TESIS DEFENDIDA POR

Edgardo Avilés López

Y aprobada por el siguiente comité:

Dr. José Antonio García Macías
Director del Comité

Dra. Ana Isabel Martínez García
Miembro del Comité

Dr. Luis Armando Villaseñor González
Miembro del Comité

Dr. Pedro Gilberto López Mariscal
Coordinador del Programa de Posgrado en Ciencias de la Computación

Dr. Edgar Gerardo Pavía López
Director de Estudios de Posgrado

10 de Noviembre de 2006

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE
EDUCACIÓN SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS
EN CIENCIAS DE LA COMPUTACIÓN

Arquitectura Orientada a Servicios para Redes Inalámbricas de Sensores

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

MAESTRO EN CIENCIAS

Presenta:

Edgardo Avilés López

Ensenada, Baja California, México. Noviembre de 2006.

RESUMEN de la tesis que presenta **Edgardo Avilés López**, como requisito parcial para obtener el grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN. Ensenada, Baja California. Noviembre de 2006.

Arquitectura Orientada a Servicios para Redes Inalámbricas de Sensores

Resumen aprobado por:

Dr. José Antonio García Macías
Director de Tesis

Las redes inalámbricas de sensores son un elemento importante para hacer realidad la visión del cómputo ubicuo, pues permiten la captura constante de parámetros como temperatura, aceleración, humedad, etc. Sin embargo, existen inhibidores para su amplia aplicación, ya que la brecha entre los requerimientos del usuario y la programación de la red es muy amplia. Este problema está siendo abordado desde distintos enfoques con varias soluciones disponibles. No obstante, muchas de ellas no son flexibles, heterogéneas, escalables, reutilizables o sencillas de utilizar. En esta tesis se explora la utilización del paradigma orientado a servicios en el diseño de una arquitectura para sistemas *middleware* de redes inalámbricas de sensores.

Con el objetivo de comprobar la viabilidad en la aplicación de la arquitectura propuesta, se realizó el diseño e implementación de un sistema *middleware* prototipo denominado TinySOA. Asimismo, se realizó una evaluación funcional de la arquitectura, en la que se contrastan sus requerimientos con la solución propuesta. Como parte de la misma, se efectuó un experimento de evaluación utilizando TinySOA con los usuarios potenciales. Esto permitió conocer su percepción en cuanto a utilidad, facilidad e intención de uso. Este trabajo contribuye a la creación de nuevos mecanismos que permitan cerrar cada vez más la brecha entre necesidades de aplicación y lógica de bajo nivel.

Palabras clave: Redes Inalámbricas de Sensores, Arquitecturas Orientadas a Servicios, Servicios Web, Internet, *Middleware*.

ABSTRACT of the thesis presented by **Edgardo Avilés López**, as a partial requirement to obtain the MASTER SCIENCE degree in COMPUTER SCIENCE. Ensenada, Baja California. November 2006.

Service-Oriented Architecture for Wireless Sensor Networks

Abstract approved by:

Dr. José Antonio García Macías
Thesis director

Wireless sensor networks are a key element to make ubiquitous computing a reality, as they allow the constant capture of parameters like temperature, acceleration, humidity, and others. However, there are inhibitors to their broad application since the gap between user requirements and network programming is too wide. This problem is being handled through different approaches and some solutions have been proposed. However, many of them are not flexible, heterogeneous, scalable, reusable or simple to use. In this thesis, we present the exploration of a service-oriented paradigm approach in the design of a middleware system architecture for wireless sensor networks.

In order to verify the feasibility of application of the proposed architecture, a middleware system prototype called TinySOA was designed and implemented. Also, a functional evaluation of the architecture was performed, contrasting the design requirements with the proposed solution. As a part of this, an evaluation experiment with the potential users of TinySOA was conducted. This allowed to know their perception on topics like utility, easiness and intention of use. This work contributes to the creation of new mechanisms for gradually closing the gap between application needs and low level logic.

Keywords: Wireless Sensor Networks, Service-Oriented Architectures, Web Services, Internet, Middleware.

Dedicatoria

A Gianni Malenito

«*Et Eärello Endorenna utúlien. Sinome maruvan ar Hildinyar teen' Ambar-metta!*»

– John Ronald Reuel Tolkien (1892-1973)

Agradecimientos

A mi asesor, Dr. J. Antonio García Macías,
por su atención, consejos y apoyo.
Sobre todo por su confianza.

A los miembros de mi comité, Dra. Ana Isabel Martínez García y Dr. Luis Armando Villaseñor González, por su tiempo e invaluables observaciones.

A mi familia, mis padres y mis hermanos,
por ser lo que son, por su amor y por alentarme en todo momento.

A mis amigos en el otrora Laboratorio de Cómputo Colaborativo y en la HilloBase, por los ratos de ocio, sus consejos y apoyo incondicional.

A mis amigos en Mocorito, Sinaloa.
Por su apoyo y compañía a distancia.

A los compañeros de la generación 2006, por su participación en la evaluación de este trabajo.

Al Centro de Investigación Científica y de Educación Superior de la ciudad de Ensenada, Baja California.

Al Consejo Nacional de Ciencia y Tecnología,
por su apoyo económico sin el cual no hubiera sido posible este trabajo de investigación.

Ensenada, México
10 de Noviembre de 2006.

Edgardo Avilés López

Tabla de Contenido

Capítulo Página

Tabla de Contenido (Continuación)

Capítulo Página

Lista de Figuras

Lista de Figuras (Continuación)

Lista de Tablas

Capítulo I

Introducción

«*Yo nunca pienso en el futuro. Viene demasiado pronto*»
– Albert Einstein (1879-1955)

El cómputo ubicuo tiene como objetivo el crear un ambiente en el existan dispositivos embebidos de tal manera que las capacidades de cómputo y comunicaciones se encuentren disponibles de manera discreta y constante. Para hacer realidad esta visión, una de las tecnologías necesarias son las redes de sensores que capturan fenómenos tales como temperatura, aceleración, humedad, ubicación geográfica, etc. Estos sensores tienen una capacidad de cómputo mínimo y se comunican entre ellos, componiendo una red inalámbrica de sensores (WSN por sus siglas en inglés).

