no hay conexión con alguno de los routers configurados por el usuario en el sistema. Su funcionamiento es muy similar al de Alarma Datos ya que la alarma no se activa o desactiva directamente sino que primero pasa por un estado de alerta.

La verificación de si hay conexión o no con cada router se realiza cada vez que se ejecuta el thread *GatewayConnection*, por lo que será necesario que el thread se ejecute tres veces y no establezca conexión en las tres para activar una alarma de este tipo. Se considera que en este caso no hay una urgencia tan importante como para el caso de Alarma Datos, se entiende que el usuario puede esperar el tiempo correspondiente hasta ser notificado. Por defecto *GatewayConnection* se ejecuta cada diez minutos, por lo tanto si realmente se ha perdido comunicación con un router, el usuario será notificado a la media hora. Se cree que esto además es una ventaja porque podría perderse momentáneamente la conexión hacia un router justo al hacer el chequeo pero esto no implica que se haya perdido conexión definitivamente.

A diferencia también de la alarma datos no se establece un margen para saltear directamente el estado de alerta, siempre tiene que pasar por este estado para desactivarse. Los métodos de la clase *Alarm* utilizados para variar el estado de esta alarma son: *subirEstadoAlertaEnlace(Connection conn)* y *bajarEstadoAlertaEnlace(Connection conn)*.

Al igual que el resto de las alarmas, al activarse se notifica a los usuarios configurados y se guarda un registro en el log tanto al activarse como desactivarse.

Ejecución

La ejecución del demonio está configurada para que se haga en forma automática por el sistema operativo al iniciarse. Puede observarse una ventana de comandos de Windows minimizada que si restauramos mostrará diferentes mensajes de salida del demonio (figura 3.7).

La idea es que, en principio, no sea necesario estar pendiente de la salida del demonio, pero puede servir para dar tranquilidad al usuario que todo se está ejecutando en forma correcta o que ha habido una excepción y debería hacerse algún cambio. En general, se recomienda al usuario que en caso de detectar un mensaje que describe un mal funcionamiento, se contacte directamente con el administrador para identificar correctamente el problema y solucionarlo de forma adecuada.

Algunas excepciones podrían darse por falta de conexión con la base de datos o problemas de conexión a Internet al momento de enviar e-mails con alarmas, entre otros posibles casos.

El demonio está diseñado de forma de continuar su ejecución indefinidamente y en forma autónoma a pesar de detectar excepciones. Esto es porque si bien hasta solucionar el causante de la excepción esa parte no funcionará, puede haber otros threads que sí estén en condiciones de continuar su ejecución.

Rendimiento Una vez ejecutado el demonio se analizó su rendimiento en el servidor a través de la herramienta JConsole (incluida en java) [8] y se comprobó que los recursos de memoria consumidos por los procesos

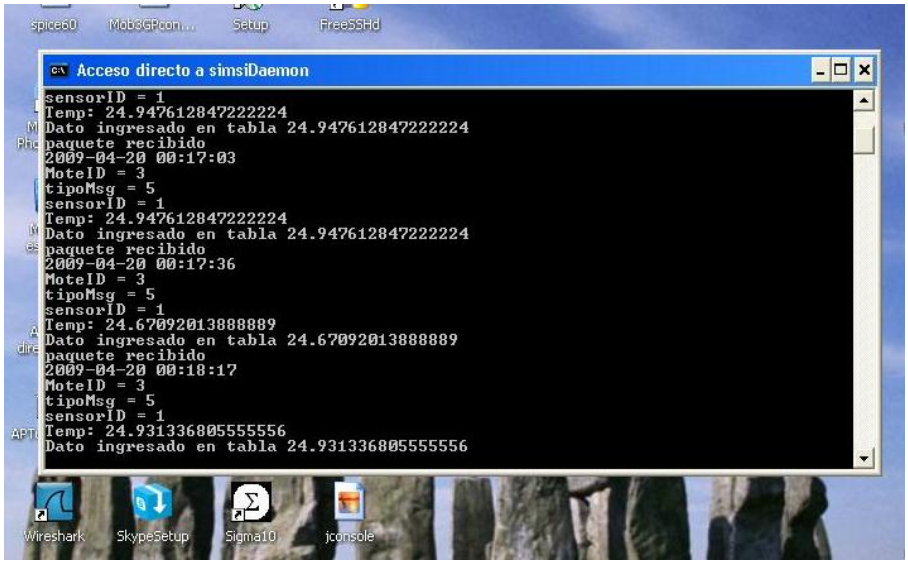

Figura 3.7: Ventana de Ejecución del Demonio

del demonio son bajos (se observaron menores a 15MB recibiendo un mensaje por minuto durante 12 horas). No se dispone de una cantidad importante de motes como para realmente testear su funcionamiento frente a un intercambio importante de mensajes, sin embargo se estima no requerirá una cantidad de memoria mucho mayor. Se muestra en la figura 3.8 una gráfica desplegada por JConsole donde se muestra lo mencionado anteriormente. Se puede ver en la figura cómo el consumo de memoria aumenta gradualmente pero al ejecutarse el Garbage Collector este consumo disminuye.

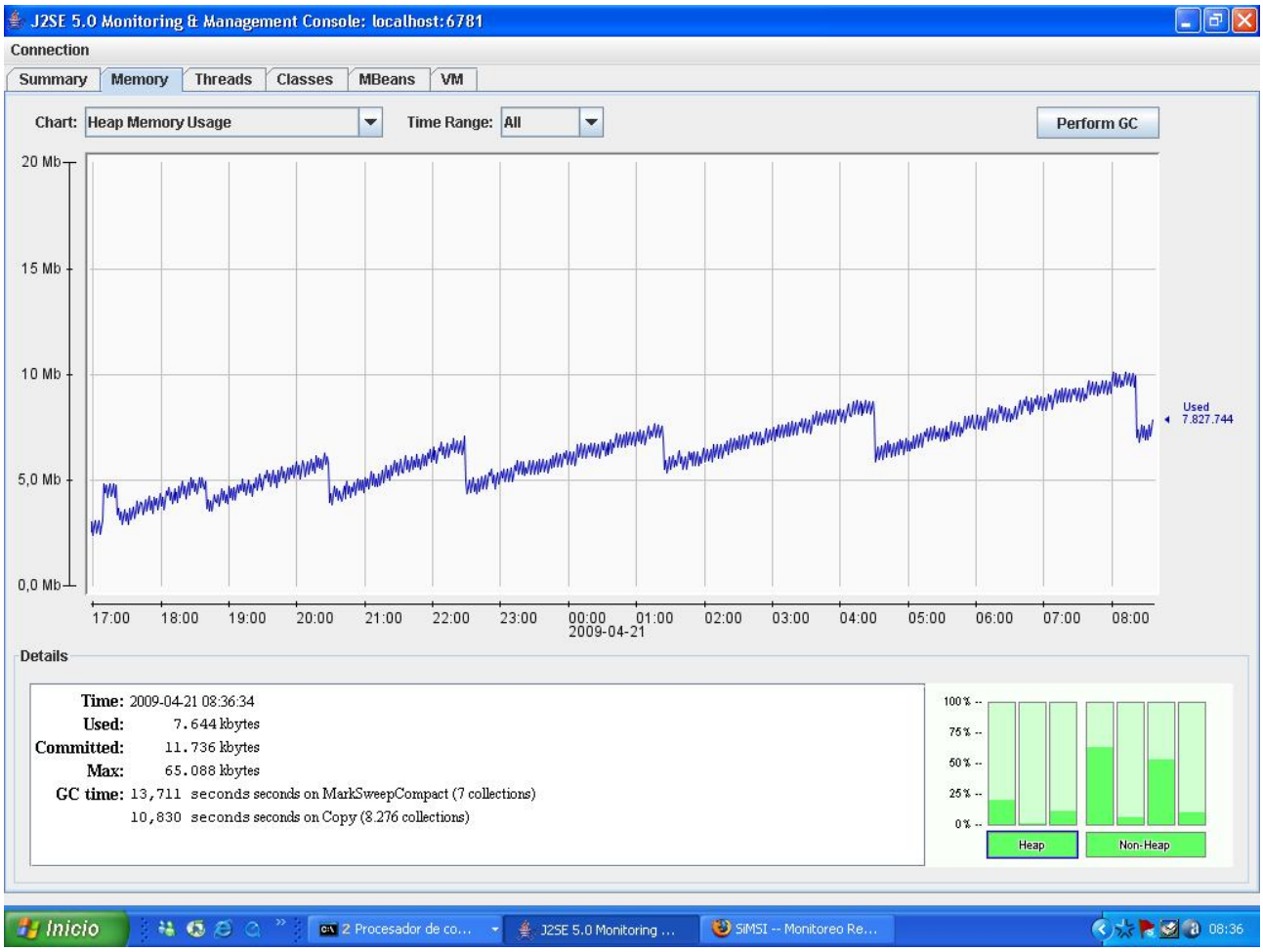

Figura 3.8: Herramienta JConsole

Capítulo 4

Interfaz de Usuario

4.1. Ingeniería de Software

4.1.1. Introducción

El diseño de la aplicación es uno de los puntos más delicados del proyecto, debido a que es la interfaz que interactúa con el usuario final.

Para el desarrollo de la ingeniería de software se pensó en brindar un acceso tanto local (en el propio lugar físico donde se encuentra la red instalada) como un acceso a través de Internet. Por ello se tuvieron en mente dos posibles desarrollos.

El primero se basaba en crear una aplicación Java en el servidor central que controle todos los aspectos de la red, y aparte una aplicación web que de un acceso más restringido pudiendo observar datos y realizar reportes.

Con estas dos aplicaciones se tendría:

1. Un acceso total en la aplicación Java del servidor pudiendo configurar routers, motes, sensores, alarmas, además de realizar otras tareas como la administración de usuarios y verificar el estado de la red.

2. Un software basado en PHP mediante el cual se podría acceder vía web a un número restringido de acciones, más contemplativas de toda la red SiMSI sin poder realizar configuraciones. Podría solamente ver reportes ya configurados y observar en un mapa los últimos datos de toda la red.

Luego de un largo debate dentro del grupo y con los tutores, se arribó a una segunda opción para desarrollar toda la aplicación. Habiendo estudiado las características y el potencial existente en las herramientas para el desarrollo de aplicaciones web, se propuso una única aplicación, basada en PHP y JavaScript, que contenga todas las funciones posibles, y que el acceso a diferentes servicios se restrinja mediante un acceso de usuarios diferenciado.

De esta manera, existe una sola aplicación corriendo, tanto web como en el servidor local, lo que facilita el desarrollo de la misma y la configuración de la red se puede realizar desde cualquier punto, sin encontrarse necesariamente en el campo.

Finalmente la segunda opción fue la elegida, lo que llevó a tener modificaciones en la ingeniería de Software. Al hacer la aplicación totalmente web, se centró el trabajo en los distintos casos de uso buscando una aplicación que sea ágil e intuitiva para el usuario. Estos casos de uso se detallan a continuación.

4.1.2. Casos de Uso

En esta sección presentamos los diferentes casos de uso para la aplicación SiMSI.

Administración de usuarios

- Actor Principal: Administrador
- Personal involucrado e intereses:

  Administrador: Crear usuarios que puedan interactuar con el sistema
- Precondiciones:

  SW instalado

  Adminstrador logueado
- Post condiciones:

  Usuarios pueden ingresar en el sistema y serán identificados

Flujo básico:

Acción del Actor	**Respuesta del Sistema**
1. Administrador ingresa en Usuarios→Nuevo Usuario	2. Despliega campos para ingresar Usuario
3. Ingresa datos y elige tipo (administrador, usuario básico o personalizado). Click en Actualizar	4. Despliega mensaje que el Usuario fue creado correctamente

Flujo Alternativo:

Acción del Actor	**Respuesta del Sistema**
1a1. Administrador ingresa en Usuarios→Usuarios existentes	1a2. Despliega tabla con todos los Usuarios
1a3. Click en Eliminar en el usuario deseado	1a4. Despliega mensaje que el Usuario fue eliminado
1b1. Administrador ingresa en Usuarios→Usuarios existentes	1b2. Despliega tabla con todos los Usuarios en panel izquierdo
1b3. Click en botón de Editar en el usuario deseado	1b4. Despliega campos para editar en panel derecho
1b5. Administrador cambia los campos que desea. Click en Actualizar	1b6. Despliega mensaje que el Usuario fue editado
1c1. Administrador ingresa en Usuarios →Usuarios existentes	1c2. Despliega tabla con todos los Usuarios en panel izquierdo
1c3. Administrador hace click en un Usuario	1c4. Despliega datos del Usuario seleccionado en panel derecho
3a1. Ingresa nombre o contraseña no válido (caracteres no válidos o password con menos de 4 caracteres)	3a2. Despliega aviso de Usuario o Password no válido

Logueo y deslogueo de Usuario al sistema

Logueo al sistema

- Actor Principal: Usuario
- Personal involucrado e intereses:

  Usuario: Acceder al sistema
- Precondiciones:

  SW instalado,

  Usuarios creados

- Post condiciones:

  El usuario está identificado por el sistema

Flujo básico:

Acción del Actor	**Respuesta del Sistema**
1. Usuario ingresa nombre, password. clic Login	2. Despliega ventana principal

Deslogueo del sistema

- Actor Principal: Usuario
- Personal involucrado e intereses:

  Usuario: Salir del sistema
- Precondiciones:

  Usuario logueado
- Post condiciones:

  El usuario sale del sistema

Flujo básico:

Acción del Actor	**Respuesta del Sistema**
1. Usuario hace clic en Logout	2. Despliega ventana de Logueo de Usuario

Configurar Router

- Actor Principal: Usuario con derecho a administrar Routers
- Personal Involucrado e Intereses:

  Usuario: Configura routers. Crea nuevos, así como los edita y los elimina
- Precondiciones:

  Router instalado y configurado.

  SW instalado y usuario logueado en el sistema.

  Usuario conoce usuario y contraseña de cada router.
- Postcondiciones:

  Tabla de routers con los datos actualizados.

Flujo Básico:

Acción del Actor	**Respuesta del Sistema**
1. Usuario ingresa en el menú Configurar → *Routers*	2. Despliega sección de Routers
3. Ingresa en Nuevo Router en panel izquierdo	4. Despliega Tabla para ingresar nuevos datos en panel derecho
5. Ingresa datos del Router y da clic en Actualizar	6. Despliega en pantalla que *El Router fue guardado correctamente*

Flujo alternativo:

Acción del Actor	**Respuesta del Sistema**
5a1. Ingresa los datos y el router_id ya existe	5a2. Despliega mensaje de router_id incorrecto
3a1. Ingresa en Routers Existentes en panel izquierdo	3a2. Despliega Routers disponibles
3a3. clic en Nombre del router seleccionado	3a4. Despliega en pantalla datos del router
3b1. Ingresa en Routers Existentes en panel izquierdo	3b2. Despliega Routers disponibles
3b3. clic en Editar en router seleccionado	3b4. Despliega en pantalla tabla para editar datos del router
3b5. edita los campos necesarios y clic en Actualizar	3b6. Despliega aviso que se guardaron los datos
3c1. Ingresa en Routers Existentes en panel izquierdo	3c2. Despliega Routers disponibles
3c3. clic en Eliminar en router seleccionado	3c4. Despliega en pantalla confirmación de Eliminar un router
3c5. Acepta la confirmación	3c6. Despliega aviso que se eliminó el router

Configurar Mote

- Actor Principal: Usuario con derecho a administrar motes

- Personal Involucrado e Intereses:

  Usuario: Configura motes. Crea nuevos, así como los edita y los elimina

- Precondiciones:

  Mote instalado y configurado.

  SW instalado y usuario logueado en el sistema.

- Postcondiciones:

  Tabla de motes con los datos actualizados.

Flujo Básico:

Acción del Actor	**Respuesta del Sistema**
1. Usuario ingresa en el menú Configurar → *Motes*	2. Despliega sección de motes
3. Usuario hace clic en mote a editar	4. Despliega Tabla para ingresar nuevos datos en panel derecho
5. Ingresa datos del mote y da clic en Actualizar	6. Despliega en pantalla que *El mote fue guardado correctamente*

Flujo alternativo:

Acción del Actor	Respuesta del Sistema
3a1. Ingresa en Estado de Motes en panel izquierdo	3a2. Despliega motes disponibles según Router al cual le envían los datos
3a3. Selecciona motes a Configurar	3a4. Despliega en pantalla motes seleccionados
3a5. Selecciona mote a Ver	3a6. Despliega en pantalla datos del mote seleccionado
3b1. Ingresa en Estado de Motes en panel izquierdo	3b2. Despliega motes disponibles según Router al cual le envían los datos
3b3. Selecciona motes a Configurar	3b4. Despliega en pantalla motes seleccionados
3b5. clic en Editar en mote seleccionado	3b6. Despliega en pantalla tabla para editar datos del mote
3b7. Edita los campos necesarios y clic en Actualizar	3b8. Despliega aviso que se guardaron los datos
3c1. Ingresa en Estado de Motes en panel izquierdo	3c2. Despliega motes disponibles según Router al cual le envían los datos
3c3. Selecciona motes a Configurar	3c4. Despliega en pantalla motes seleccionados
3c5. clic en Eliminar en mote seleccionado	3c6. Despliega en pantalla confirmación de Eliminar un mote
3c7. Acepta la confirmación	3c8. Despliega aviso que se eliminó el mote

Configurar Sensores

- Actor Principal: Usuario con derecho a administrar Sensores

- Personal Involucrado e Intereses:

  Usuario: Configura sensores de los motes. Crea nuevos tipos, así como los edita y los elimina.

- Precondiciones:

  SW instalado y usuario logueado en el sistema.

- Postcondiciones:

  Tabla de sensores con los datos actualizados.