I.1. Planteamiento del problema

Actualmente, se ha presentado un gran desarrollo en el área de WSN, tanto en *hardware* como en *software*, sin embargo, aún existe una gran problemática en su aplicación ya que la brecha entre los requerimientos del usuario y la programación de la red es muy ancha; esto debido principalmente a que la programación de la red es muy dependiente del *hardware* sobre el que se implementa, utilizando lenguajes de bajo nivel para lograr comunicar instrucciones al *hardware*. Este problema está siendo abordado por distintas investigaciones con un buen número de soluciones disponibles. Aunque la mayoría están muy enfocadas a un problema en específico, causando que la solución no pueda ser aplicada en otras situaciones. No obstante de la gran cantidad de soluciones, muchas de ellas

no son flexibles, heterogéneas, reutilizables o simplemente sencillas de utilizar.

I.2. Objetivo general

El objetivo principal de este trabajo es diseñar una arquitectura para el desarrollo de un *middleware* prototipo para WSN utilizando el enfoque de servicios.

I.3. Objetivos específicos

- Explorar la utilización de la arquitectura orientada a servicios en la lógica interna y externa de la red de sensores.
- Proponer una arquitectura cuidadosamente enfocada a las capacidades limitadas actuales de los sensores.
- Proveer una abstracción de alto nivel para facilitar la creación de herramientas de monitoreo/visualización y para hacer transparente el control interno de la red.
- Analizar los alcances y limitaciones del trabajo.

I.4. Metodología

La realización de este trabajo se llevó a cabo en varias etapas. Para iniciar se realizó una revisión de literatura en el área general de redes inalámbricas de sensores con la finalidad de comprender sus características únicas y específicas, para posteriormente realizar la revisión de literatura sobre soluciones de extracción de datos para observar los problemas que se han presentado y analizar en qué medida una arquitectura orientada a servicios podría ser útil. Durante esta etapa, analizando diferentes aplicaciones, se lograron comprender las necesidades de los programadores, así como también los requerimientos que debe cumplir un sistema *middleware* para redes de sensores.

Para lograr los objetivos de este trabajo se utilizó la metodología de espiral. Se seleccionó esta metodología debido a que permite ir depurando el diseño continuamente, logrando resultados cada vez más completos. Esta metodología consta de cuatro etapas: planeación, análisis, desarrollo y evaluación, las cuales se van desarrollando continuamente hasta la liberación del producto final. En la etapa final se procedió al diseño e implementación de un sistema *middleware* prototipo basado en la arquitectura propuesta y al desarrollo de una aplicación de monitoreo y visualización en base a los servicios web proveídos por el sistema. Todo esto con el objetivo de comprobar la viabilidad de aplicación de la arquitectura, lo cual, nos permitió conocer eventualidades en su concepción y obtener información que nos ayudó a mejorar el diseño.

I.5. Contenido de la tesis

Este documento se encuentra organizado de la siguiente manera: en el capítulo II se hace una descripción detallada del área de redes inalámbricas de sensores, de algunas de las plataformas de *hardware* disponibles actualmente, así como también de sus aplicaciones, importancia y problemática actual, para terminar con algunas líneas abiertas de investigación; en el capítulo III, se habla de algunos sistemas *middleware* actuales, sus características y su clasificación, además, se analizan los requerimientos de este tipo de sistemas; en el capítulo IV, se describen las arquitecturas orientadas a servicios y se mencionan algunos trabajos anteriores en cuanto al desarrollo de sistemas *middleware* para redes de sensores utilizando un enfoque orientado a servicios; en el capítulo V se introduce la arquitectura propuesta, en donde se describen sus componentes y se presenta su análisis y diseño; en el capítulo VI se describe el diseño de un sistema con base en la arquitectura propuesta con el propósito de hacer una evaluación de la utilidad real de la misma, cuyos resultados también son presentados en el capítulo; en el capítulo VII se presenta la evaluación de este trabajo y en el capítulo final (VIII) se concluye mencionando las aportaciones así como también el trabajo a futuro.

Capítulo II

Redes Inalámbricas de Sensores

«*Las tecnologías más profundas son aquellas que desaparecen.*
Se tejen en la vida diaria hasta que son indistinguibles de ella»
– Mark Weiser (1952-1999)

Las redes inalámbricas de sensores están emergiendo como un nuevo elemento en el ecosistema de la tecnología de información y como un rico dominio de activa investigación involucrando *hardware* y diseño de sistemas, redes, algoritmos distribuidos, modelos de programación, administración de datos, seguridad, y factores sociales. Están empezando a hacer realidad la visión de una Internet embebida, en la cual redes de dispositivos de cómputo interconectados ampliamente inmersas en el ambiente físico transforman amplios campos de ciencia, ingeniería, y manufactura al proveer instrumentación detallada de muchos puntos sobre espacios amplios, naturales y artificiales. Nos proveen de un nuevo tipo de instrumento llamado macroscopio, que nos permite observar los fenómenos físicos a un nivel de fidelidad nunca antes obtenido (Culler y Hong, 2004).

Gracias a los avances y reducción de costos en dispositivos electrónicos y de comunicación inalámbrica, es posible construir nodos de sensado multifuncionales y multipropósito de bajo costo que operan con poca energía, de un tamaño pequeño, y con capacidad de comunicación a corta distancia. Estos sensores constan de una unidad de procesamiento con un poder de cómputo mínimo, una pequeña capacidad de memoria, una unidad de comunicación inalámbrica y uno o varios dispositivos de sensado que capturan parámetros tales como como temperatura, aceleración, humedad, etc. Estos nodos llevan a lo que

conocemos como una red inalámbrica de sensores.

Una red de sensores está compuesta por cientos y de ser necesario miles de nodos que se encuentran esparcidos en un área. La distribución de los mismos puede ser aleatoria o planeada, lo cual permite su uso en prácticamente cualquier ambiente físico. Esta característica es proveída por un conjunto de protocolos de redes de sensores y algoritmos los cuales permiten que la red se autoorganice. Otro de los beneficios que caracteriza a una red de sensores es que sus nodos pueden trabajar de modo cooperativo, por ejemplo procesando parcialmente la información capturada antes de comunicarla a la red.

II.1. Conceptos clave

A continuación se definen algunos términos y conceptos clave (Zhao y Guibas, 2004) que serán utilizados en este trabajo:

- *Sensor:* Un transductor que convierte un fenómeno físico tal como calor, luz, sonido o movimiento en señales eléctricas u otras que pueden ser manipuladas por otros aparatos.
- *Nodo de sensado:* Una unidad básica en una red de sensores, con sensores embebidos o intercambiables, procesador, memoria, comunicación inalámbrica, y fuente de poder. Es a menudo abreviado como *nodo*.
- *Nodo gateway:* Un nodo de sensado común conectado hacia el exterior que funge como puente entre el funcionamiento interno de la red y la unidad de control externa. Más comúnmente conocido como *nodo sink* o sumidero.

II.2. Funcionamiento

En la Figura 1 se puede observar un escenario típico de una red de sensores. Su funcionamiento inicia al desplegar los nodos en el área que se desea estudiar, al hacer esto, se

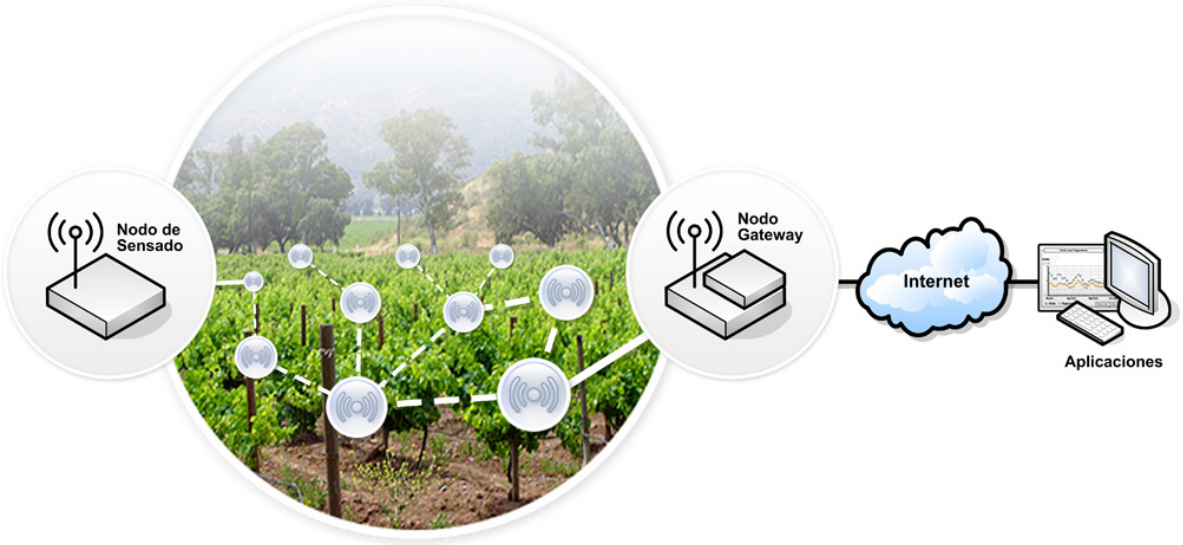


Figura 1: **Escenario común de una red inalámbrica de sensores**

establece una asociación con los elementos a ser estudiados (plantas, criaturas, ambientes, etc.). Esta asociación puede ser una-a-una o bien destinar varios sensores a un solo objeto y viceversa (Römer et al., 2002). Una vez que se han distribuido una cantidad suficiente de sensores, la red puede iniciar la tarea para la que fue programada. Las redes de sensores están diseñadas para funcionar de manera desatendida a no ser que necesiten de mantenimiento, como puede ser reemplazar algún sensor dañado o algún nodo sin energía.

Los resultados de la tarea son enviados a una entidad externa conectada a la red (por ejemplo una aplicación a través de Internet) a través de uno de los nodos de la red que funge como puente; a este nodo se le conoce como *nodo gateway* o más comúnmente *nodo sink*. Esta comunicación es bidireccional, de esta manera la entidad externa puede inyectar tareas a la red de sensores. Idealmente, estas tareas le son indicadas a un alto nivel, es decir, mediante el uso de expresiones como: «reportar temperatura promedio en el área». Adicionalmente los nodos pueden trabajar de manera cooperativa para, procesando una pequeña parte de la tarea, fusionar las lecturas y obtener el resultado final.

II.3. Diferencias con las redes tradicionales

Podría parecer que muchos de los protocolos, algoritmos y mecanismos con los que se cuenta actualmente para las redes inalámbricas ad hoc podrían aplicarse con éxito en las redes de sensores, sin embargo, existen muchas características propias de los sensores a las que no se podría dar soporte. Muchos investigadores están actualmente comprometidos en desarrollar esquemas para cumplir estos requerimientos.

Algunas de estas características únicas son (Akyildiz et al., 2002):

- El número de nodos en una red inalámbrica de sensores puede ser de una magnitud muchas veces más alta que la de los nodos en una red ad hoc.
- Los nodos están densamente distribuidos.
- Las redes inalámbricas de sensores son tolerantes a fallas.
- La topología de una red de sensores cambia muy frecuentemente.
- Los nodos están limitados en energía, capacidades de cómputo, y memoria.
- Heterogeneidad. Los nodos sensores pueden variar en fabricante, capacidades de cómputo y memoria, interfaces de comunicación, etc.
- Calidad de servicio. Podrían existir aplicaciones en las cuales la recepción de eventos capturados por los sensores sea crítica.