Flujo Básico:

Acción del Actor	Respuesta del Sistema
1. Usuario ingresa en el menú Configurar → *Sensores*	2. Despliega sección de Sensores
3. Ingresa en Nuevo Sensor en panel izquierdo	4. Despliega Tabla para ingresar nuevos datos en panel derecho
5. Ingresa datos del Sensor y da clic en Actualizar	6. Despliega en pantalla que *El Sensor fue guardado correctamente*

Flujo alternativo:

Acción del Actor	Respuesta del Sistema
5a1. Ingresa los datos y el nombre ya existe	5a2. Despliega mensaje de nombre incorrecto
5b1. No existe un tipo de dato para el sensor, entonces hace clic en *Ingresar nuevo tipo de dato*	5b2. Despliega pop up con campos para ingresar nuevo tipo de datos
5b3. Ingresa nombre y unidad del nuevo tipo de dato	5b4. Despliega mensaje que *El Tipo de Dato fue guardado correctamente*
3a1. Ingresa en Sensores Existentes en panel izquierdo	3a2. Despliega Sensores disponibles
3a3. clic en Nombre del sensor seleccionado	3a4. Despliega en pantalla datos del sensor
3b1. Ingresa en Sensores Existentes en panel izquierdo	3b2. Despliega Sensores disponibles
3b3. clic en Editar en sensor seleccionado	3b4. Despliega en pantalla tabla para editar datos del sensor
3b5. Edita los campos necesarios y clic en Actualizar	3b6. Despliega aviso que se guardaron los datos
3c1. Ingresa en Sensores Existentes en panel izquierdo	3c2. Despliega Sensores disponibles
3c3. clic en Eliminar en sensor seleccionado	3c4. Despliega en pantalla confirmación de Eliminar un sensor
3c5. Acepta la confirmación	3c6. Despliega aviso que se eliminó el sensor

Configurar Frecuencia

- Actor Principal: Usuario con derecho a configurar frecuencias
- Personal Involucrado e Intereses:

  Usuario: Configurar frecuencia del envío de datos por parte de los sensores de los motes.
- Precondiciones:

  Mote instalado y configurado.

  SW instalado y usuario logueado en el sistema.
- Postcondiciones:

  Se envía mensaje a los motes seleccionados para el cambio de frecuencia

  Tabla de motes y sensores con la nueva frecuencia guardada y esperando confirmación de los motes.

Flujo Básico:

Acción del Actor	Respuesta del Sistema
1. Usuario ingresa en el menú Configurar → *Frecuencia*	2. Despliega sección de frecuencia
3. Ingresa en sección Nueva Frecuencia del panel izquierdo	4. Despliega Routers a seleccionar y campo para modificar frecuencia
5. Selecciona los routers para enviar la nueva frecuencia (donde se encuentran los motes a configurar su frecuencia) y nueva frecuencia a setear. clic en Actualizar	6. Despliega aviso que se seteó la nueva frecuencia y quedar a la espera de la confirmación de los motes.

Flujo alternativo:

Acción del Actor	Respuesta del Sistema
3a1. Ingresa en sección Distribución de Frecuencias en panel izquierdo	3a2. Despliega tabla con Routers, frecuencia deseada de sus motes, y estado de confirmación de la nueva frecuencia

Configurar Alarmas

- Actor Principal: Usuario con derecho a administrar Alarmas
- Personal Involucrado e Intereses:

  Usuario: Configura Alarmas para la llegada de datos y el estado de los equipos. Crea nuevas alarmas, así como las edita y las elimina
- Precondiciones:

  Routers instalados y sensores configurados.

  SW instalado y usuario logueado en el sistema.
- Postcondiciones:

  Tablas de alarmas con los datos actualizados para que el demonio verifique los datos recibidos.

Flujo Básico:

Acción del Actor	**Respuesta del Sistema**
1. Usuario ingresa en el menú Configurar → *Alarmas*	2. Despliega sección de Alarmas
3. Ingresa en Nueva Alarma en panel izquierdo	4. Despliega Tabla para ingresar nuevos datos en panel derecho
5. Ingresa datos de la Alarma, elige las zonas y los usuarios a notificar. Da clic en Actualizar	6. Despliega en pantalla que *La Alarma fue guardada correctamente*

Flujo alternativo:

Acción del Actor	**Respuesta del Sistema**
5a1. Ingresa los datos y el Nombre de la Alarma ya existe	5a2. Despliega mensaje de Nombre incorrecto
5b1. Ingresa una zona que ya tiene asignada una Alarma con el mismo Tipo de Sensor	5b2. Despliega mensaje de que no ingresa esa Zona a la Alarma. Guarda los demás datos de la Alarma y despliega aviso
3a1. Ingresa en Alarmas Existentes en panel izquierdo	3a2. Despliega Alarmas disponibles
3a3. clic en Nombre de la Alarma seleccionada	3a4. Despliega en pantalla datos, zonas y usuarios notificados de la Alarma
3b1. Ingresa en Alarmas Existentes en panel izquierdo	3b2. Despliega Alarmas disponibles
3b3. clic en Editar en alarma seleccionada	3b4. Despliega en pantalla tabla para editar datos de la alarma
3b5. Edita los campos necesarios y clic en Actualizar	3b6. Despliega aviso que se guardaron los datos
3c1. Ingresa en Alarmas Existentes en panel izquierdo	3c2. Despliega Alarmas disponibles
3c3. clic en Eliminar en alarma seleccionada	3c4. Despliega en pantalla confirmación de Eliminar una alarma
3c5. Acepta la confirmación	3c6. Despliega aviso que se eliminó la alarma

Ver Mapa

- Actor Principal: Usuario
- Personal Involucrado e Intereses:

  Usuario: Ver la distribución física de los diferentes equipos de la red en un mapa del campo. Acceder a últimos datos de sensores, routers y motes.

- Precondiciones:

  Motes y Routers instalados con sus coordenadas.

  SW instalado y usuario logueado en el sistema.

  Conexión a Internet (*GoogleMaps*)

- Postcondiciones:

  El usuario obtiene una imagen de cómo están distribuidos los motes y routers en el terreno y el status de cada uno.

Flujo Básico:

Acción del Actor	**Respuesta del Sistema**
1. Usuario ingresa en el menú Mapa	2. Despliega mapa con motes y routers desplegados. En el panel izquierdo muestra la lista de todos los motes y routers en el mapa.
3. El usuario hace clic en un equipo en la lista en el panel izquierdo o en el equipo sobre el mapa	Se despliega información del equipo seleccionado con sus datos más recientes y un link para acceder a la configuración el equipo.

Crear y Visualizar Reportes

- Actor Principal: Usuario

- Personal Involucrado e Intereses:

  Usuario: Tener un informe sobre los datos recabados por los sensores

- Precondiciones:

  Motes instalado y configurado.

  SW instalado y usuario logueado en el sistema.

  Base de datos recibiendo datos de los motes.

- Postcondiciones:

  Reportes realizados y visualizados por el usuario.

  Configuración de Reporte almacenada en Reportes existentes

Flujo Básico:

Acción del Actor	**Respuesta del Sistema**
1. Usuario ingresa en el menú Reportes	2. Despliega sección de Reportes
3. Ingresa en Nuevo Reporte en panel izquierdo	4. Despliega campos para crear reporte como nombre del reporte, motes para seleccionar, tipo de sensor, período de toma de datos, tipo de reporte (evolución temporal de motes, de routers o evolución temporal de una hora)
5. Ingresa datos del reporte y da clic en Actualizar	6. Despliega en pantalla que *El reporte fue guardado correctamente*
7. Usuario hace clic en Reportes existentes en el panel izquierdo	8. Despliega lista de Reportes existentes
9. Usuario hace clic en el nombre del reporte guardado	10. Despliega la gráfica del reporte seleccionado

Flujo alternativo:

Acción del Actor	Respuesta del Sistema
5a1. Ingresa los datos y el nombre del reporte ya existe	5a2. Despliega aviso de que el reporte no se pudo guardar
3a1. Ingresa en Reportes Existentes en panel izquierdo	3a2. Despliega lista de Reportes existentes
3a3. Usuario hace clic en el nombre del reporte guardado	3a4. Despliega la gráfica del reporte seleccionado
3b1. Ingresa en Reportes Existentes en panel izquierdo	3b2. Despliega lista de Reportes existentes
3b3. clic en Editar en reporte seleccionado	3b4. Despliega en pantalla campos para editar datos del reporte
3b5. Edita los campos necesarios y clic en Actualizar	3b6. Despliega aviso que se guardaron los datos
3c1. Ingresa en Reportes Existentes en panel izquierdo	3c2. Despliega lista de Reportes existentes
3c3. clic en Eliminar en reporte seleccionado	3c4. Despliega en pantalla confirmación de Eliminar un mote
3c5. Acepta la confirmación	3c6. Despliega aviso que se eliminó el mote

Ver Datos Puntuales

- Actor Principal: Usuario
- Personal Involucrado e Intereses:

  Usuario: Tener un informe sobre los datos en un período de tiempo personalizado
- Precondiciones:

  Motes instalado y configurado.

  SW instalado y usuario logueado en el sistema.

  Base de datos recibiendo datos de los motes.
- Postcondiciones:

  Reporte visualizado por el usuario.

Flujo Básico:

Acción del Actor	Respuesta del Sistema
1. Usuario ingresa en el menú Reportes	2. Despliega sección de Reportes
3. Ingresa en Ver Datos Puntuales en panel izquierdo	4. Despliega campos para crear reporte como motes para seleccionar, tipo de sensor, período de toma de datos, tipo de reporte (evolución temporal de motes, de routers o evolución temporal de una hora)
5. Ingresa datos del reporte y da clic en Actualizar	6. Despliega la gráfica del reporte seleccionado

Ver Logs

- Actor Principal: Usuario con derechos a visualizar los logs
- Personal Involucrado e Intereses:

  Usuario: Observar últimas acciones de los usuarios, configuraciones de equipos frecuencias, o alarmas.
- Precondiciones:

  Logs creados.

  SW instalado y usuario logueado en el sistema.

- Postcondiciones:
  El usuario observa los logs.

Flujo Básico:

Acción del Actor	**Respuesta del Sistema**
1. Usuario ingresa en el menú Logs	2. Despliega sección de Logs. En el panel izquierdo muestra los diferentes logs como son Acciones de usuarios, Configuración de Equipos y Alarmas.
3. El usuario hace clic en el log que desea ver	Se despliega el log deseado.

4.2. Implementación y Diseño del Software

Al momento del desarrollo de la aplicación Web se tuvo especial cuidado en que ésta sea ágil e intuitiva. Por esto mismo se eligió un diseño similar al de las páginas webs más populares de hoy en día de forma que el usuario se sienta familiarizado con el entorno.

Se decidió utilizar PHP, JavaScript y CSS como bases de nuestra aplicación ya que son lenguajes de programación muy rápidos, potentes y populares.

Como se menciona en la sección 3.2.2, PHP es un lenguaje de programación interpretado (es decir que no requiere compilación) diseñado especialmente para la creación de páginas Web dinámicas. Generalmente, y como es utilizado en SiMSI, se ejecuta en un servidor Web, tomando el código PHP como entrada y creando páginas Web como salida [3].
Este lenguaje permite entre otras cosas interactuar con base de datos y la utilización de variables dentro del código HTML.

Otro de los lenguajes utilizados sin necesidad de compilación es JavaScript. La mayor diferencia con el anterior es que éste es ejecutado del lado del cliente al mismo tiempo que las sentencias se descargan junto con el código HTML [10]. El mismo permitió ejecutar funciones en la página Web en base a acciones del usuario. Esto quiere decir, ejecutar sentencias al momento de hacer un clic, de seleccionar un objeto, de cerrar una ventana, etc.

Junto con propiedades de CSS es posible desplegar y ocultar secciones logrando efectos importantes para el uso intuitivo de la página.

CSS (Cascading style sheets) es un lenguaje utilizado para definir la presentación de un documento estructurado escrito en HTML o XML. El objetivo detrás del desarrollo de CSS es separar la estructura de un documento de su presentación o estilo [9]. El estilo puede ser descripto tanto en el código HTML como en una hoja estilos adjunta. En el último caso, se debe etiquetar un cierto bloque HTML de manera de ser identificado en la hoja de estilos. Este identificador puede no ser único obteniendo varios bloques HTML relacionados a las mismas propiedades de diseño.
En SiMSI se utilizó una misma hoja de estilo para todas las páginas llamada SiMSI.css donde se definió el diseño de grandes bloques como bordes, imágenes, menúes, cuadros, etc.
Esto permite modificar simple y rápidamente la visualización de la aplicación. En especial, con la herramienta Firebug de Firefox es posible realizar estas modificaciones en tiempo real lo que facilita considerablemente el diseño.

4.2.1. Estructura de la Página

Basados en la Ingeniería de Software y con las herramientas recién mencionadas, se prosiguió con el diseño e implementación de la interfaz de usuario.

La estructura de la página es la mostrada en la figura 4.1, donde un usuario una vez logueado accede a la página de Inicio de la misma. En ésta se crean accesos directos a los favoritos del usuario minimizando, de esta manera, la cantidad de clics que se deberán realizar para acceder a las funciones deseadas.

Figura 4.1: Diseño de la Página de Inicio

En el menú horizontal se muestran las 5 grandes secciones en las que está dividida la aplicación, Configurar, Usuarios, Mapa, Reportes y Logs.

Este menú puede variar dependiendo los privilegios de cada usuario, mostrando solamente las secciones donde se tienen permisos.

También, como en todas las páginas, se puede observar en la esquina inferior izquierda el estado de las alarmas o alerta existentes de forma dinámica.

Configurar

En la pestaña Configurar se podrán crear y editar nuevos routers, motes, frecuencias de muestreo de motes, alarmas y sensores.

Routers Dentro de la sección Routers, entre otras cosas, se puede acceder a la configuración Web del router seleccionado a travéz de un puerto previamente establecido por el usuario.

Esto permite configurar los routers desde cualquier lado evitando así ir hasta el campo para realizar los cambios. Por otro lado, se colocan recomendaciones y sugerencias en las páginas de configuración predefinidas ayudando así al usuario a realizar las modificaciones necesarias en la configuración de los routers.

Se muestra en la figura 4.2 la página donde se configuran los puertos a redirigir en los routers.

Figura 4.2: Configuración de los puertos del router

Frecuencia A modo de ejemplo, se muestra en la figura 4.3 el formato de *Configurar Frecuencia*, otra de las importantes acciones a realizar en esta sección.

Alarmas El manejo de las Alarmas también se encuentra dentro de la sección configurar.
En la misma el usuario puede configurar las alarmas de la red como se vio en la sección 4.1.2.

Al crear una nueva Alarma se seleccionan los valores de la misma, así como las zonas donde se va a habilitar y los usuarios a notificar. Se utilizan los routers y sus clústers de motes como las diferentes zonas posibles.

En cuanto a los usuarios, existe la opción de enviarles una notificación por e-mail o por SMS, según sus datos ingresados en el sistema.

Una opción importante que existe al crear o editar una alarma es el hecho de poder habilitarla y deshabilitarla.

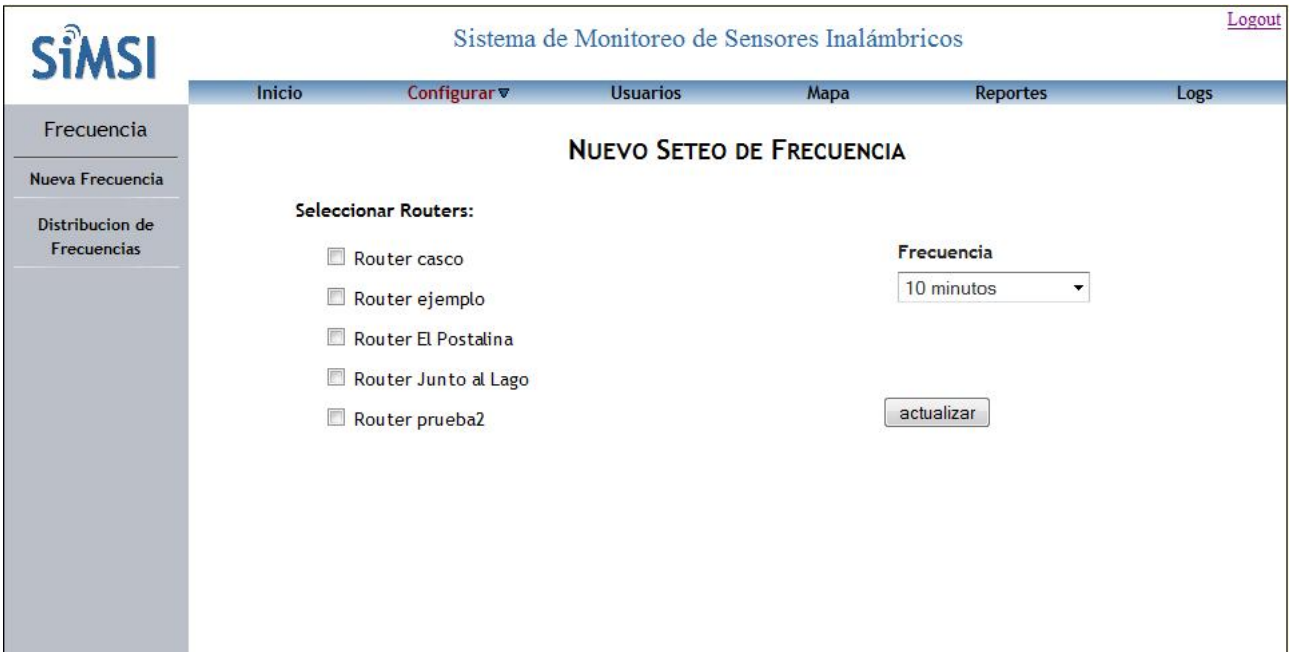

Figura 4.3: Diseño de la Sección Configurar Frecuencia

Al deshabilitar una alarma, se pierden las activaciones de la misma y el demonio no la va a considerar al momento de verificar las alarmas.

Como este es un punto muy delicado que puede llevar a errores, se pide una confirmación al momento de guardar cambios en la alarma si la misma se encuentra deshabilitada (figura 4.4).

Existen también alarmas preestablecidas que el usuario puede modificar mínimamente. Éstas son Estado Enlace, Sensor Id y Batería, vistas en detalle en la sección 3.2.5, donde el usuario puede solamente modificar los usuarios a ser notificados en caso de activación.

Son alarmas propias de la red que deben encontrarse siempre activadas para todos los routers.

Usuarios

En la sección Usuarios se podrán crear y editar nuevos usuarios así como también definir los permisos de cada uno.