II.4. Plataformas de *hardware*

El principal componente de *hardware* de una red de sensores es conocido como *mote* (mota en español, en referencia a una partícula de polvo), el cual tiene tres componentes principales: un microprocesador, un sistema microelectromecánico y dispositivos de comunicación. El primer modelo de este componente fue diseñado en el proyecto Smart-Dust (Figura 2). Fue a partir de este modelo que empezaron a surgir plataformas, cada vez

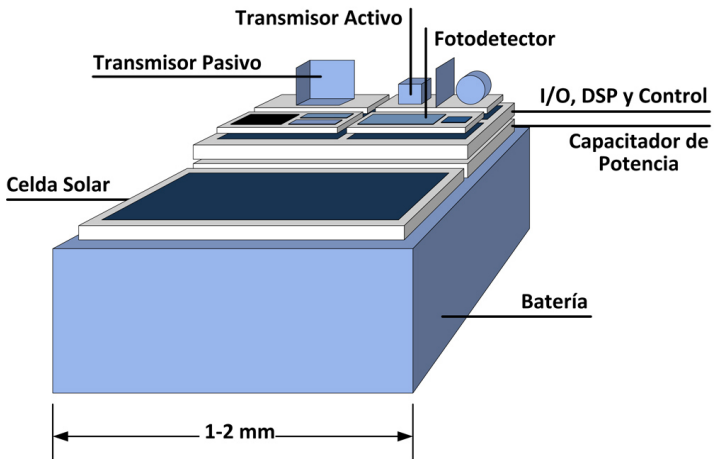


Figura 2: **Modelo de un nodo de sensado SmartDust**

con mayores capacidades de memoria, procesamiento y comunicación. En la Figura 3 se puede observar su evolución y las características principales de cada una de ellas. Uno de los primeros desarrollos fue la plataforma weC (Hollar, 2000), desarrollada en 1999. La comunicación de este *mote* se realizaba a través de radio frecuencia a una velocidad de 10 Kbps, incluía sensores de temperatura e intensidad de luz, con tan solo 5 KB de memoria RAM y 8 KB de espacio para programas. Su mayor aportación fue la incorporación de la programación «al vuelo» la cual permitía reprogramar los nodos en tiempo de ejecución.

En el año 2000 surge la plataforma René y poco más adelante René 2. Ambos primeros desarrollos por parte de la compañía Crossbow[1]. Su principal aportación es el diseño modular del *hardware* de sensado del nodo, lo cual permite una gran flexibilidad al poder intercambiar los dispositivos de sensado del nodo. Una evolución directa de esta plataforma es Dot, con características similares pero de un tamaño mucho más reducido.

Un nuevo paso en la evolución de las plataformas surgiría a principios del 2002 con la introducción de la plataforma Mica por parte de la misma compañía. El tamaño de memoria se vio aumentado drásticamente, así como la velocidad de transmisión. Una nueva

[1] Crossbow Technology Inc. `http://www.xbow.com/`

	 weC 1999	 René 2 2000	 Dot 2001	 Mica 2002	 Mica2 2002	 MicaZ 2003	 TelosB 2005
Memoria para Programas (KB)	8	16	16	128	128	128	48
Memoria RAM (KB)	0.5	1	1	4	4	4	10
Active Power (mW)	24	24	24	27	89	89	41
Sleep Power (µW)	45	45	45	75	< 75	< 75	15
Wakeup Time (µs)	1000	36	36	180	180	180	6
Memoria Externa (KB)	32	32	32	512	512	512	1000
Velocidad de Transmisión (Kbps)	10	10	10	40	38.5	250	250

Figura 3: **Evolución de las plataformas de *hardware***

mejora de la plataforma, Mica2, surgiría más adelante. Esta nueva plataforma presentaba mejoras en cuanto a la transmisión y recepción de datos. MicaZ la sucedería a su vez, en ella se incorporan protocolos de comunicación basados en la norma IEEE 802.15.4[2], con lo cual se aumentaba la velocidad de transmisión a 250 Kbps.

Una de las más recientes plataformas en surgir sería Telos, cuya principal característica es el ahorro del consumo de energía. Esta última plataforma ofrece nuevas características en cuanto a la programación ya que incluye una interfaz integrada para la programación y comunicación. De esta plataforma surgen TelosB y Tmote Sky[3].

II.5. Aplicaciones

Las posibilidades de aplicación de las redes inalámbricas de sensores ofrecen, son amplias. Algunos ejemplos son (Römer y Mattern, 2004):

- *Monitoreo de glaciales.* Se observan y estudian zonas glaciales con el objetivo de realizar un estudio del clima del planeta y de las dinámicas dentro del glacial.

[2]IEEE 802.15 Working Group for WPAN. `http://www.ieee802.org/15/`
[3]Moteiv Corporation. `http://www.moteiv.com/`

- *Monitoreo de aguas oceánicas.* Se observan la temperatura, salinidad y perfil actual de la superficie oceánica para obtener patrones de variación de clima.

- *Rescate de víctimas de avalanchas.* Se provee una automatización en la prioridad del rescate de víctimas. Al conocer sus estados vitales (pulso, respiración, presión sanguínea, etc.) se pueden identificar a aquellas víctimas que necesitan una atención más urgente que las demás, con lo que se puede salvar vidas al tener un mejor control de los recursos y manejo de la emergencia.

- *Localización de francotiradores.* Se localiza a los francotiradores y la trayectoria de disparo al proveer pistas valiosas sobre las mismas.

- *Observación de aves.* Se hace uso de una red inalámbrica de sensores para estudiar el comportamiento de aves, las cuales son fácilmente perturbadas por la presencia humana.

- *Monitoreo de un hábitat.* Con el propósito de recabar datos de las condiciones y la escala de organismos individuales para ayudar a resolver problemas que afectan animales, plantas y personas (Szewczyk et al., 2004).

- *Monitoreo de cultivos.* Se monitorean las condiciones que influyen en el crecimiento de las plantas (temperatura, humedad de la tierra, luz, etc.).

II.6. Importancia

¿Por qué es importante la investigación en redes inalámbricas de sensores?. Como se puede observar, las posibilidades de aplicación son tantas como se puedan imaginar, incluso éstas pueden servir como medios para lograr realizar investigación en muchas más áreas (por ejemplo el caso del estudio de los glaciales, superficie oceánica y comportamiento de animales), así como también el de ayudar a hacer realidad el cómputo ubicuo.

La situación del área de redes de sensores es comparable a la situación de Internet hace treinta años. El campo es altamente específico a la aplicación, los límites y requerimientos de las aplicaciones no están completamente comprendidos y como consecuencia, muchas de las aplicaciones actuales aún no están listas para el mundo real (Estrin et al., 2002).

II.7. Problemática actual

Programar sensores involucra el uso de lenguajes de programación de bajo nivel para comunicar las instrucciones al *hardware*. Por lo que existe una fuerte necesidad de abstracciones de programación que simplifiquen el asignar tareas a los sensores y de *middleware* que soporte tales abstracciones (Römer, 2004).

El *middleware* es *software* que funge como comunicador entre interfaces de programación de diferente nivel de abstracción, permitiendoles intercambiar datos (puede ser visto como una especie de traductor). Esto nos lleva a una de las problemáticas abordadas en este documento, la cual es la de proveer un paradigma de programación que sea fácil de utilizar por un programador, es decir, que oculte lo mejor posible los mecanismos de *hardware* y de distribución, sin perder la eficiencia de implementación para una amplia variedad de aplicaciones. Otra de las problemáticas es la de la recuperación de datos por medio de mecanismos diseñados para apoyar las características propias de las redes inalámbricas de sensores.

Ambos son problemas actuales de investigación de gran importancia, el facilitar la programación de las redes de sensores ayuda a que sea más sencilla su adopción y el contar con un buen sistema de recuperación de datos es vital para cualquier aplicación.

II.8. Líneas abiertas de investigación

Existen muchas características en las que aún hay mucho trabajo por delante. Incluso muchas de las tecnologías con las que hoy se cuenta podrían ser mejoradas. Algunas de las líneas abiertas de investigación son las siguientes (Perrig et al., 2004):

- El diseño de *hardware* más pequeño, de bajo consumo de energía, y de bajo costo. Más unidades receptoras, de sensado y de procesamiento, necesitan ser diseñadas. El desarrollo de nuevas estrategias de administración eficiente de reservas de energía también es esencial.
- Modalidades de ahorro de energía de operación. Para prolongar el tiempo de vida de una red, un nodo sensor debe entrar en periodos de reducida actividad cuando sus baterías estén casi agotadas.
- Aunque existen algunos protocolos para la intercomunicación en las redes inalámbricas de sensores, éstos necesitan ser mejorados o nuevos protocolos deben ser desarrollados para solucionar mayores cambios en topología y mayor escalabilidad.
- Las estrategias de seguridad en las redes inalámbricas de sensores son necesarias para asegurar la integridad de la información y la protección contra ataques (captura de nodos, negación de servicios, etc.).
- Mayor soporte para la calidad de servicio. Para brindar apoyo a aplicaciones en las cuales la recepción de eventos capturados por los sensores sea crítica.
- Proporcionar mayor heterogeneidad de dispositivos.
- Dar soporte y coordinar aplicaciones concurrentes.
- Mayor desarrollo en los sistemas *middleware*.

Existen muchas más problemáticas, algunas de ellas relacionadas con la construcción de abstracciones de más bajo nivel y de diseño de dispositivos (Hill et al., 2004).

Kay Römer clasifica las investigaciones en el área en seis categorías principales:

- *Aplicaciones.*
- *Equipamiento.* Consta de temas como energía, miniaturización, sensores, convertidores analítico-digitales, transmisores de radio, entre otros.
- *Redes.* Incluye trabajos en la definición de protocolos y distribución de datos.
- *Procesamiento de datos.* Incluye temas tales como la propagación de consultas, obtención de información, compresión de datos, entre otros.
- *Infraestructura.* Temas que conforman el soporte para aplicaciones, programación distribuida, servicios e integración con Internet.
- *Sistemas Operativos.*

Capítulo III

Middleware para WSN

«*La alegría de ver y entender es el más perfecto don de la naturaleza*»
– Albert Einstein (1879-1955)

Se conoce como *middleware* a aquél *software* que sirve como «pegamento» entre dos aplicaciones de *software* (Bernstein, 1996). Puede ser visto como una especie de traductor o mediador y a veces, es llamado «tubería». Típicamente estos sistemas proveen servicios de mensajería los cuales son utilizados por distintas aplicaciones para lograr comunicarse entre ellas. Los sistemas *middleware* proveen un conjunto API con el principal propósito de hacer transparente el mecanismo de comunicación para el cual está diseñado. Generalmente (aunque no necesariamente) actúa como intermediario entre dos APIs de sistema, una de alto nivel de abstracción y otra de bajo nivel. Una API (del inglés Application Programming Interface 'Interfaz de Programación de Aplicaciones') es un conjunto de especificaciones de comunicación entre componentes de *software* cuyo propósito principal es el de proporcionar un conjunto de funciones de uso general (Hines, 1996). De esta manera, los programadores pueden acceder a la funcionalidad proveída sin necesitar de un entendimiento detallado del trabajo interno de las mismas. El uso de este tipo de sistemas acelera el desarrollo de nuevas aplicaciones, dejando a los desarrolladores la tarea de diseñar los componentes específicos a la aplicación. En los sistemas distribuidos, las tecnologías *middleware* liberan a los desarrolladores de aplicaciones de problemas relacionados a la distribución, heterogeneidad, escalabilidad, recursos compartidos, entre otros. El *middleware* provee un alto nivel de abstracción, escondiendo la complejidad introducida por la distribución. En otras palabras, la distribución se vuelve transparente.