En un principio se tiene pensado determinar 3 tipos de usuarios: Administrador, básico y personalizado. El Administrador tendrá todos los permisos, entre ellos setear las acciones permitidas de cada usuario.

El usuario básico es un usuario predefinido con potestades limitadas mientras que en el personalizado el administrador puede definir los privilegios.

Esta sección se diseñó como en la figura 4.5.

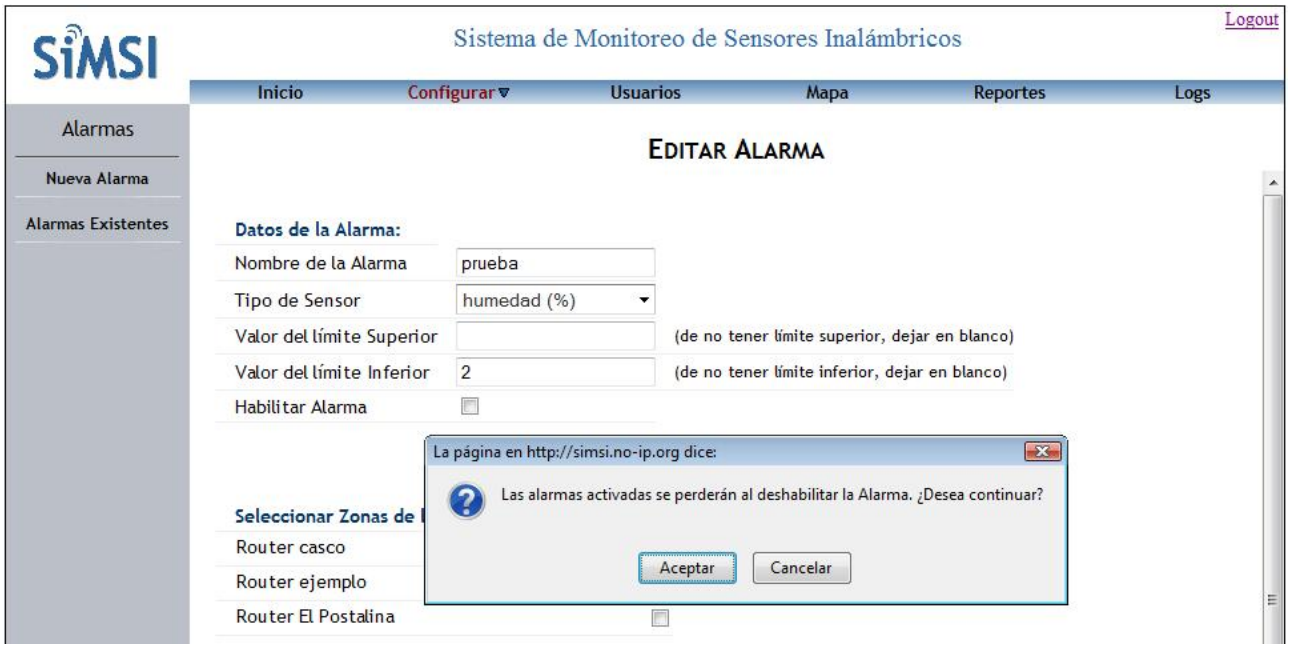

Figura 4.4: Mensaje de confirmación para alarmas deshabilitadas

Mapa

Uno de los atractivos de la aplicación es el obtener datos de la red lo más recientes posibles, pudiendo observar el estado de la red global casi en tiempo real.

Para mostrar toda la red, se utiliza un mapa de todos los routers y motes distribuidos en el campo.

Para la implementación del mismo, se pensó en un principio darle la posibilidad al usuario de cargar un mapa manualmente (un archivo de tipo .jpg, por ejemplo) y luego referenciar las coordenadas de los equipos a ese mapa.

Esta idea fue descartada rápidamente debido a la complicación innecesaria de obtener una fotografía aérea del campo, cuando ya se encuentra disponible una herramienta global que perimte obtener imágenes satelitales: la aplicación diseñada por Google llamada Google Maps.

Con Google Maps se obtienen mapas satelitales de cualquier punto del planeta con sólo designar las coordenadas de un punto. Esto permite una flexibilidad enorme al mapa, ya que si se cambian los equipos de posición, modificando correctamente las coordenadas de los mismos en nuestra aplicación Web ya tenemos el nuevo mapa de la red.

Google Maps Google Maps es una tecnología y una aplicación de mapeo web desarrollada por Google que fue lanzada al mercado en el año 2005. Con ella, el usuario puede ubicar cualquier punto del planeta gracias a sus mapas satelitales. Presenta diversas funcionalidades básicas como el acercamiento, alejamiento o desplazamiento de la imagen vista en el mapa [11].

Provee además mapas de calles y rutas de varios países, y la posibilidad de agregar posiciones en el mapa con sus coordenadas, marcándolas con imágenes tipo PNG.

Lo interesante de esta aplicación para este proyecto es su posibilidad de embeber un mapa Google Map

Figura 4.5: Diseño de la Sección Usuarios

dentro de un sitio web personal. Para ello, el grupo Google creó la división Google Maps API [12].

Gracias a una clave otorgada por Google Maps, un administrador de un sitio puede obtener el permiso para disponer de un mapa con todas sus funcionalidades dentro de un dominio en particular.

Es importante tener en cuenta este punto para la implementación de la aplicación ya que para sitios de acceso público esta clave es gratuita, pero para aplicaciones comerciales y/o restringidas, la clave debe ser comprada a la empresa Google. Se diseñó el Google Maps Premier API para estos casos, que a su vez de otorgar una clave para la utilización de los mapas, brinda servicios adicionales como opciones de asistencia y servicio por parte de Google y un control sobre los anuncios en los mapas.

Hasta ahora Google Maps API no presenta anunciantes, pero es una posibilidad de que en un futuro se encuentren en la aplicación lo que no es para nada deseable en una aplicación orientada a ser comerciable, como lo es la nuestra [13].

Por tratarse de una aplicación académica en estos momentos, se decidió utilizar el Google Maps API, obteniendo una clave de manera gratuita para su libre utilización dentro de nuestro sitio web.

Implementación La implementación de la sección Mapa en la aplicación Web requiere un diseño utilizando varios lenguajes de programación.

Por un lado, al igual que el resto de las secciones, utiliza como lenguaje principal PHP para la comunicación con la Base de Datos y, principalmente, para continuar con los lineamientos principales de la aplicación. De esta manera la misma es homogénea en su conjunto, un punto muy importante a la hora de pensar en la accesibilidad para el usuario.

Google Maps API se basa principalmente en dos lenguajes para su diseño: JavaScript y XML.

Con Javascript se controlan todas las funciones del mapa al momento de seleccionar la zona del planeta a

visualizar, así como todo lo necesario para agregar los equipos.

Los datos necesarios se toman de un documento XML creado gracias al archivo *xmlSiMSI.php*.

Este archivo se encarga de hacer las consultas a la Base de Datos, y de esta manera se obtiene el documento XML con las coordenadas de los equipos, los nombres de los equipos, las alarmas que tienen activadas y demás datos.

Para los equipos que son Routers, realiza las consultas para obtener su router_id, su dirección IP, la disponibilidad del mismo, sus coordenadas y si tiene activada la alarma de Estado Enlace.

En el caso de los motes, se obtienen el mote_id, la IP del router al cual se encuentra asociado, sus coordenadas, las alarmas que tiene activadas, el id y las coordenadas del mote padre (si lo tiene) y finalmente los últimos datos obtenidos por sus sensores.

Luego, el archivo *mapa.php* es el archivo que se abre a la hora de ingresar a la sección Mapa.

Éste, en su sección *head* de HTML tiene el código de Javascript para crear el Google Map con todos los dispositivos y procesar los datos del documento XML.

Aquí es donde se inserta la clave otorgada por Google para la utilización de Google Maps API y donde se crean los íconos de los elementos a aparecer en el mapa.

En este caso se distinguen cuatro íconos que corresponden a motes y routers que tienen y no tienen alarmas activas.

Si el equipo tiene valores no nulos en su latitud y longitud se crea un ícono y se marca en el mapa.

Para cada ícono se crea una etiqueta que se despliega al momento de hacer clic con los datos más recientes y con un link para ver la configuración del dispositivo.

Se puede observar esta etiqueta en la figura 4.6.

Figura 4.6: Ejemplo de etiqueta de los equipos del Mapa

Para los motes con mote padre que se encuentran en el mapa, se traza su relación con una línea entre los mismos.

Finalmente también se crean dos "sidebars" autodesplegables con los Routers y Motes presentes en el Mapa mostrando si tienen o no alarmas activas.

La figura 4.7 muestra toda la sección Mapa.

Figura 4.7: Diseño de la Sección Mapa

Reportes

La sección Reportes es una de las más importantes ya que es donde se podrán crear, editar y visualizar los reportes existentes. Para generar un reporte, se deben seleccionar los motes a monitorear, el tipo de sensor, el período de monitoreo y el tipo de gráfico.

Los motes se seleccionan a través de un árbol desplegable que los agrupa según el router al cual están asociados en el momento de generar el reporte.

El tipo de sensor se selecciona a partir de una lista desplegable con todos los tipos de datos posibles, previamente configurados por el usuario.

El período de monitoreo se puede seleccionar mediante un desplegable con las siguientes opciones:

- Últimas 24 horas
- Última semana
- Último mes

- Últimos 3 meses
- Últimos 6 meses
- Último año
- Mes Corriente

Estas opciones posibilitan que al momento de visualizar el reporte se obtenga la información correspondiente al período de la opción en cuestión. En la última, los datos desplegados corresponden únicamente a las medidas obtenidas en el mes en el que se visualiza el reporte.

A grandes rasgos existen 2 tipos de gráfico posibles:

- *Evolución temporal*: En esta gráfica se muestra un promedio de los datos de los sensores seleccionados diferenciados por routers o por mote. El promedio se realiza dependiendo el período de tiempo seleccionado. A modo de ejemplo, para las últimas 24hs se promedia cada hora mientras que para el último mes cada 6 horas.
- *Comparación de promedios por horas*: En este reporte se muestra la variación del tipo de dato seleccionado, a la misma hora durante distintos días. De esta manera un productor podrá evaluar, por ejemplo, si su sistema de riego es constante durante cierto período de tiempo o si hay algún momento crítico donde haya que tener extremo cuidado, etc.

En la figura 4.8 se muestra la página de configuración de un nuevo reporte.

Una vez creado un reporte, en el menú de la izquierda aparecerá bajo la sección *Reportes Existentes*. Al hacer clic en el nombre del reporte se despliega una gráfica como la mostrada en la figura 4.9.

La última opción del menú vertical, Ver Datos puntuales, permite al usuario obtener las medidas comprendidas entre fechas y horas específicas, sin generar un reporte. Esto se aprecia en la figura 4.10.

Haciendo clic en el botón actualizar, se visualiza la gráfica correspondiente a las opciones seleccionadas, como se ve en la figura 4.11.

Cada vez que se genera un gráfico, se pueden también exportar los datos graficados en un archivo Excel. Haciendo clic en *Archivo Excel Generado*, se tendrá la opción de descargar el archivo generado.

Logs

La siguiente, y última sección está destinada a los logs.

En ésta se muestran 3 tipos de acontecimientos:

- *Acciones de Usuarios*: Todo lo relativo a lo que hace un usuario en la página como mirar reportes, borrar routers, cambiar una frecuencia, etc.
- *Configuración de equipos*: Se registran todos los cambios de configuración de routers y motes.
- *Alarmas*: Se guardan todos los registros relacionados a la activación y desactivación de alarmas. Cuando la misma es de Estado de Enlace, hace mención al router que tiene problemas con su enlace y cuando es de tipo Sensor Id, nos detalla el sensor que se necesita configurar. De otro modo, siendo la alarma de Batería o una configurada por los usuarios, se muestra el mote que la está generando o desactivando.

  En la figura 4.12 se puede observar la estructura de esta sección.