En este capítulo se describen algunos de los requerimientos de las aplicaciones WSN, se enumeran los retos a superar en el diseño de un sistema *middleware* para WSN y se finaliza con una clasificación de los mismos.

III.1. Requerimientos de las aplicaciones

Para conocer los requerimientos que un sistema *middleware* debe cumplir hay que analizar primero los requerimientos de las aplicaciones a las que se dará soporte.

No obstante de la gran diversidad de aplicaciones WSN, existen dos aplicaciones que pueden actuar como ejemplos canónicos para el estudio de aplicaciones con gran movilidad y aplicaciones estáticas: CarTALK 2000 (Reichardt et al., 2002) y Sustainable Bridges[1].

- *CarTALK 2000*. El objetivo de este proyecto es el desarrollo de un sistema cooperativo de asistencia para automovilistas que provee un sistema ad hoc de advertencias de embotellamientos, accidentes y cruces de carril o carretera. En cada automóvil se encuentra integrado un sensor y estos cooperan con los demás para detectar y transmitir las advertencias. Adicionalmente se puede obtener información tal como la velocidad promedio, las condiciones del camino y la posición del automóvil a través de una interfaz de consulta estándar.

- *Sustainable Bridges*. El objetivo de este proyecto es el de proveer un monitoreo a un bajo costo de las condiciones de puentes utilizando sensores en posiciones estáticas del mismo con el objetivo de detectar defectos estructurales. Un amplio rango de datos tales como temperatura, humedad, vibración, detección de ruido y mecanismos de localización son utilizados para lograr la detección. Al encontrarse un problema, la red localiza el origen del fenómeno utilizando métodos de triangulación, al finalizar, esta informa a una entidad externa el problema encontrado y su localización para el posterior mantenimiento.

[1] Sustainable Bridges. `http://www.sustainablebridges.net/`

Propiedad	Sustainable Bridges	CarTALK 2000
Modelo de Datos	Específico	Genérico/flexible
Modelo de Consulta	Basado en *push*	Basado en *pull*
Paradigma de Progr.	*Publish/Subscribe*	Genérico basado en consultas
Trans. de Distribución	○	●
Energía	●	◔
Movilidad	◔	●
Sincronización	◕	◔
Topología	●	◑

○ No es importante ◑ Importancia media ● Muy importante

Figura 4: **Diferencias de requerimientos de dos aplicaciones (Marrón et al., 2005)**

En la Figura 4 se pueden observar los requerimientos de estas dos aplicaciones, los cuales presentan diferencias importantes. En lo referente al modelo de datos, debido a que Sustainable Bridges tiene un objetivo claro y específico este puede utilizar un modelo de datos en específico, mientras que CarTALK debe utilizar uno genérico debido a la gran heterogeneidad de dispositivos y usuarios. En lo referente al modelo de consulta, como Sustainable Bridges debe advertir sobre eventos detectados este funciona mejor con el modelo *push*, es decir, a través de un modelo *Publish/Subscribe* de tal manera que la información sea «empujada» a aquellos que se hayan suscrito al servicio, mientras que en CarTALK el modelo que funciona mejor es *pull*, de esta manera la información está disponible a través de consultas. En cuanto al requerimiento de transparencia de la distribución, Sustainable Bridges necesita saber qué nodo detectó algún evento por lo que la transparencia no es necesaria, caso contrario a CarTALK. La energía es fundamental en Sustainable Bridges ya que uno de sus requerimientos funcionales es el de extender la utilidad de la aplicación lo más posible mientras que en CarTALK no tiene mucha importancia. La movilidad es un requerimiento fundamental en CarTALK debido al escenario de la aplicación. En Sustainable Bridges la red debe estar sincronizada ya que un evento es detectado por información recolectada por diversos nodos. Y finalmente, para Sustainable Bridges la topología es importante ya que los nodos deben ser posicionados en lugares en los que se tenga una mayor

probabilidad de problemas tales como las columnas del puente.

Marrón *et al.* (2005) mencionan que a pesar de las diferencias, estas dos aplicaciones de ejemplo tienen algunas similitudes y que por lo tanto, es posible simplificar el desarrollo de ambas aplicaciones y de otras que compartan algunas propiedades con ellas. Propiedades que muchas de las aplicaciones desarrolladas actualmente parecen compartir al menos en parte. Estos mismos autores mencionan que muchas de las aplicaciones WSN:

- *Son data-centric y/o data-driven.* Es decir, las operaciones son realizadas haciendo referencia a la información producida por los nodos, en lugar de referirse al número de identificación o dirección de un cierto nodo. Esto brinda muchos beneficios, uno de ellos es que permite una mayor escalabilidad de la red.

- *Están basadas en estados.* Esto debido a que las necesidades pueden cambiar dependiendo del estado actual de la aplicación. Por ejemplo, en la aplicación Sustainable Bridges, la cual, tiene un estado de monitoreo en el que lo más importante es detectar la ocurrencia de un evento y notificar a otros nodos lo más rápido posible. Una vez que se recolectó información suficiente, esta cambia al estado de análisis en el cual se intercambia y analiza la información recolectada.

- *Deben ser tolerantes a fallas con respecto a fallas del mismo sistema y/o condiciones del ambiente.* Esto debido principalmente a que las WSN están diseñadas para funcionar por largos periodos de tiempo sin ningún tipo de atención, a no ser que sea necesario algún tipo de mantenimiento, tal como reemplazar algún sensor dañado, intercambiar algún nodo sin energía, etc.

- *Requieren una alta disponibilidad de los sensores y nodos.* Los sensores y los nodos deben ser confiables y estar disponibles para cualquier lectura o instrucción. Por ejemplo, en las dos aplicaciones anteriores, las lecturas deben ser confiables y los nodos estar disponibles para cualquier solicitud.

- *Deben ser flexibles o reconfigurables.* Debido a que las plataformas de WSN evolucionan continuamente las aplicaciones deben estar listas para adaptarse a estos cambios. Una manera de lograr esto es proveer mecanismos que permitan parametrizar componentes genéricos, de esta manera se pueden cumplir los requerimientos específico de la aplicación. Además las aplicaciones deben reaccionar a cambios en el ambiente y tener diferentes parámetros de optimización.

III.2. Retos de un sistema *middleware*

En el capítulo anterior se menciona que muchas de las soluciones de bajo nivel correspondientes a las redes ad hoc no pueden ser directamente aplicadas con éxito en las redes de sensores. Esto pasa también con los sistemas *middleware* para sistemas distribuidos actuales. Las redes WSN presentan requerimientos propios para los que estos no están preparados. Existen diversos problemas a resolver en el diseño de un sistema *middleware* para WSN. Actualmente, las aplicaciones y plataformas de *hardware* para WSN son altamente heterogéneas y de una complejidad algorítmica muy variada. Por lo que se desea encontrar una abstracción que pueda ser aplicada de manera similar para muchos tipos diferentes de aplicaciones y plataformas, en donde se evite la redundancia y se favorezca la reimplementación de código. Algunos autores han encontrado los siguientes requerimientos para el diseño de un sistema *middleware* para WSN (Römer, 2004, Marrón, 2005):

- *Abstracción.* El sistema *middleware* debe permitir la especificación de las tareas a un alto nivel, es decir, mediante el uso de expresiones como «reportar la temperatura promedio» o «reportar humedad cada cinco segundos». También debe esconder en lo más posible los detalles de *hardware* y de distribución al programador. Además, debe evitar hacer referencia a los nodos individuales de la red, y en su lugar, proveer una vista holística de la red. Esto podría ser posible mediante el uso de modelos tales como *Publish-Subscribe*, *data-centric* o basarse en eventos.

- *Eficiencia*. Los recursos de los sensores son muy limitados, por lo que los componentes del sistema *middleware* deben ser amigables con los recursos. Además, se debe prestar especial cuidado en el consumo de energía, ya que es el recurso más consumido y valioso del nodo al afectar directamente su vida útil.

- *Programabilidad*. Debe ser flexible, es decir, debe permitir que sus características y funcionamiento sean configuradas o reconfiguradas mediante el uso de políticas de comportamiento que le son enviadas por el programador o usuario de la red.

- *Adaptabilidad*. El sistema *middleware* debe poder adaptar dinámicamente su rendimiento y el consumo de recursos, esto con el objetivo de lograr seguir cumpliendo con las necesidades de una aplicación. Por ejemplo, en el caso en el que las reservas de energía se estén agotando, el nodo podría cambiar su estrategia de comunicación por una que le siga proporcionado esa funcionalidad pero a un bajo costo de energía. Una adaptación que es provocada por un evento dado es conocida como adaptación reactiva, para funcionar necesita que se provea algún mecanismo que le permita monitorear las condiciones del sistema. Un mecanismo para lograr la adaptabilidad son los algoritmos de fidelidad adaptable.

- *Escalabilidad*. En el futuro, se espera que los nodos de sensado sean cada vez más pequeños y baratos, y que las WSN estén conformadas por cientos o incluso miles de nodos. Un sistema *middleware* debe estar listo para el manejo de cantidades grandes de nodos. Una forma de lograr esto son los mecanismos localizados, donde los dispositivos interactúan solamente con su vecindad inmediata.

- *Topología*. Las WSN pueden exhibir una topología de red muy dinámica debido a la movilidad de los nodos, obstrucciones ambientales o a fallas de *hardware* (principalmente agotamiento de baterías o a la falla crítica de algún dispositivo del nodo). Por lo que los sistemas *middleware* deben soportar una operación robusta de las WSN sin prestar importancia a eventos desfavorables que se llegaran a presentar. Por ejemplo, al utilizar un mecanismo *data-centric* (centrado en los datos), si uno de los nodos

llegara a fallar, otro nodo con características similares sería utilizado en su lugar.

- *Integración*. Las WSN son raramente sistemas independientes, en la mayoría de las ocasiones, están conectadas a dispositivos e infraestructuras externas por diversas razones. Una es que esto podría ser necesario para la asignación de las tareas a la red, también para la evaluación y almacenamiento de los resultados. Infraestructuras tales como Internet, proveen recursos (sobre todo de poder de procesamiento y dimensiones de almacenamiento) que no están disponibles en la red. El *middleware* debe permitir la integración con infraestructuras externas y al mismo tiempo proporcionar una vista homogénea de la red. Una nota importante es que los nodos no poseen los recursos necesarios para ejecutar componentes complejos de *middleware*, una solución viable sería externar los componentes a la infraestructura y mantener solamente una funcionalidad mínima en los nodos. Aunque eventualmente esta funcionalidad tendría que ser cambiada para apartarse a nuevas necesidades, por lo que también serían necesarios mecanismos para actualizar dinámicamente esta funcionalidad.

- *Seguridad*. Se deben proveer mecanismos de seguridad que permitan mantener a salvo la información durante su procesamiento y sobre todo en su comunicación. También se debe tener cuidado en el sabotaje los dispositivos de sensado, ya sea por la alteración de los mismos o por la falsificación de las lecturas de sensado.

- *Calidad de servicio*. Las WSN son utilizadas para monitorear fenómenos del mundo real, por lo que se deben tener en cuenta las propiedades no funcionales de la red, tales como la disponibilidad, tolerancia a fallos, puntualidad, entre otras.

III.3. Clasificación del *middleware*

Aún no se ha llegado a una clasificación concisa para el estudio de los diferentes sistemas *middleware* para WSN propuestos y en desarrollo. Marrón (2005) propone una clasificación en base al tipo de nivel de abstracción. A continuación, se describe cada una de las

categorías y se mencionan algunos de los proyectos que le corresponderían, haciendo una breve reseña del proyecto más representativo de la misma.