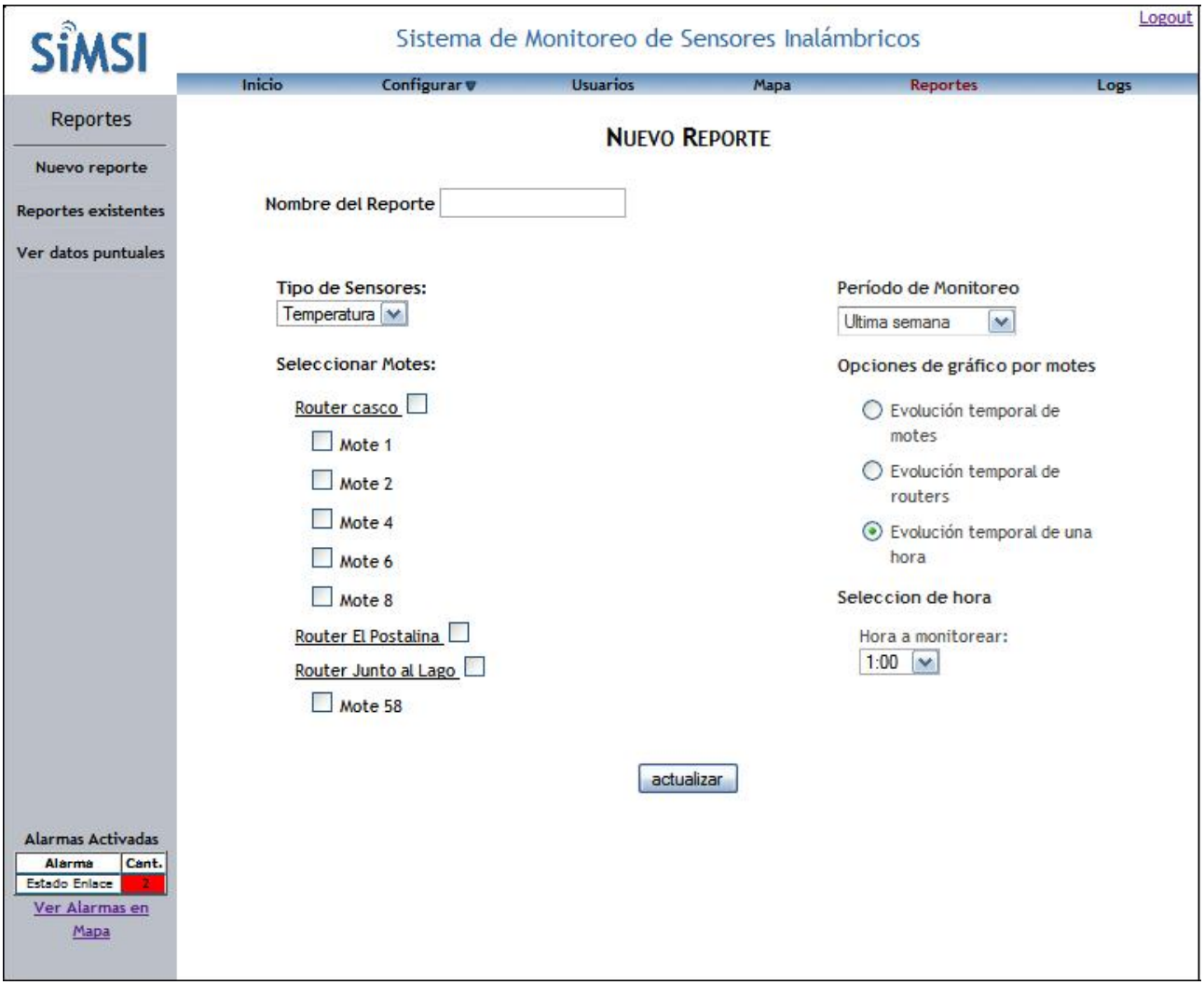

Figura 4.8: Diseño de la Sección Nuevo Reporte

Figura 4.9: Ejemplo de una Gráfica de la Sección Reportes

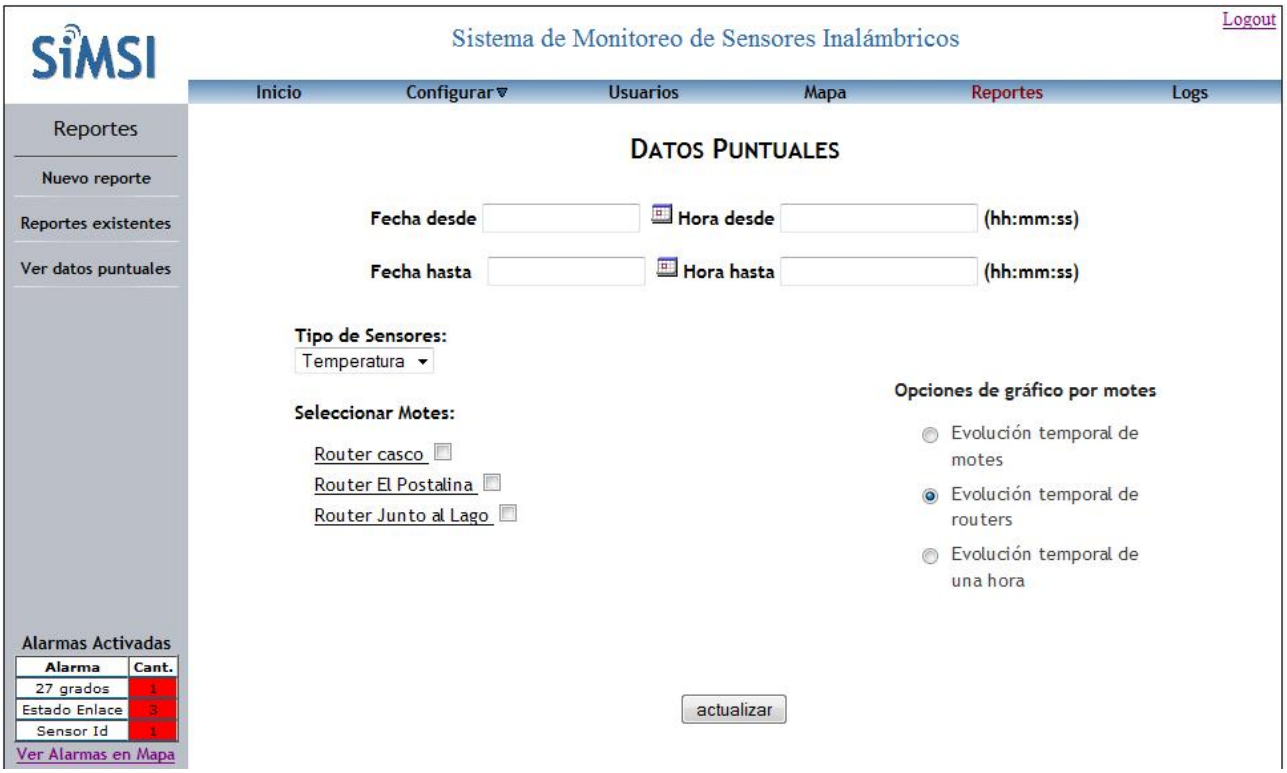

Figura 4.10: Ver Datos Puntuales

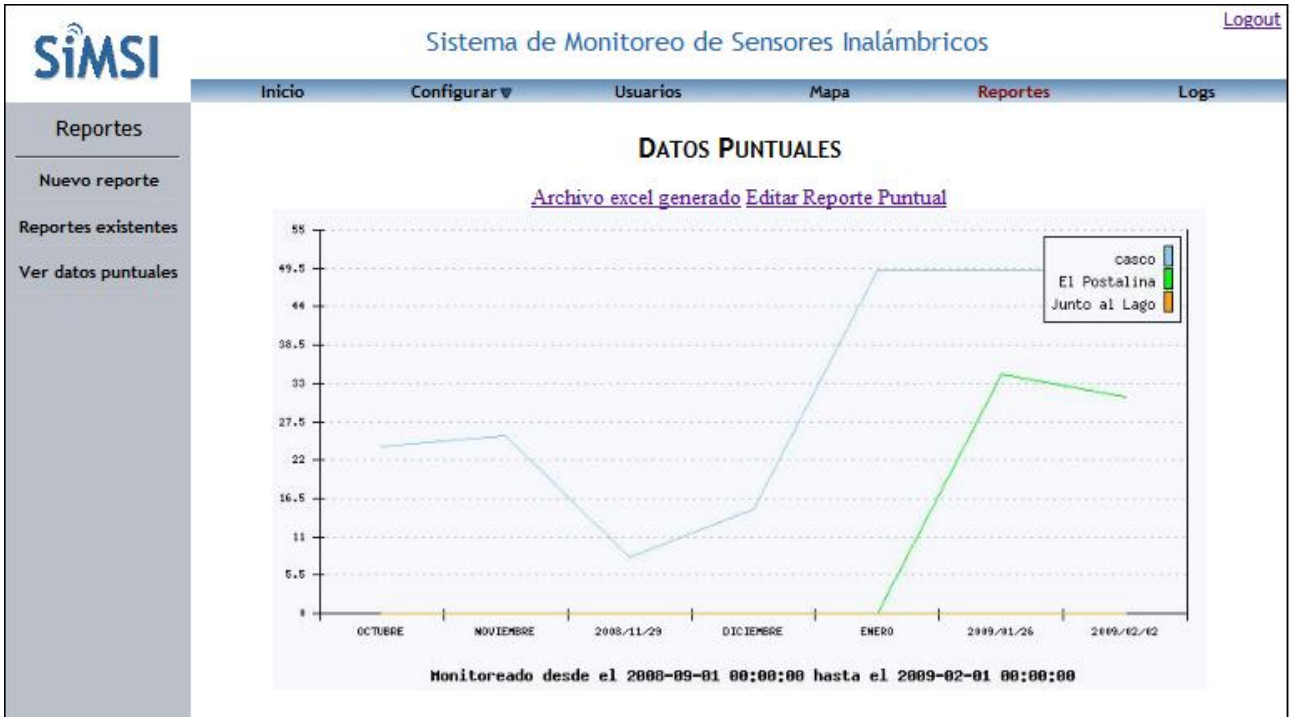

Figura 4.11: Ejemplo de una Gráfica de Datos Puntuales

Figura 4.12: Diseño de la Sección Logs

4.3. Instalación del Software

A pesar de que la aplicación es totalmente web, un requerimiento para el proyecto es el de poder instalar la aplicación en el servidor con un archivo autoextraíble, un instalador.

Ya se observó en la sección 3.2.2 que el servidor se basa en un servidor WAMP (Windows, Apache, MySQL y PHP). Existe una aplicación WAMP que conjuga estos 4 actores que puede ser instalada fácilmente y además es Open Source, lo que brinda una libertad muy importante a la hora de diseñar la aplicación.

Para ello se decidió utilizar la versión *WampServer Version 2.0c* que es la versión más reciente actualmente.

Este servidor WAMP provee de versiones de PHP, MySQL y Apache que pueden ser actualizadas por el usuario cuando nuevas versiones sean liberadas. El único problema a tener en cuenta aquí es que algunas versiones de PHP no son compatibles con otras de Apache. En ese caso, la aplicación WAMP puede avisar si son o no compatibles asegurando su perfecto funcionamiento [15].

Actualmente, las versiones preconfiguradas presentes en el *WampServer Version 2.0c* son las siguientes:

- Apache version 2.2.8
- MySQL version 5.0.51b
- PHP version 5.2.6
- Phpmyadmin version 2.11.6
- SQLiteManager version 2.8.17

Se ve en la lista anterior la presencia de Phpmyadmin y SQLiteManager, dos aplicaciones que permiten la interacción con la base de datos.

SQLiteManager implementa la gestión de la Base de Datos basada en SQLite [14]. Asimismo, SQLite es un motor para la Base de Datos en SQL.

En tanto Phpmyadmin es una aplicación basada en PHP que provee la administración de MySQL a través de la web [16].

Por todo lo expresado anteriormente, el instalable de *WampServer Version 2.0c* se debe encontrar en el paquete de instalación.

Es necesario, con el WAMP ya instalado en el servidor, sobreescribir y cargar nuevos archivos en la configuración del WAMP para poder instalar las funcionalidades de la aplicación.

Asimismo, todos los archivos php necesarios para la página web, así como las tablas de la base de datos son propios del proyecto por lo que se integran al WAMP por primera vez con el instalador.

El otro elemento que forma parte de la instalación es el demonio. Éste es una aplicación basada en el lenguaje Java. Por lo tanto, un requerimiento del Servidor es que cuente con un JRE (Java Runtime Environment) para poder correr dicha aplicación. Se debe entonces instalar en el servidor el demonio con todo su desarrollo en Java.

Como las aplicaciones utilizadas son el servidor WAMP y JRE, los requerimientos del PC para el proyecto son los de estas aplicaciones.

Para el JRE, utilizando su última versión disponible (en este caso es la versión 6.0), sus requerimientos mínimos son:

- Procesador Pentium 166 MHz o superior
- 64 MB de Memoria RAM
- Windows XP o posterior
- Mozilla 1.4 o posterior, o IE 5.5 o posterior

[17]

Los requerimientos para la instalación del Servidor WAMP son:

- Windows XP o posterior
- Procesador Pentium IV o superior
- 128MB de Memoria RAM

Independientemente de los requerimientos particulares de cada componente, para un correcto funcionamiento sugerimos como mínimo instalar SiMSI en un equipo con las siguientes características:

- Windows XP
- Procesador Pentium IV
- 512MB de Memoria RAM
- 10 GB de espacio libre en disco duro

En cuanto al navegador web, la aplicación se diseñó para su utilización en Mozilla Firefox 1.4 o posterior. Se trata de un navegador libre y de código abierto que deberá ser descargado e instalado por el usuario en caso de no disponer de él aún [18].

Con todos los archivos referidos al WAMP y el demonio podemos hacer un archivo instalador autoextraíble.

La creación del instalador se puede lograr gracias a un programa llamado *Create Install*.

Esta aplicación permite crear un instalador (un archivo .exe) con todas las funcionalidades básicas como ser la extracción de los archivos, la creación de accesos directos y de un desinstalador, así como mostrar durante el proceso de instalación las diferentes etapas de la misma.

Se puede observar el instalador en funcionamiento en la figura 4.13.

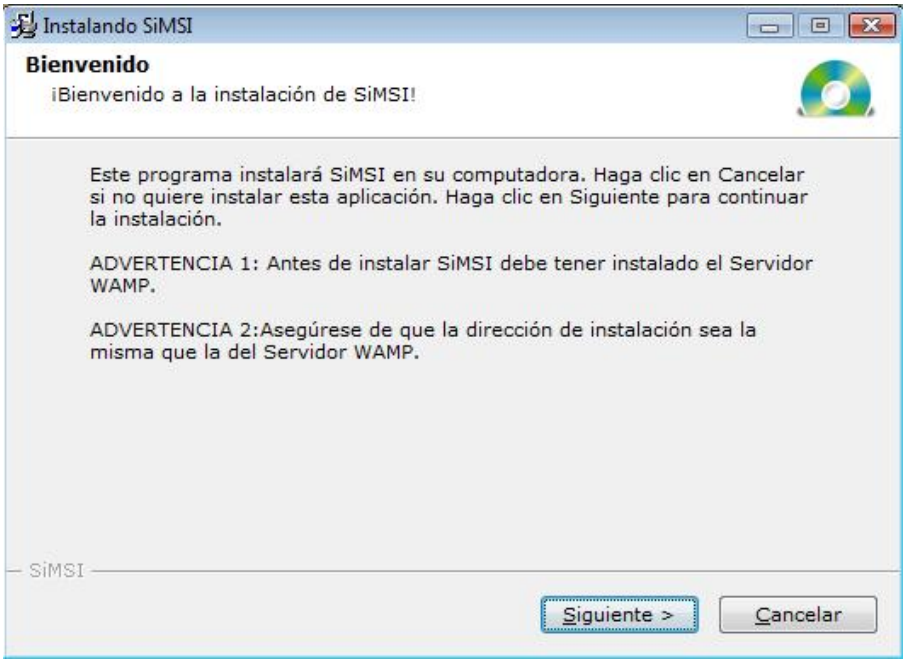

Figura 4.13: Instalador de SiMSI

En cuanto al demonio, es necesario que comience a correr una vez encendido el servidor, por lo tanto se creó un acceso directo del mismo que se instale en la carpeta INICIO. En esta carpeta se encuentran los archivos que se ejecutan al momento de iniciar el sistema operativo.

Particularmente para ejecutar el demonio, el acceso directo ejecutará un archivo del tipo .bat (*simsiDaemon.bat*) que a su vez ejecutará en línea de comando el archivo demonio.jar en el cual está contenido el demonio y todas sus funcionalidades.

El archivo *simsiDaemon.bat* se encuentra con una ruta específica para ejecutar el demonio. Esta ruta es la que viene por defecto en el instalador. En el caso que se cambie durante la instalación, se deberá editar este archivo efectuando la misma modificación.

Resumiendo, para instalar correctamente SiMSI en el servidor es necesario entonces tener previamente instalado el servidor WAMP proporcionado así como un JRE.

Con estas precondiciones cumplidas se procede a instalar la aplicación SiMSI procurando que la dirección en la cual hacemos la instalación sea la misma que la del Servidor WAMP.

Esta advertencia se hace durante el proceso de instalación (figura 4.13) para que el usuario esté al tanto en el momento del mismo.

Si no se instala en el mismo directorio, el servidor WAMP no encontrará los archivos referidos a SiMSI para la Base de Datos ni los diseñados para la aplicación web.

Teniendo el WAMP instalado en el servidor, así como los archivos SiMSI y el demonio, ya es posible el funcionamiento de toda la aplicación. Sin embargo, para acceder a ésta desde cualquier punto de la web todavía no.

Para lograr esta funcionalidad existen varias opciones:

La primera es la posibilidad de adquirir un dominio web, de modo de tener un nombre para la dirección IP del servidor. De esta manera, con la dirección web, cualquier persona puede acceder a la página de inicio de la herramienta, y con su usuario y password acceder a todas las funciones permitidas.

En el proyecto SiMSI se utilizó la aplicación no-ip que permite obtener un nombre de host gratis para cualquier IP dinámica [19]. Accediendo al Sitio de No-ip se puede crear un usuario y un nombre para el sitio deseado. De esta manera, se obtiene una dirección web con cualquier extensión de dominio (.com, .org, .edu, etc) pero con .no-ip obligatoriamente en ella. Esta es una solución que permite el desarrollo académico del proyecto ya que es muy simple de utilizar y provee un servicio gratuito, pero para un futuro proyecto comercial se recomendaría utilizar la primera opción. De esta manera, el nombre del dominio es totalmente independiente de No-ip

Si se quiere un acceso más restringido a la herramienta, se puede configurar una VPN entre el servidor en el campo y los usuarios en sus respectivas PCs. Con la VPN se pueden tener los mismos privilegios y servicios que en una red privada, pero utilizando una red pública. Esto provee mayor seguridad a los usuarios de la aplicación y sería la opción recomendada si esta aplicación es utilizada dentro de una empresa por ejemplo.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

Luego de un año de trabajo e investigación, y teniendo en cuenta las restricciones, limitaciones y objetivos planteados, se puede decir que se logró con éxito la implementación del proyecto SiMSI.

Se estudió y decidió el protocolo a utilizar para el envío y recepción de datos en la comunicación entre el router y el servidor, optando finalmente por la tunelización de serial sobre TCP.

Se desarrolló un driver en el router que maneja la comunicación bidireccional entre éste y el mote central. Dicho driver recibe los datos a través del puerto USB y los coloca en un puerto TCP a disposición del servidor, y viceversa.

Se logró crear una imagen SiMSI con la cual programar los routers a utilizar en el sistema de monitoreo, automatizando el proceso de instalación de los mismos.

Se implementó un servidor con los servicios correspondientes al de servidor web y de base de datos, utilizando Windows como sistema operativo.

Se desarrolló un demonio que se encarga de recolectar, procesar y almacenar los datos recibidos desde el router, como también enviar información necesaria para la configuración y manejo de los motes.

Se creó una interfaz web amigable al usuario para, entre otras cosas, observar los datos recolectados en reportes personalizados, configurar los distintos equipos de la red, crear y editar usuarios y observar los logs del sistema.

Se logró implementar un sistema de alarmas, donde el usuario puede seleccionar a quién y cómo avisar (e-mail y/o sms), en las circunstancias que crea convenientes. Además de estos mensajes enviados especialmente, el usuario puede visualizar en cualquier página de la interfaz web un cuadro que muestra las alarmas activadas al momento.

Se embebió la aplicación GoogleMaps en la interfaz web para que el usuario pueda visualizar en un mapa los equipos instalados. De esta forma se puede entender la topología de la red de routers y motes, obtener últimos datos recibidos, identificar alarmas activadas y otros datos relevantes de manera gráfica e intuitiva.

Finalmente se creó un archivo ejecutable que se encarga de instalar y configurar los elementos necesarios en el computador.

Se considera que el proyecto implementado se aproxima a lo que es el desarrollo de una aplicación comercial. Se estudiaron las necesidades del cliente, se investigaron las herramientas a utilizar y se obtuvo una aplicación

global, eficiente y simple de usar cumpliendo con los requisitos establecidos. Para esto se conjugaron varios módulos independientes teniendo que trabajar en distintas ramas de la ingeniería, como programación en lenguajes de alto y bajo nivel, diseño de la base de datos y de la arquitectura de la red.

A los desafíos técnicos sobrellevados en el proyecto se agregó todo lo que implica el desarrollo de una aplicación de principio a fin. Para esto se contó con el apoyo de Leonardo Steinfeld y Pablo Mazzara quienes suplantaron la ausencia del tutor inicial. Teniendo como base un trabajo en grupo eficiente y organizado se logró finalizar, de gran manera, el proyecto planteado a principios del año 2008.

5.2. Trabajo a futuro

Una vez finalizado el proyecto y evaluando el producto obtenido se encontraron varias mejoras y complementos al Sistema de Monitoreo de Sensores Inalámbricos desarrollado.

Dado que este proyecto es una primera etapa en el desarrollo de una solución final, futuros emprendimientos deberían crear una imagen de router SiMSI para un router outdoor (que como se dijo, todo el trabajo realizado fue hecho para que este proceso sea lo más fácil posible), mejorar la aplicación Web para que soporte cualquier tipo de explorador, realizar una mayor interacción entre el protocolo de los motes (RSIS) y la aplicación como por ejemplo que los motes manden la hora de sensado del dato.

Otro tema a evaluar es la configuración del tiempo de muestreo específico por sensor y no por mote como se realiza hoy. Según lo discutido en reuniones con tutores, co-tutores y otros grupos que trabajan con los motes, esto puede no ser necesario ya que una vez que se despierta el mote no vale la pena que no mande todos los datos que pueda obtener ya que el mayor consumo de batería es debido al encendido.

Sería interesante además migrar el sistema a Linux para usuarios que deseen modificar esta solución o simplemente utilizar otro sistema operativo.

Otro de los grandes cambios que consideramos necesario es modificar el concepto de zona que se maneja hoy y como fue descripto por nuestro entonces cliente en la definición del proyecto. Hoy una zona está asociada a un router y los datos relevados quedan asociados a este equipo. O sea, el router marca la zona geográfica de los datos, por lo que si se mueve el router los datos anteriores no quedan asociados a ninguna zona y tampoco pueden existir dos routers en un mismo lugar. Vemos entonces importante agregar el concepto de zona.
Con este concepto, cada dato estaría asociado a una zona geográfica (por ejemplo “junto al lago”) y los routers serían transparentes a la solución final. Se podrían mover de lugar, agregar o quitar y ni los datos ni el usuario sentirían estos cambios.

Dado los grandes cambios climáticos que están ocurriendo hoy en día en todas partes del mundo, consideramos que para obtener una buena producción agraria es necesario contar con un sistema de monitoreo en el campo. En especial, sería interesante tener un sistema de control o de contingencia. Se podría entonces utilizar lo desarrollado por SiMSI y a partir de esto activar riego, ventiladores, etc.

Para finalizar, consideramos que el sistema de monitoreo desarrollado está pronto para ser instalado en un ambiente real. Una vez en producción, en base a los resultados obtenidos y a la experiencia de los usuarios, se podrán desarrollar nuevas funcionalidades que permitan una mejora significativa del sistema.

Bibliografía

[1] Wikipedia, Tiny OS, http://es.wikipedia.org/wiki/TinyOS

[2] Sitio de la empresa Sentilla, www.sentilla.com

[3] Wikipedia, PHP, http://es.wikipedia.org/wiki/Php

[4] Wikipedia, Demonio (informática), http://es.wikipedia.org/wiki/Demonio_(informatica)

[5] El Lenguaje de Programación Java, Guías de Clase, Parte I: Lenguaje de Programación Java, DesaSoft - Desarrollo de Software para Ingeniería Eléctrica, IIE - Instituto de Ingeniería Eléctrica

[6] Sitio de TinyOs, www.tinyos.net

[7] Foro de Sun, Tema de Array List vs Vector, http://forums.sun.com/thread.jspa?threadID=756553\&start=0\&tstart=0

[8] Sitio Oficial de Java (Sun), Using JConsole to Monitor Applications, http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html

[9] Wikipedia, CSS, http://es.wikipedia.org/wiki/CSS

[10] Wikipedia, JavaScript, http://es.wikipedia.org/wiki/JavaScript

[11] Wikipedia, Google Maps, http://es.wikipedia.org/wiki/Google_Maps

[12] Wikipedia, Google Maps API, http://en.wikipedia.org/wiki/Google_Maps#Google_Maps_API

[13] Sitio de Google Code, Google Maps API, http://code.google.com/intl/es-AR/apis/maps/

[14] Sitio de SQLiteManager, http://www.sqlitemanager.org

[15] Sitio Oficial de WAMP, www.wampserver.com

[16] Sitio Oficial de Phpmyadmin, www.phpmyadmin.net

[17] Sitio Oficial de Java, Sección Descargas, java.com/es/download

[18] Sitio Oficial de Firefox, Descarga de Firefox en español, http://es-ar.www.mozilla.com/es-AR/

[19] Sitio de No-ip, www.no-ip.com

[20] Wikipedia, Extensible Markup Language, http://es.wikipedia.org/wiki/XML

[21] Foro de Universidad de Berkeley, EEUU, Tema: MoteIF try catch on java, https://www.millennium.berkeley.edu/pipermail/tinyos-help/2007-September/027847.html

Apéndice A

Estudio de Protocolos para Envío de Datos al Servidor

Una de las primeras decisiones tomadas en el proyecto fue el protocolo a utilizar para la comunicación bidireccional entre el router y servidor. Las opciones evaluadas fueron XML, SNMP y tunelización de Serial sobre TCP. La elección final fue la utilización de tunelización desarrollada en la sección 2.1.1. Una de las características más importantes es que el router no debe procesar ni entender la comunicación entre las puntas.

Por otro lado, tanto con SNMP como con XML se debe explicitar en el router de dónde leer los datos a enviar y dónde escribir los recibidos. Además de esto, se deberá ejecutar un driver que haga la traducción a serial de estos datos para enviarlos al mote o extraer del mensaje serial los campos a enviar al servidor. Esto, entre otras cosas, obliga a definir en el router los tipos y estructuras de mensajes, debiendo compilar y modificar el firmware cada vez que se modifique el protocolo de los motes. Además, dado que por lo general el sistema operativo de los routers es muy limitado se decidió pasar la mayor cantidad de inteligencia al servidor, donde hacer modificaciones es más simple y se deben realizar solamente en un lugar (en vez de en todos los routers por ejemplo).

A.1. SNMP

SNMP (Simple Network Management Protocol) es uno de los protocolos de gestión de redes con más aceptación hoy en día. El mismo es un protocolo de capa de aplicación que utiliza las funcionalidades brindadas por TCP/IP y permite el monitoreo en redes complejas con equipos de varios fabricantes y distintas topologías de red.

Existen 3 componentes claves utilizados por SNMP: Administradores, Dispositivos administrados y agentes.

Los administradores, ubicados en los equipos de gestión de red (en nuestro caso el servidor), son los encargados de supervisar y controlar a los dispositivos administrados. Estos últimos (routers, Access Points, servidores, etc.) envían al administrador los datos recolectados ya sea periódicamente o en base a consultas. Estos datos son especificados en el agente, el cual los traduce a un formato SNMP y los organiza en jerarquías.

Para poder enviar los datos recibidos desde el mote por SNMP es necesario configurar una rama especial en una MIB. Una MIB es una base de datos conteniendo la información a monitorear organizada jerárquicamente.

Es posible también configurar Traps en los Dispositivos Administrados, que son mensajes que el dispositivo manda automáticamente cuando detecta cierto suceso. En nuestro caso podría ser la recepción de un mensaje de alarma, donde se desea enviar y procesar esta información lo más rápidamente posible.

Existen varias versiones de SNMP, implementando en las versiones 2 y 3 un mecanismo de seguridad por cifrado y métodos de acceso a la información más confiables.

Por otro lado, existen varias aplicaciones que corren en los servidores administradores para realizar el procesado y despliegue de información, por ejemplo MRTG. Esta herramienta soluciona el despliegue de datos desarrollando un informe en formato HTML con gráficas, mostrando la variación en el tiempo de las variables seleccionadas.

En un principio MRTG fue diseñado para monitorear simplemente el tráfico de una red pero a medida que se popularizó, se fue adaptando para el manejo de otro tipo de información.

En cuanto a la instalación en el router, se debería instalar el lenguaje LUA que contiene una librería para trabajar con SNMP y permite ejecutar un script determinado al realizar una consulta a la MIB en busca de un valor determinado. Este script debería leer los datos recibidos por el driver que interactúa con el mote a través del puerto USB, o, en caso de una comunicación Servidor - Mote, debería escribir el mensaje en algún lugar donde el driver lo iría a leer periódicamente.

LUA es un lenguaje de programación compacto diseñado como un lenguaje de script que obtuvo gran popularidad en el desarrollo de videojuegos debido a su gran velocidad y pequeño tamaño. Se puede considerar que LUA es un lenguaje desarrollado para poder ser utilizado en variadas aplicaciones.

En vez de definir estructuras complejas, define una pequeña cantidad de propiedades generales que pueden ser extendidas permitiendo, por ejemplo, utilizarlo en programación orientada a objetos.

Como se dijo anteriormente, y se puede ver en lo descripto en esta sección, la utilización de SNMP obliga al router a manejar inteligentemente los paquetes. Esto quiere decir, interpretar los mensajes, extraer la información pertinente y traducirla al protocolo necesario para lograr la comunicación. Para esto, además de soportar el procesamiento necesario, deberá conocer los protocolos y tipos de paquetes utilizados. Dado que uno de los puntos claves establecidos al momento del diseño de la solución es que el router no tenga un rol importante al momento de expandir la red ya sea con más protocolos, tipos de mensajes, etc., esta opción fue descartada.

A.2. XML

XML es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium*. Decimos metalenguaje ya que no es un lenguaje en sí mismo sino que permite definir lenguajes a partir de él. A partir de esto se propone XML como un estándar para el intercambio de información estructurada, y se puede aplicar en varias situaciones como manejo de base de datos, edición de texto, configuración remota de equipos, etc.

La idea detrás de este metalenguaje es lograr organizar la información de una forma estructurada, agrupando la misma en partes y subpartes. Se logra entonces tener el documento separado en bloques de información. Una etiqueta es una marca en el documento que permite asociar un bloque de texto como un elemento.

Todo documento XML debe tener definida su gramática que se explicita en la DTD (Document Type Definition). Allí se definen los distintos tipos de elementos, atributos y entidades permitidas. Para que un documento se declare como válido los elementos utilizados, deben estar declarados en la DTD.

Un ejemplo básico de un documento XML es el siguiente:

<?xml version=“1.0” encoding=“ISO-8859-1” ?>

<!DOCTYPE Edit_Mensaje SYSTEM “Lista_datos_mensaje.dtd”
[<!ELEMENT Edit_Mensaje (Mensaje)*>]>

```
<Edit_Mensaje>

   <Mensaje>

      <Remitente>
         <Nombre>Nombre del remitente</Nombre>
         <Mail> Correo del remitente </Mail>
      </Remitente>

   <Destinatario>
      <Nombre>Nombre del destinatario</Nombre>
      <Mail>Correo del destinatario</Mail>
   </Destinatario>


      <Texto>
         <Asunto>
            Este es mi documento con una estructura muy sencilla
            no contiene atributos ni entidades....
         </Asunto>
         <Parrafo>
            Este es mi documento con una estructura muy sencilla
            no contiene atributos ni entidades....
         </Parrafo>

      </Texto>

   </Mensaje>

</Edit_Mensaje>
```

La DTD asociada a este documento es la siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!– Este es el DTD de Edit_Mensaje −− >

<!ELEMENT Mensaje (Remitente, Destinatario, Texto)*>
   <!ELEMENT Remitente (Nombre, Mail)>
      <!ELEMENT Nombre (PCDATA)>
      <!ELEMENT Mail (PCDATA)>

   <!ELEMENT Destinatario (Nombre, Mail)>
      <!ELEMENT Nombre (PCDATA)>
      <!ELEMENT Mail (PCDATA)>

   <!ELEMENT Texto (Asunto, Parrafo)>
      <!ELEMENT Asunto (PCDATA)>
      <!ELEMENT Parrafo (PCDATA)>
```

Un documento XML tiene una estructura bien definida que debe respetarse. En primer lugar se cuenta con un prólogo (que no es obligatorio) en el que generalmente se informa de la versión de XML a utilizar, así como la DTD correspondiente al documento. Otra parte de la estructura es el cuerpo, en el que hay un único elemento raíz. Dentro de este elemento, se definen todos los elementos pertenecientes al documento. Los elementos pueden

tener atributos que lo caractericen [20].

En el caso del proyecto SiMSI, se podría evaluar una aplicación del envío de datos aplicando XML. Para esto se podría generar las etiquetas en forma jerárquica, teniendo por ejemplo: router, mote_id, sensor_id, dato. De esta forma se simplifica la tarea a la hora de procesar los datos y se tiene la información de una manera estructurada y ordenada.

Durante el estudio de XML como forma de envío, se tomo conocimiento de la existencia de una codificación específica estándar para describir procesos de sensores. Este estándar, aprobado por el Open Geospatial Consortium, es llamado SensorML. Algunos de los procesos descriptos en la codificación son medidas de datos, ubicaciones geográficas, auto-descubrimiento de sensores, etc. Sensor ML provee codificaciones pre-definidas que logran estandarizar los documentos XML a utilizar según el tipo de aplicación. De esta manera, cualquier entidad podría entender los datos enviados por un sensor sin ser parte de la red de sensores. A través de este estándar los sensores se hacen conocer describiendo sus posibilidades y posibles entradas y salidas. Dado que hoy en día existen muchas redes de sensores desplegadas por todos lados, se busca que las mismas se apoyen entre ellas. Se fomenta entonces el desarrollo de aplicaciones que utilicen varios tipos de sensores instalados por distintas compañías para brindar una idea global de lo que está pasando en ese momento o lugar.

En el 2006 la NASA entró como uno de los sponsors del estándar y ya existen algunas herramientas que analizan los datos de distintas redes de sensores por medio de SensorML.

De igual manera que se descartó SNMP por incrementar la inteligencia en el router se descarta XML y sensorML.

Apéndice B

Pruebas del Sistema

A modo de verificar el funcionamiento del sistema se realizaron distintas pruebas. Se buscó verificar su funcionamiento ininterrumpido, obtener una correcta ejecución de los casos de uso y simular situaciones límite de no conexión.

Para las pruebas se dispuso de un router y dos motes (uno de ellos central). Esta cantidad de equipos está lejos de permitir tener una configuración exigente que permita simular una situación real. Lo ideal sería tener al menos dos routers y unos cinco motes por cada uno como punto de partida. Si bien no se logró aumentar la cantidad de motes, se logró simular un router adicional por medio de un PC con Linux instalado.

Se puede ver en la figura B.1 una foto de los motes y router utilizados para las pruebas:

Figura B.1: Motes y Router utilizados durante el proyecto

A continuación se mostrarán en detalle algunas de las pruebas realizadas.

B.1. Ejecución sin router, ni mote central

Se ejecutó el demonio sin estar el servidor central conectado a ningún router. Se puede observar en la figura B.2 cómo se identifica que no hay enlace a ninguno de los routers configurados y nada ocurre. También puede observarse en la parte inferior derecha el ícono de Windows que muestra que el cable está desconectado.

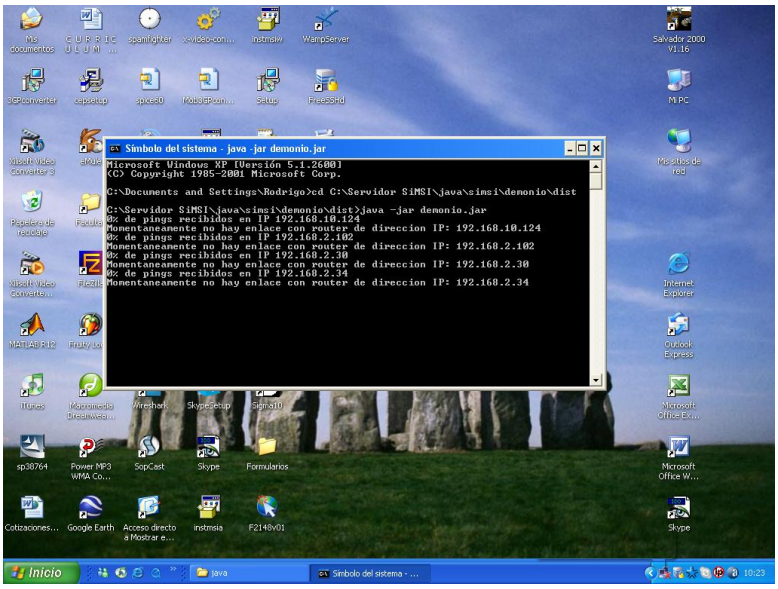

Figura B.2: Ejecución sin router ni mote central conectado

Luego conectamos el servidor y vemos cómo después de algunos minutos se vuelve a verificar el estado de enlace y se intenta levantar una conexión con cada router conectado (figura B.3). Sin embargo, el mote central no estaba conectado por lo que salta una interrupción por cada router sin mote central.

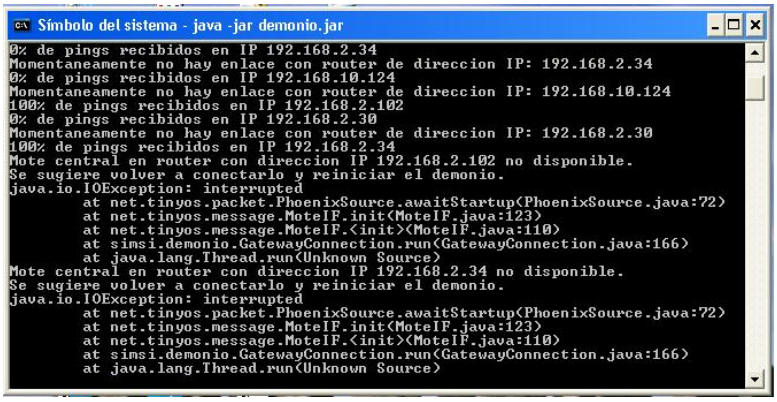

Figura B.3: Interrupción por mote central desconectado

Originalmente cuando se intenta levantar un túnel y no se detecta el mote central, se genera una excepción en la clase *MoteIF* que forma parte de la API provista por TinyOS. El problema es que esta excepción no es lanzada, sino que está implementado internamente en dicha clase, que se ejecute la función *System.exit(2)* y se salga de la ejecución del programa. Esto fue un inconveniente porque la idea es que no suceda y que el demonio se ejecute sin interrupción.

No hay forma de modificar la clase *MoteIF* pero sí existe un truco que permite continuar con la ejecución sin salir. Se implementó en la clase *GatewayConnection* la interfaz *PhoenixError* y se sobreescribe el método *error(IOException ioe)* que es el responsable de llamar al *System.exit(2)*. Dentro de este método se generó una interrupción que corta un loop infinito que queda al no poder levantar el enlace con el mote central y con esto se logra continuar la ejecución del demonio sorteando el error.

Dado que lo ideal sería atrapar esta excepción con un catch y esto no es posible, se recomienda al usuario (dado el caso y en la medida de lo posible) reconectar correctamente los motes centrales y reiniciar el sistema a pesar de que el demonio continúa ejecutándose sin inconvenientes. Puede encontrarse más información respecto a este tema en el foro de la universidad de Berkeley [21].

Como puede verse en este ejemplo, si luego de detectada la falla conectamos el mote central en el router, el demonio continúa su ejecución. En la figura B.4 puede observarse cómo se levanta un túnel con la IP 192.168.2.34 (router al que se le conectó el mote central) y luego se reciben mensajes correctamente.

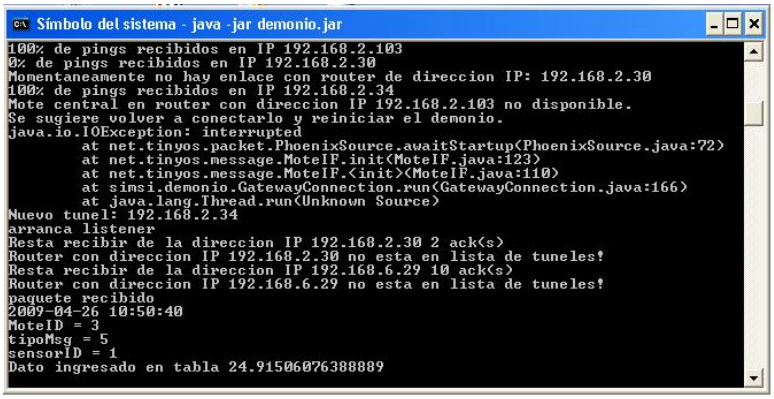

Figura B.4: Reconexión de mote central

B.2. Envío cambio de período de muestreo con mote central desconectado y túnel levantado

Para esta prueba se inició el sistema normalmente y se configuró una nueva frecuencia. Sin embargo, en el tiempo entre que se verifica el estado de enlace y se envía un cambio de frecuencia se desconectó el mote central para estudiar su comportamiento. Puede verse en la figura B.5 que al intentar enviar la nueva frecuencia se detecta la falta del mote central. Debido a esto se interrumpe la ejecución y se quita el túnel de la lista.

Luego se conectó nuevamente el mote central y se puede observar que finalmente vuelve a levantarse el túnel, enviarse el cambio de frecuencia y se recibe el ack correspondiente.

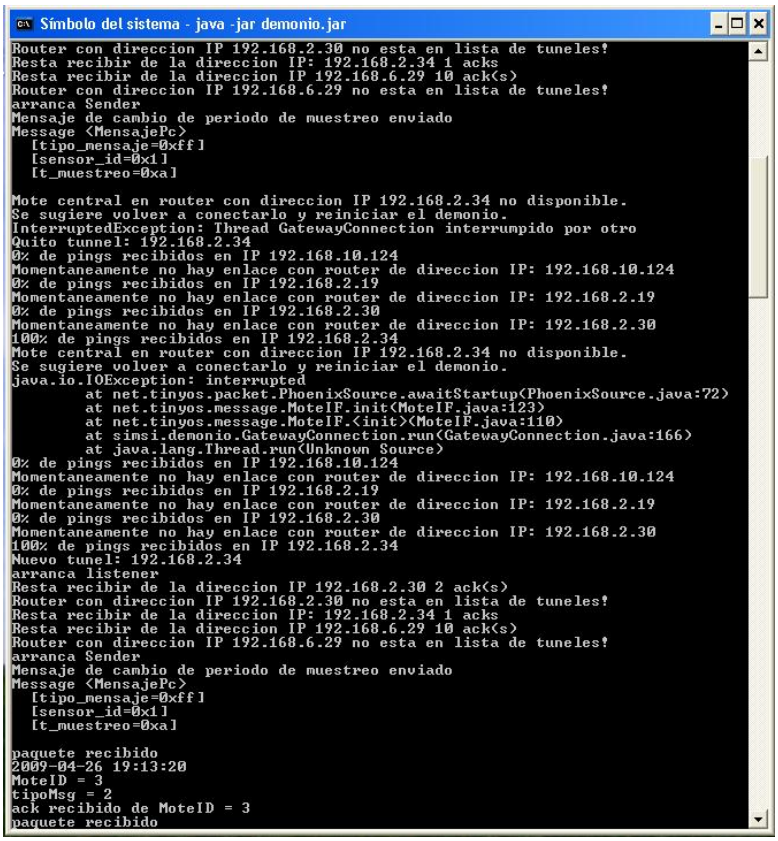

Figura B.5: Envío cambio de período de muestreo con mote central desconectado y túnel levantado

B.3. Desconexión del cable de red

Una vez que se estaban recibiendo mensajes de la red de sensores se desconectó el cable de red del servidor.

En una situación similar a la prueba de la sección B.1, se detecta el error y prosigue la ejecución. Puede verse el detalle de la ejecución en las figuras B.6 y B.7.

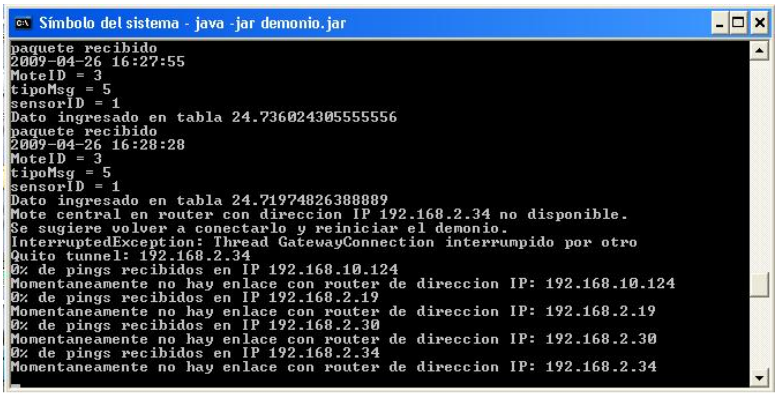

Figura B.6: Ejecución al desconectar el cable de red

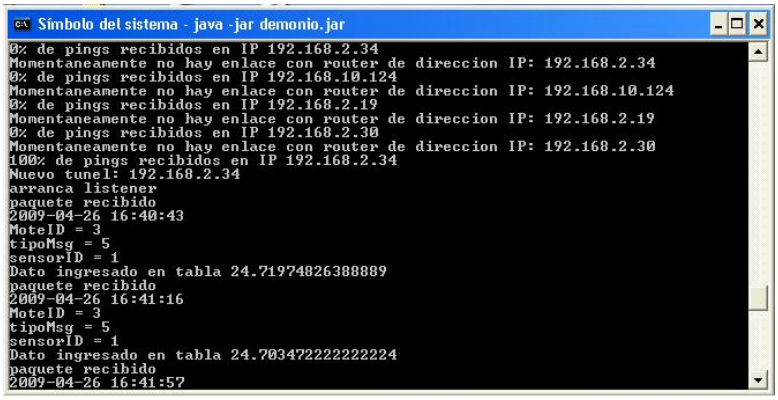

Figura B.7: Recuperación del túnel con cable de red reconectado

B.4. Red con un salto intermedio

Otra prueba interesante fue simular con un PC otro router, conectarlo al router que se disponía y de esta forma obtener una topología de red con un salto intermedio.

Se observa en la figura B.8 que al conectar los motes a este PC se reciben mensajes sin ningún inconveniente. Incluimos además en la figura B.9 la consola de TinyOS, ejecutada en el router simulado, donde se muestra la recepción de mensajes que luego son reenviados.

Figura B.8: Mensajes recibidos desde Router simulado en un PC

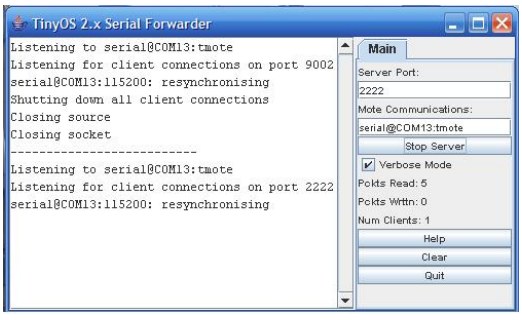

Figura B.9: Ejecución de Serial Forwarder en Router simulado

B.5. Cambio de frecuencia y activación de alarma

Se inició el sistema normalmente y se reconfiguró en la página el período de muestreo a 1 (un) minuto. Puede verse el mensaje *ack* recibido y cómo luego empiezan a recibirse mensajes a esa tasa (figuras B.10 y B.11).

Reconfiguramos la alarma de temperatura correspondiente a ese router de forma de que se active (figura B.12). Puede verse cómo finalmente se activa y se envían los mensajes a los usuarios en la figura B.13 y figura B.14.

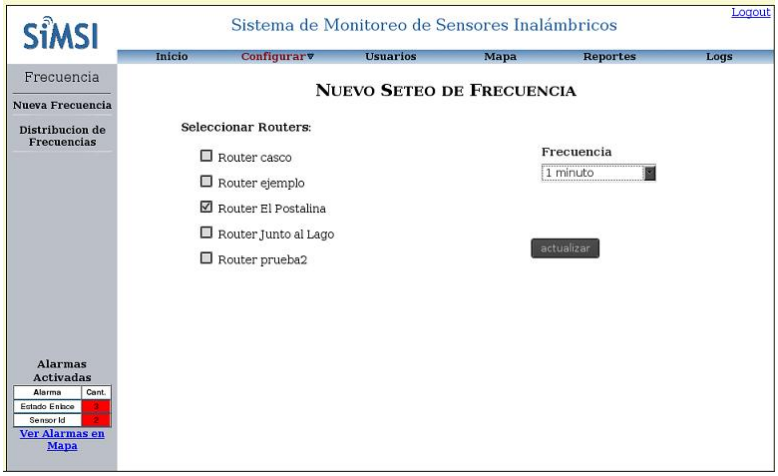

Figura B.10: Cambio de frecuencia en página

Figura B.11: Cambio de frecuencia en demonio

Además el despliegue de las alarmas puede verse en cualquier página de la web (figura B.15) y con mayor detalle en el mapa (figura B.16).

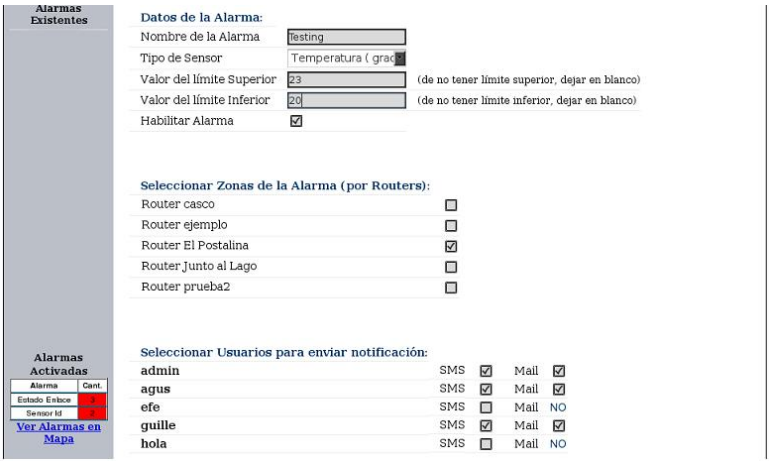

Figura B.12: Configuración de alarma en página

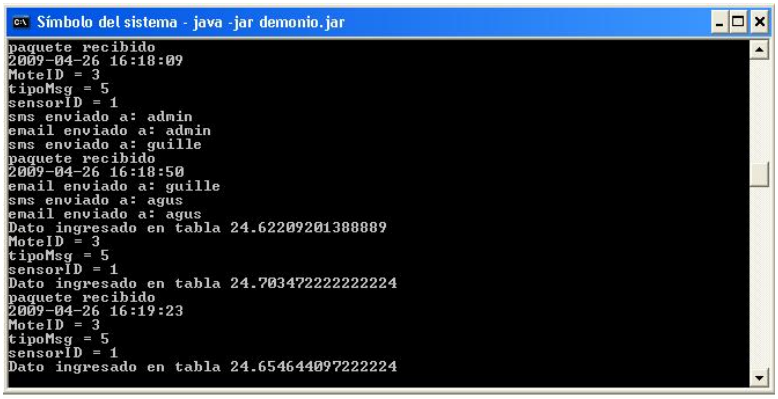

Figura B.13: Detección de alarma en demonio

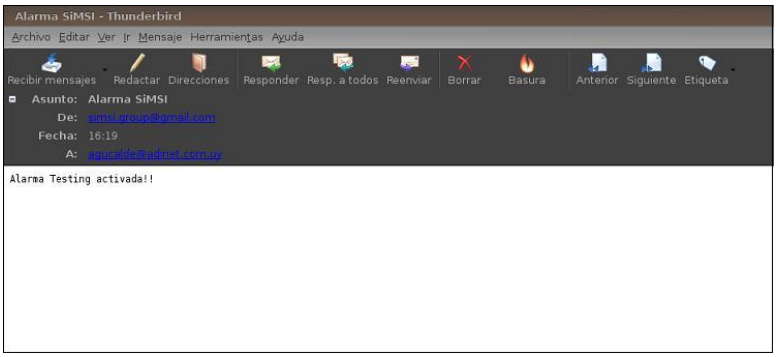

Figura B.14: Notificación de alarma vía e-mail

Figura B.15: Notificación de alarma en la aplicación

Figura B.16: Notificación de alarma en mapa

B.6. Topología y estado de baterías

Se comprobó que se envían y reciben correctamente los mensajes del tipo de solicitud de topología y de sensado de baterías.

B.7. Limpieza de la Base de Datos

Se ejecutó el thread *CleanDB* para verificar su funcionamiento. Previamente se modificó la fecha de una gran cantidad de datos obtenidos de un mote al año 2008 de forma de que fueran reconocidos como datos anteriores a un año.

Se comprobó que la ejecución fue efectiva y se promediaron los datos. En la figura B.17 logra observarse cómo se eliminan datos que están siendo promediados.

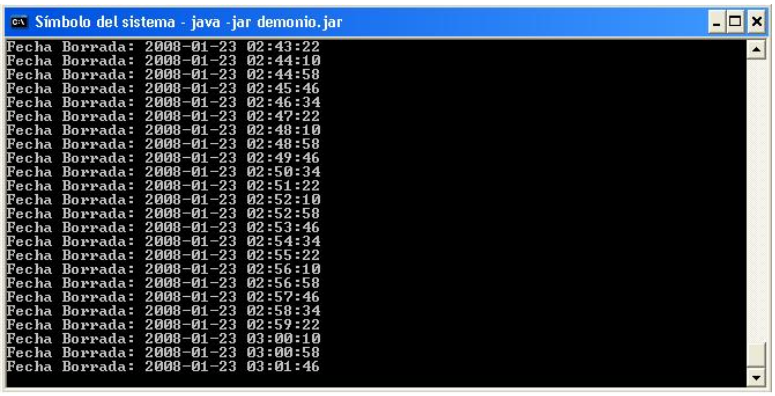

Figura B.17: Limpieza de la Base de Datos

B.8. Vista de logs

En los logs puede apreciarse la actividad reciente. En las figuras B.18 y B.19 se ven los registros correspondientes al cambio de frecuencia y activación de alarmas de la sección B.5.

Figura B.18: Log de Configuración de Equipos

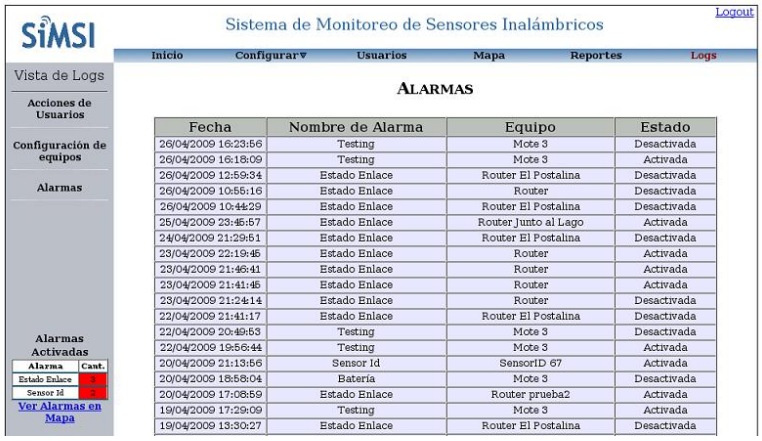

Figura B.19: Log de Alarmas

B.9. Configuración de Nuevo Reporte

Crear un nuevo reporte es otra de las funciones interesantes de SiMSI.
Se puede ver en la figura B.20 la configuración del mismo y en la figura B.21 el gráfico obtenido.

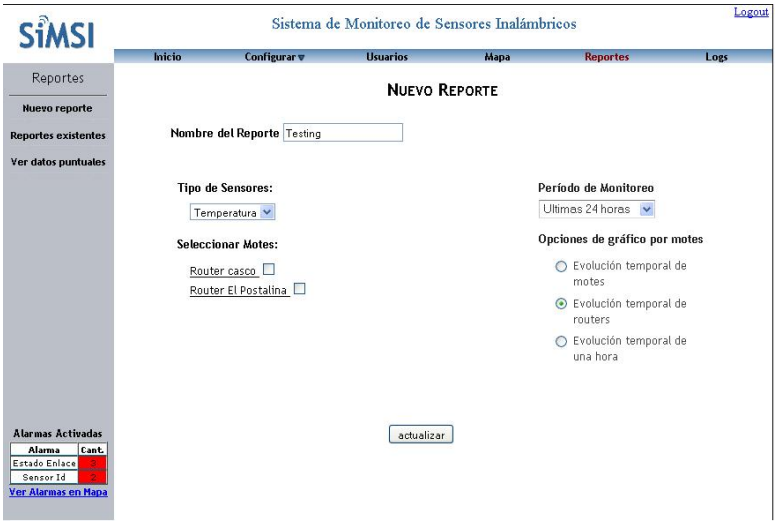

Figura B.20: Configuración de Nuevo Reporte

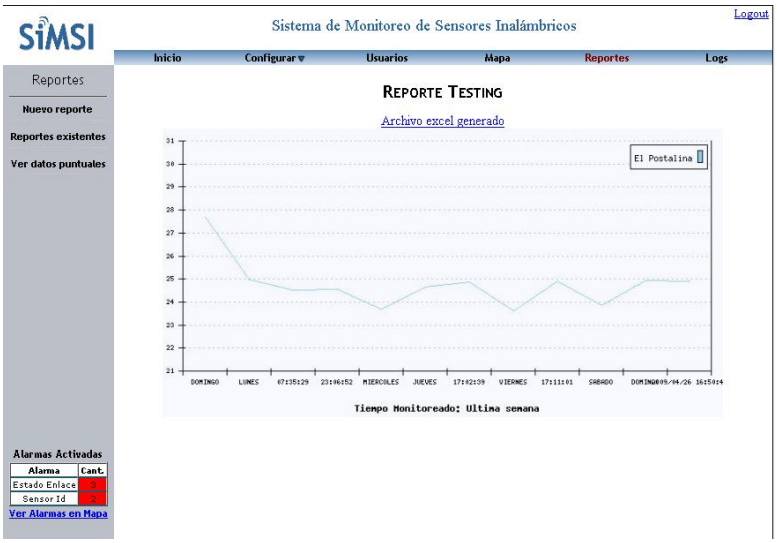

Figura B.21: Gráfico obtenido

B.10. Recursos consumidos por el Router

Como se puede ver en la salida del comando “top” ejecutado en el router, los recursos de CPU consumidos son mínimos:

Mem: 16776K used, 13516K free, 0K shrd, 2096K buff, 6324K cached

Load average: 0.00 0.07 0.05

PID	USER	STATUS	RSS	PPID	%CPU	%MEM	COMMAND
841	root	R	440	463	1.9	1.4	top
451	root	S	768	173	0.0	2.5	dropbear
102	root	S	764	1	0.0	2.5	httpd
106	root	S	608	1	0.0	2.0	httpd
463	root	S	596	451	0.0	1.9	sh
175	root	S	512	1	0.0	1.6	pppd
195	root	S	508	188	0.0	1.6	sh
173	root	S	500	1	0.0	1.6	dropbear
188	root	S	476	1	0.0	1.5	IniciSF.startup
178	root	S	440	1	0.0	1.4	nas
99	root	S	400	1	0.0	1.3	telnetd
1	root	S	388	0	0.0	1.2	init
472	root	S	368	195	0.0	1.2	sleep
325	root	S	340	1	0.0	1.1	igmprt
320	root	S	328	1	0.0	1.0	process$_m$*onitor*
115	root	S	320	1	0.0	1.0	dnsmasq
13	root	S	308	1	0.0	1.0	watchdog
114	root	S	304	1	0.0	1.0	wland
70	root	S	304	1	0.0	1.0	resetbutton
176	root	S	300	1	0.0	0.9	redial
397	root	S	280	1	0.0	0.9	cron

Apéndice C

Evaluación de la Posibilidad de Inversión

En el apéndice que se desarrolla a continuación se busca evaluar la viabilidad del proyecto SiMSI en la realidad del agro uruguayo. Para ello, en primer lugar, se evaluarán los costos de la implementación en sí misma y cómo estos varían modificando ciertos parámetros en el cálculo. Luego se estudiará la situación tomando un caso real de una explotación y viendo los costos que generaría la inclusión del sistema de monitoreo en la plantación.

C.1. Evaluación de los costos

Según lo visto en la sección 2.3, el alcance de un router ubicado a 1.5 metros de altura es de 200 metros en condiciones óptimas. El alcance de los motes, según lo estudiado en la sección 2.4, es de entre 50 y 75 metros, por lo que para el presente análisis se tomará 50 metros, siendo conservadores. El área de cobertura de cada router es entonces de $200^2 \times 3{,}14 = 125600m^2$ pudiendo cubrir un área de 12,5 has. El área de cobertura de un mote es de $50^2 \times 3{,}14 = 7850m^2$. Para cubrir con motes el área que cubre un router, necesitamos 16 motes dispuestos uniformemente. El costo de un mote es de, aproximadamente, U$S 80 y el del router planteado en el Capitulo 9 (MikroTik RB230) alcanza los U$S 350. El costo en equipos por área de cobertura de un router (12 has), asciende a U$S 1600, lo que equivale a unos U$S 130 por hectárea.

El panorama presentado anteriormente fue un tanto optimista en cuanto a la cobertura del router. Decimos esto ya que en algunos casos tal vez algún router no llegue a cubrir los 200 metros debido a posibles obstrucciones como árboles, edificaciones, etc. En un panorama más conservador se puede suponer, por ejemplo, que se cubren 9 hectáreas. En este caso se necesitarían entonces 12 motes distribuidos uniformemente, alcanzando un costo de U$S 145 por hectárea, sólo en equipos. Ello muestra que no es un cambio muy significativo en la inversión por ha.

Hasta el momento se estudió el caso de ubicar los motes en el área de cobertura del router de manera uniforme. Se podría evaluar la posibilidad de tener más precisión y, para esto, sería necesario instalar más motes. Por ejemplo, en el caso en el que se supone que el router cubre un área de 9 has, en lugar de 12 motes se podrían instalar 20 y, en ese caso, el costo asciende a U$S 220 por hectárea.

Los costos anteriormente detallados incluyen solamente el equipamiento en el lugar de la plantación. A ellos debe sumársele, el costo de llegar desde el casco de la estancia (donde estará ubicado el servidor) hasta el primer router de la plantación. En lo explicado en la sección 2.3, suponiendo que el router ubicado en el casco se encuentre a 6 metros de altura se tendría un alcance de 600 metros. Si la plantación se ubica a una distancia mayor serán necesarias otras consideraciones como, por ejemplo, instalar más routers en el camino o que los routers cuenten con antenas más potentes. Otro costo a tener en cuenta es el del servidor en sí, que es el costo de un PC doméstico. Por otra parte, el hecho de instalar los routers y los motes en su debido lugar para lograr

la cobertura deseada implica un costo de ingeniería que varía según el caso. El productor podrá tener también un costo mensual de mantenimiento y control de los equipos en cuestión, ya sean routers, motes o el servidor en sí mismo. Otro costo mensual que el productor deberá tener en cuenta es el acceso a internet, para que los usuarios remotos puedan acceder al sistema.

Todas estas suposiciones se hacen en base a que en el lugar de la plantación sea posible tener acceso a corriente eléctrica. En muchos casos la plantación en sí requiere la instalación de riego manejado por válvulas que están alimentadas a 220V. Pero si no es el caso, se puede tornar realmente caro lograr tener acceso a 220V en el medio de la plantación. Este costo varía obviamente con la plantación, pudiendo llegar a valores que puedan tornar la inversión inviable.

C.2. Análisis del costo total en un campo

En esta parte se realiza un análisis del costo global en el que incurre un productor a la hora de implementar SiMSI. Se supondrá que el campo tiene las siguientes características detalladas a continuación.

Se toma como tamaño de la plantación 50 has.

La distancia supuesta desde el casco hasta el primer router de la plantación se supone menor a 600 metros.

En el tramo del casco de la estancia se supone que no hay grandes arboledas o maquinaria que puedan generar obstrucciones de la señal.

Como se explicó en la parte anterior es necesario poder tener acceso a alimentación 220V para los routers.

Teniendo en cuenta las consideraciones anteriores, serán necesarios 6 routers en la plantación para cubrir el área deseada, tomando que un router cubre 9 has. Con el supuesto anterior se cubre más área de lo necesario, por lo tanto el estudio de la cantidad de motes a instalar se evaluará para cubrir el área total pudiendo un router tener más motes asociados que otro. El área cubierta por un mote es de 7850 m2 , por lo que para cubrir uniformemente las 50 has, serían necesarios 64 motes. Para lograr mayor precisión sería aconsejable instalar 75 motes, lo que llevaría a un costo de U$S 6000 en motes. Luego se deben sumar los 7 routers (6 en la plantación más uno en el casco), es decir U$S 2450, y el costo de un PC doméstico, calculado en unos U$S 600. El costo total de la implementación, sólo en equipos, asciende entonces a U$S 9050, es decir U$S 181 por hectárea plantada. A esto hay que agregarle el costo de la mano de obra de ingeniería para la instalación, más los costos mensuales del acceso a Internet y del mantenimiento.

El costo anteriormente calculado resulta muy esperanzador, pero hay que tener en cuenta que se supuso tener acceso a alimentación 220V en la plantación. Otro de los puntos que se debería estudiar es cuánto aumenta la producción al utilizar el sistema de monitoreo. De esta manera se podrá tener una estimación de las ganancias que produce SiMSI para luego poder evaluar si es conveniente o no la utilización. Esto último no se puede determinar hasta no poner el sistema en producción en una plantación y ver, al cabo de un año, los cambios.

Apéndice D

Manual de Usuario

En este manual el usuario de SiMSI encontrará los pasos necesarios para hacer uso de la aplicación.

D.1. Instalación

D.1.1. Requerimientos

Para instalar SiMSI en su Servidor deberá tener los siguientes requisitos mínimos:

- Windows XP/2003/Vista
- Procesador Pentium IV
- 512 MB de Memoria RAM
- 10 GB de espacio libre en el disco
- JRE (Java Runtime Environment) instalado en servidor

D.1.2. Procedimiento de Instalación

En primer lugar, deberá instalar el Servidor WAMP. Para ello, podrá usar la versión *WampServer Version 2.0c* provista en el CD SiMSI, o una versión más reciente descargándola del sitio oficial de WAMP.

Luego, ejecute el archivo *SiMSI setup.exe* y siga los pasos para completar la instalación.

ADVERTENCIA: Asegúrese que la ruta de instalación de SiMSI sea la misma que la del servidor WAMP instalado

El archivo *simsiDaemon.bat* (RUTA_INSTALACION/java/simsi/simsiDaemon.bat) es el encargado de ejecutar la interfaz con los routers. Éste contiene la ruta por defecto. En caso de cambiar la ruta de instalación, se deberá editar este archivo con la nueva ruta siguiendo las indicaciones dentro del mismo.

D.2. Desinstalación

Si se desea desinstalar SiMSI, se debe hacer clic en el ícono *uninstall SiMSI.exe* y seguir los pasos de la desinstalación.

Luego se deberá desinstalar el Servidor WAMP con su archivo desinstalador.

D.3. Uso de la Aplicación

En la página principal de cada sección y sub-sección usted dispondrá de una ayuda para la utilización de la misma.

D.3.1. Logueo al sistema

Si usted se loguea por primera vez a SiMSI, debe ingresar como nombre de usuario *admin* y contraseña *admin*. Se recomienda cambiar la contraseña una vez ingresado por primera vez.

D.3.2. Deslogueo del sistema

Hacer clic en *Logout* en la esquina superior derecha de la aplicación.

D.3.3. Secciones

En el menú horizontal se encuentran las distintas secciones de SiMSI: *Inicio*, *Configurar*, *Usuarios*, *Mapa*, *Reportes*, *Logs*.

Inicio

En la página principal, existen vínculos directos a las secciones favoritas propias de cada usuario, como se aprecia en la figura D.1.

Si se ingresa por primera vez, aparece una notificación sobre lo anterior (figura D.2).

Figura D.1: Sección Inicio con los Favoritos del Usuario

Figura D.2: Notificación de Favoritos

Configurar

Dentro de esta sección hay distintas opciones posibles: *Frecuencia*, *Motes*, *Routers*, *Alarmas*, *Sensores*.

Routers En esta sección se configuran los routers de SiMSI. Haciendo clic en *Nuevo Router*, se configura un router nuevo (figura D.3)

Figura D.3: Crear Nuevo Router

Por otra parte, en la opción *Routers Existentes*, se despliegan los routers ya configurados en el sistema, y haciendo clic en el nombre podemos visualizar las características del router. Esto lo vemos en la figura D.4.

En las opciones de *Accesos directos del router* podemos entrar a la configuración del router, si el puerto fue seteado correctamente (figura D.5).

Para editar o eliminar routers, simplemente hacer clic en *Reportes Existentes* y luego en el ícono deseado ( para editar,  para eliminar).

Motes En esta sección se configuran los motes del sistema. Los motes que han sido instalados pero aún no configurados aparecen en la página principal de motes para ser editados directamente, como lo muestra la figura D.6.

Haciendo clic en alguno de ellos se procede a editar el mote (figura D.7).

En la opción *Estado de Motes*, en una primera instancia se deben elegir los motes a ser desplegados (figura D.8). Luego, seleccionando los motes deseados, se llega a un cuadro, donde se muestra, por mote, los valores de frecuencia deseada, actual, el mote padre, el router asociado, y opciones para editar y borrar al mote en

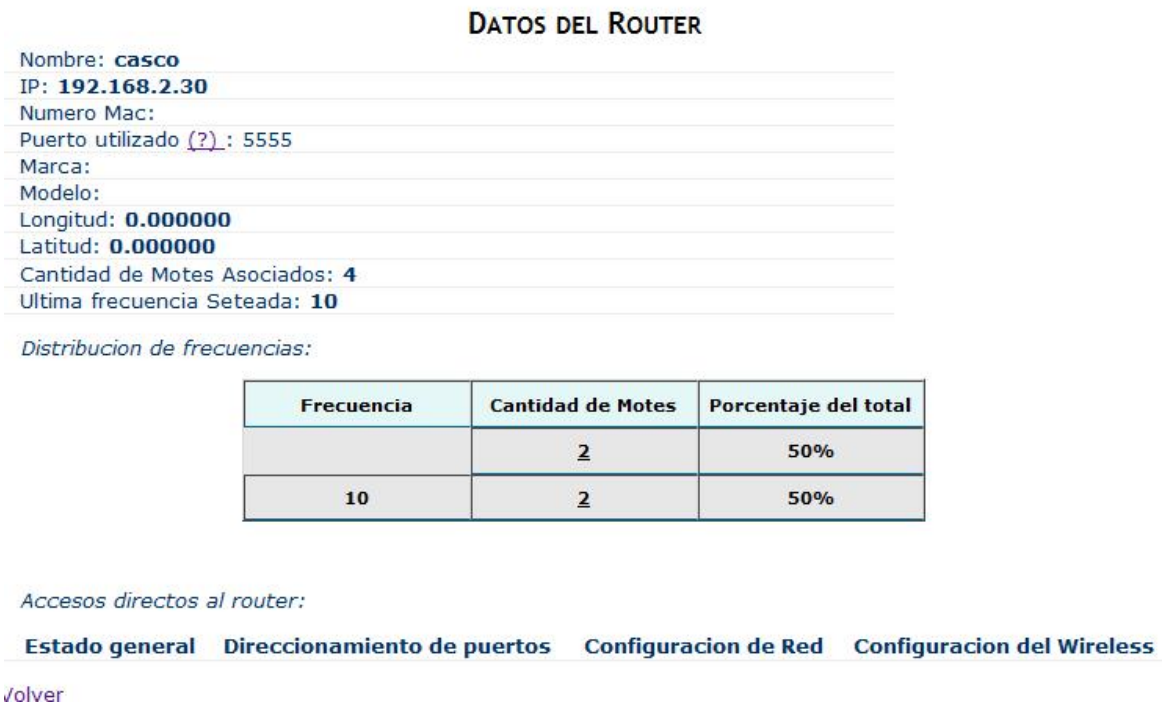

Figura D.4: Datos del Router

cuestión (figura D.9).

Sensores Los diferentes tipos de sensores que tienen configurados los motes se deben crear en la aplicación SiMSI para poder recibir datos correctamente.

Ingresando en Nuevo Sensor se crea un nuevo tipo de sensor que debe tener el *sensor_id* de la red de sensores y el *tipo de dato* asociado (figura D.10).

Para editar o borrar un sensor, se hace clic en *Sensores Existentes*, y se elige la opción deseada.

Frecuencia Aquí es donde el usuario podrá configurar la frecuencia de los motes del sistema. Para setear una nueva frecuencia debe ingresar a la opción Nueva Frecuencia, en el menú vertical del lado izquierdo. Se van a seleccionar los motes, por router al cual están asociados, y la frecuencia a setear (figura D.11).

En la opción *Distribución de Frecuencias*, se visualiza un cuadro que muestra, por router, la última frecuencia seteada, los motes asociados a cada router, y el porcentaje de motes confirmados a esa frecuencia. Se aprecia en la figura D.12.

Alarmas En esta sección se manejan las alarmas del sistema. Para configurar una nueva alarma se ingresa a la opción *Nueva Alarma*. Se deben completar los parámetros propios de la alarma, las zonas de su habilitación y los usuarios a notificar (por SMS o mail) (figura D.13).

En Datos de la Alarma se encuentra la opción de setear el límite inferior y/o el límite superior de la Nueva Alarma. La opción *Habilitar Alarma* está marcada por defecto.

Cada zona pertenece a un router y todos los motes que se encuentran conectados con el mismo.

Figura D.5: Configuración del Router

Figura D.6: Motes Nuevos

Para la selección de notificación, se deben chequear las casillas correspondientes a cada opción. Estas casillas aparecerán si los usuarios tienen configurado el teléfono móvil para la recepción de SMS, y/o la dirección de correo electrónico.

Para editar o eliminar las alarmas, hacer clic en *Alarmas Existentes* en el menú izquierdo y seleccionar la opción deseada.

Figura D.7: Editar Mote

Figura D.8: Elegir Motes

Figura D.9: Cuadro con el Estado de Motes

Figura D.10: Crear Nuevo Sensor

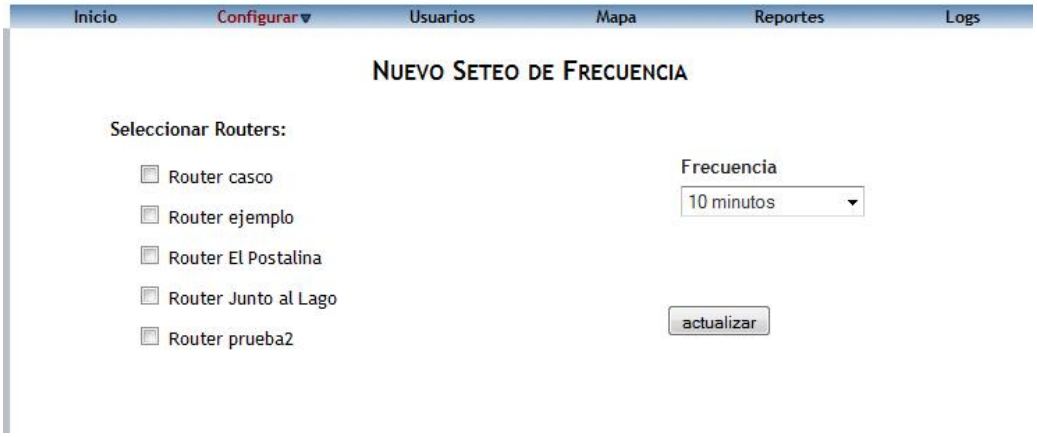

Figura D.11: Setear Nueva Frecuencia

Figura D.12: Distribución de Frecuencias

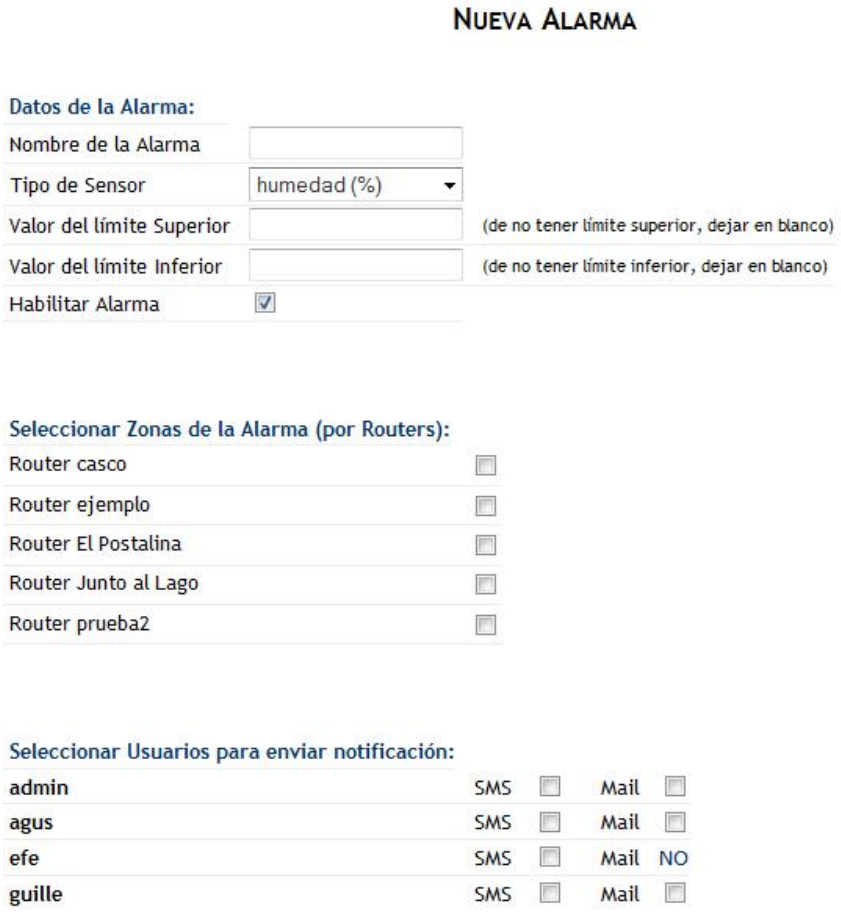

Figura D.13: Crear Nueva Alarma

Usuarios

En esta sección se administran los usuarios del sistema. Para crear un nuevo usuario se debe ingresar a la opción Nuevo Usuario. Se deberán completar los datos personales correspondientes y además se deberá definir qué tipo de usuario se está creando. Existen tres tipos: Básico, Administrador, Personalizado. El usuario Administrador tiene todos los privilegios, en tanto que el básico tiene privilegios reducidos.

La opción Personalizado permite seleccionar uno por uno los permisos que tiene el usuario.

Todas estas configuraciones se aprecian en la figura D.14.

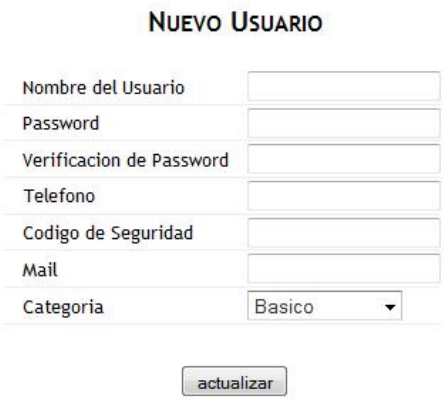

Figura D.14: Crear Nuevo Usuario

Siguiendo el formato de las secciones anteriores, para editar y/o borrar un usuario se debe acceder a la opción *Usuarios Existentes*, y elegir la opción deseada.

Mapa

Para utilizar la sección *Mapa* es necesario adquirir una clave que otorga *Google Maps*, provista al momento de adquirir la aplicación SiMSI.

Si no se tiene una clave, contactarse con el Administrador (simsi.group@gmail.com) para poder cargarla en la aplicación.

Con el mapa es posible observar toda la red de motes y routers.

Se puede visualizar información de los dispositivos desde el menú izquierdo o directamente haciendo clic en el ícono en el mismo mapa.

En el caso de un router se despliega su nombre y su dirección IP, además de la disponibilidad del mismo y si tiene activada la alarma Estado Enlace.

Para los motes se observan los últimos datos de sus sensores y las alarmas que tiene activadas.

Cuando un dispositivo se encuentra de color rojo en el mapa y en el menú izquierdo significa que tiene alarmas activadas (figura D.15).

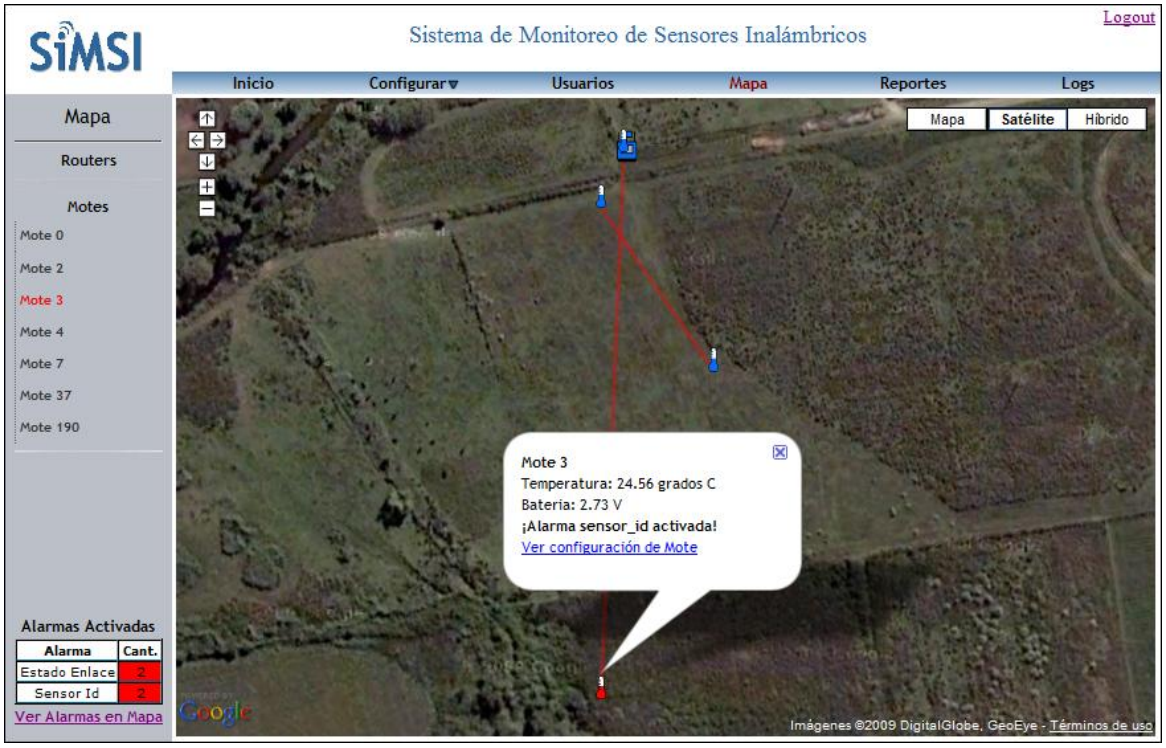

Figura D.15: Sección Mapa

NOTA: Para que los equipos se desplieguen en el mapa, es necesario que tengan completos los campos de *Longitud* y *Latitud* de la sección *Configurar*.

Reportes

En esta sección se pueden generar, y visualizar distintos reportes. Para generar un nuevo reporte se debe ingresar en la opción *Nuevo Reporte*, en el menú izquierdo.

Se debe seleccionar el tipo de dato, a través de una lista desplegable que incluye todos los tipos de datos que ya fueron configurados.

Se deben seleccionar los motes a monitorear, que se encuentran en un árbol desplegable agrupados por router.

Para elegir el período de monitoreo se disponen de varias opciones que toman como fecha base la fecha de visualización del reporte. Por ejemplo, si se selecciona la opción *Ultima Semana*, se van a procesar los datos que van desde el momento en que se quiere visualizar el reporte, hasta una semana atrás. El único caso que no cumple esta tendencia es la opción de *Mes corriente*, que toma los datos desde el día 1 del mes (actual) hasta el día en que se visualiza el reporte.

Para elegir el tipo de gráfico asociado al reporte se tienen tres opciones, de las cuales dos aluden a evoluciones temporales, ya sea promediando los motes dentro del router o no. Es decir, los tipos *Evolución temporal de motes* y *Evolución temporal de routers*, grafican los datos de los sensores correspondientes al tipo de dato elegido, en el período de tiempo seteado, pero se diferencian en que el primero grafica mote por mote, y en el segundo caso se grafica el promedio por router. La tercera opción permite graficar los datos, evaluando una hora particular del día, en el período de tiempo seteado.

Todas estas configuraciones se visualizan en la figura D.16.

Figura D.16: Crear Reporte

Para la tercera opción de tipo de gráfico se debe además seleccionar la hora del día a monitorear como se indica en la figura D.17.

Para graficar un reporte se debe acceder a la opción *Reportes Existentes*, y hacer clic en el nombre del reporte deseado. La figura D.18 muestra lo que se visualiza al graficar.

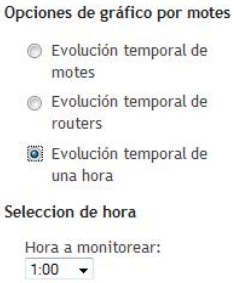

Figura D.17: Crear Reporte Evolución Temporal de Una Hora

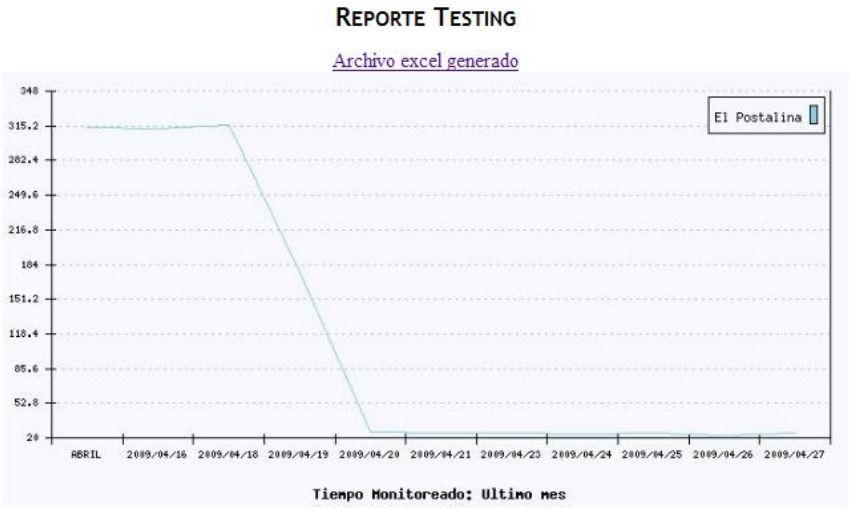

Figura D.18: Ejemplo de Gráfica de un Reporte

En esa misma figura, se aprecia que hay un link para descargar un archivo Excel generado con los datos graficados. Esto muchas veces puede ser útil para que el productor pueda llevar un registro propio del reporte deseado.

La opción *Ver Datos Puntuales* del menú izquierdo permite graficar directamente datos puntualmente elegidos, entre fechas seleccionadas. Es la misma idea de un reporte normal (misma selección de motes, datos, gráficos) a diferencia que no queda guardado. Sirve para ver algo estrictamente puntual (figura D.19).

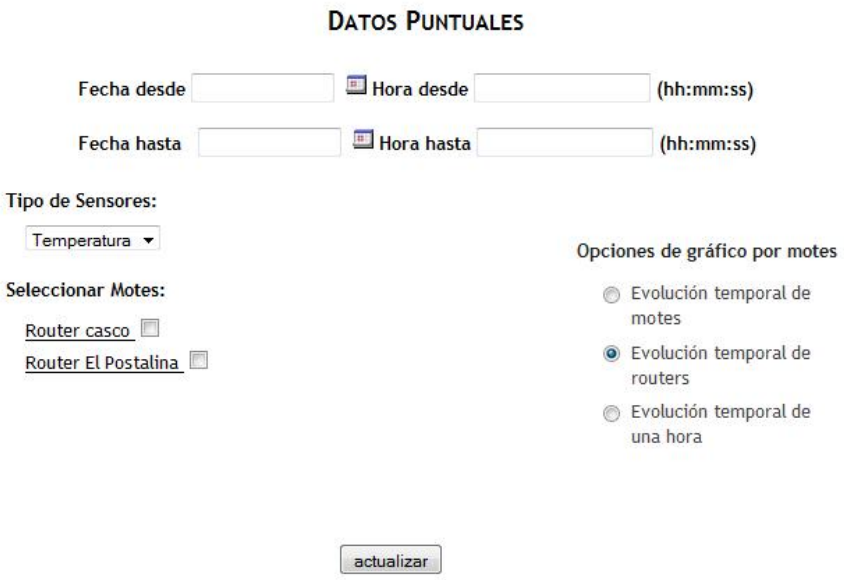

Figura D.19: Ver Datos Puntuales

Logs

En esta sección se pueden visualizar los logs del sistema.

En la opción *Acciones de Usuarios*, se visualizan las últimas 30 acciones realizadas por los usuarios en el sistema (figura D.20). Se puede filtrar por usuario a través de una lista desplegable.

ACCIONES DE USUARIO

Fecha	Usuario	Sección	Descripción
26/04/2009 16:22:22	agus	Alarmas	agus editó la Alarma Testing
26/04/2009 16:21:08	agus	mapa	agus visualizo el mapa
26/04/2009 16:20:41	admin	Logs	admin Ver Logs
26/04/2009 16:16:40	agus	Alarmas	agus editó la Alarma Testing
26/04/2009 16:07:59	agus	Frecuencia	agus seteo una nueva frecuencia
26/04/2009 16:07:20	admin	Ver reporte Testing	admin visualizo el reporte Testing
26/04/2009 15:56:21	admin	mapa	admin visualizo el mapa
26/04/2009 15:56:19	admin	log in	admin ingreso a Simsi
26/04/2009 15:54:44	admin	mapa	admin visualizo el mapa
26/04/2009 15:54:44	admin	mapa	admin visualizo el mapa
26/04/2009 15:48:29	admin	log in	admin ingreso a Simsi
26/04/2009 15:36:55	admin	log in	admin ingreso a Simsi
26/04/2009 15:26:46	agus	Motes	agus visualizo estado de motes
26/04/2009 15:26:30	agus	Frecuencia	agus visualizo distribuciones de frecuencia
26/04/2009 15:18:23	agus	Logs	agus Ver Logs
26/04/2009 15:10:55	agus	Frecuencia	agus seteo una nueva frecuencia

Figura D.20: Acciones de Usuarios

Haciendo clic en *Configuración de Equipos*, se visualizan las últimas modificaciones realizadas sobre todos los equipos del sistema (figura D.21). Se puede filtrar por equipo.

Finalmente, ingresando en la opción *Alarmas*, se pueden ver los últimos estados de las alarmas (figura D.22). Se puede filtrar por alarma.

Tabla de Alarmas

En toda la aplicación se puede observar la Tabla de Alarmas Activadas que brinda al usuario la cantidad de equipos que tienen alarmas activadas (figura D.23).

CONFIGURACIÓN DE EQUIPOS

Fecha	Equipo	Parámetro configurado	Valor anterior	Valor nuevo
26/04/2009 16:07:59	Router El Postalina	Frecuencia	7 minutos	1 minutos
26/04/2009 15:10:55	Router El Postalina	Frecuencia	7 minutos	1 minutos
26/04/2009 11:57:34	Router El Postalina	Puerto asignado	0	80
26/04/2009 11:31:52	Mote 4	Id de Mote	4	
26/04/2009 10:59:14	Router El Postalina	Frecuencia	7 minutos	10 minutos
24/04/2009 21:11:39	Router El Postalina	Frecuencia	3 minutos	20 minutos
23/04/2009 21:23:53	Router El Postalina	IP	192.168.2.34	192.168.2.33
23/04/2009 20:37:41	Router Junto al Lago	Puerto asignado	60010	
23/04/2009 20:37:41	Router Junto al Lago	IP	192.168.6.29	192.168.2.102
22/04/2009 19:47:21	Router El Postalina	Frecuencia	3 minutos	5 minutos
20/04/2009 18:23:17	Router ejemplo	Longitud	-63.410000	-65.410000
20/04/2009 15:51:19	Router prueba2	IP	10.24.123.122	192.168.10.124
15/04/2009 10:01:46	Mote	Latitud	-34.780000	-34.80000
15/04/2009 10:01:46	Mote 3	Id de Mote	3	
14/04/2009 22:58:43	Mote	Latitud		-34.3

Filtrar por posibles equipos: Todos

Figura D.21: Configuración de Equipos

ALARMAS

Fecha	Nombre de Alarma	Equipo	Estado
26/04/2009 16:23:56	Testing	Mote 3	Desactivada
26/04/2009 16:18:09	Testing	Mote 3	Activada
26/04/2009 12:59:34	Estado Enlace	Router El Postalina	Desactivada
26/04/2009 10:55:16	Estado Enlace	Router	Desactivada
26/04/2009 10:44:29	Estado Enlace	Router El Postalina	Desactivada
25/04/2009 23:45:57	Estado Enlace	Router Junto al Lago	Activada
24/04/2009 21:29:51	Estado Enlace	Router El Postalina	Desactivada
23/04/2009 22:19:45	Estado Enlace	Router	Activada
23/04/2009 21:46:41	Estado Enlace	Router	Activada
23/04/2009 21:41:45	Estado Enlace	Router	Activada
23/04/2009 21:24:14	Estado Enlace	Router	Desactivada
22/04/2009 21:41:17	Estado Enlace	Router El Postalina	Desactivada
22/04/2009 20:49:53	Testing	Mote 3	Desactivada
22/04/2009 19:56:44	Testing	Mote 3	Activada
20/04/2009 21:13:56	Sensor Id	SensorID 67	Activada
20/04/2009 18:58:04	Batería	Mote 3	Desactivada
20/04/2009 17:08:59	Estado Enlace	Router prueba2	Activada
19/04/2009 17:29:09	Testing	Mote 3	Activada
19/04/2009 13:30:27	Estado Enlace	Router El Postalina	Desactivada
19/04/2009 11:07:41	Testing	Mote 3	Desactivada

Figura D.22: Logs de Alarmas

Figura D.23: Tabla de Alarmas Activadas

Apéndice E

Contenido del CD

El CD SiMSI contiene:

- Documentación del Proyecto SiMSI: Sistema de Monitoreo de Sensores Inalámbricos (archivo *simsi.pdf*)
- Instalador de la aplicación web (archivo *setup SiMSI.exe*)
- Archivos de la aplicación web (carpeta *Aplicación SiMSI*)
- Archivos de Demonio programado en Java (carpeta *Aplicación SiMSI/demonio*)
- Documentación Javadoc del Demonio (carpeta *Aplicación SiMSI/demonio/javadoc*)
- Imagen del Firmware del Router diseñado por SiMSI (archivo *SIMSI_OKv2.bin*)
- Hoja de datos de los motes tmote sky (archivo *tmote sky.pdf*)
- Especificaciones del Router LinkSys WRT350N (archivo *Linksys wrt350n.pdf*)