III.3.1. Clásico

Este tipo de sistemas básicamente esconden la complejidad de la comunicación y la transferencia de datos en la red. Hood (Whitehouse et al., 2004) es uno de estos sistemas. Este sistema provee una interfaz a un subconjunto de los nodos de la red, el cual es conocido consecuentemente como vecindad. Para definir estas vecindades, se hace uso de un cierto criterio y de un conjunto de atributos a ser compartidos, por ejemplo, se podría hacer una vecindad en la que el vecindario esté conformados por nodos a una distancia de un salto y que sean capaces de proveer lecturas de temperatura. Hood se encarga de cualquier problema de administración. La comunicación se realiza por medio de un mecanismo *broadcast*/filtrado, cuando un nodo desea enviar alguna información, este hace un *broadcast* con la misma, los nodos receptores filtran estos mensajes y deciden cuál de ellos es de su interés para mantenerlo o ignorarlo si no es el caso. En la interfaz que Hood ofrece, una vecindad es manejada como una primitiva de programación. Hood también ofrece algunas interfaces que proveen acceso a los atributos de la interfaz de vecindad y en los que se pueden definen estrategias para el uso compartido de los datos. Gracias a este sistema, los programadores pueden utilizar la red sin hacer un esfuerzo en comprender las situaciones de bajo nivel. Otros proyectos en esta categoría: Impala (Liu y Martonosi, 2003), TinyLime (Curino et al., 2005) y EnviroTrack (Abdelzaher et al., 2004).

III.3.2. *Data-centric*

Esta categoría está conformada por los sistemas en los que se hace un enfoque principal a la información que es generada por los nodos de la red, sin hacer referencia a la identificación del nodo o nodos específicos que la generaron. Aunque también está conformada por aquellos sistemas en los que la red es observada como una base de datos, en la que las consultas y asignación de tareas son realizadas a través de llamadas descritas en

alguna forma de SQL. El proyecto representativo por de facto de esta categoría es TinyDB (Madden et al., 2002). TinyDB provee una base de datos con una «tabla» virtual llamada *sensors*, donde cada columna corresponde a un tipo específico de sensor (temperatura, luz, humedad, etc.) o a otra fuente de información (número de identificación del nodo, batería restante, etc.). Cada uno de los nodos es representado por una fila de esta tabla. El lenguaje de consulta es un subconjunto de SQL con algunas extensiones. TinyDB usa una solución descentralizada, en donde cada nodo tiene su propio procesador de consultas que preprocesa y agrega su información a la respuesta global. Este actividad es organizada por medio del uso de un árbol de expansión que es constantemente mantenido. Al entrar una consulta, esta es enviada al nodo raíz del árbol, el cual a su vez la envía a sus hijos inmediatos, y estos a su vez hacen lo mismo, y así consecutivamente hasta llegar a las hojas del árbol. Una vez alcanzadas, la información ya agregada se empieza a transitar de regreso por cada uno de los padres hasta que esta llega de nuevo a la raíz, con lo que se obtiene la información deseada. Otros proyectos en esta categoría: Cougar (Bonnet et al., 2000), DSWare (Li et al., 2003) y SINA (Jaikaeo et al., 2000).

III.3.3. Máquinas virtuales

En esta categoría se encuentran los sistemas que hacen que las redes sean vistas como una colección de interpretadores de código que se hacen cargo de programas o *scripts* ejecutables. Uno de estos sistemas es Maté (Levis y Culler, 2002). Reprogramar un nodo implica escribir en su memoria una imagen binaria del software, si esto se desea hacer una vez que la red ya se encuentra instalada y funcionando, la imagen completa, que incluye todo el *firmware* necesario para hacer funcionar el *software* en el nodo, tendría que ser transmitida en un camino multisalto hacia cada uno de los nodos de la red. Dependiendo del tamaño de la imagen, este procedimiento consume una tremenda cantidad de energía. Maté es una máquina virtual en la que las aplicaciones están escritas utilizando un conjunto específico de código en bits, gracias a esto, la reprogramación se puede llevar a cabo al enviar únicamente la aplicación deseada. Este conjunto combina instrucciones de bajo

y alto nivel y sólo permite tres tipos de operadores posibles: valores, lecturas de sensado y mensajes. Además de las operaciones básicas de cómputo. Otros proyectos en esta categoría: Smart Messages (Kang et al., 2004) y Agilla (Fok et al., 2005).

III.3.4. Sistemas adaptables

La característica principal de este tipo de sistemas es que se enfocan en la adaptabilidad del *software* del nodo. Un ejemplo de este tipo de sistemas es TinyCubus (Marrón et al., 2005). El sistema TinyCubus consiste en un repositorio de componentes clasificados en tres distintos parámetros (se puede pensar en él como un cubo): parámetros de sistema, requerimientos de la aplicación y un parámetro de optimización. La función principal de TinyCubus es la de seleccionar el mejor conjunto de componentes que permita la funcionalidad deseada en el nodo. Esto se hace de manera continua, y conforme avanza el tiempo, estos tres parámetros podrían cambiar y otra selección de componentes sería necesaria. TinyCubus permite hacer cambios en el repositorio de componentes en tiempo de ejecución, para por ejemplo, agregar un nuevo componente. Otros proyectos en esta categoría: MiLAN (Heinzelman et al., 2004) y AutoSec (Han y Venkatasubramanian, 2001).

III.3.5. Orientación a servicios

Los sistemas *middleware* diseñados a partir de una arquitectura orientada a servicios son aquellos en el que los nodos, o bien la red misma, son utilizados como proveedores de servicios. Este tipo de sistemas está aún siendo abordado por nuevas investigaciones y surge a partir de la necesidad de soluciones con mayor abstracción, facilidad de uso e integración. Algunos trabajos anteriores en esta categoría son discutidos en el siguiente capítulo. El diseño de una arquitectura que permita el desarrollo de sistemas *middleware* orientados a servicios es el objetivo de este presente trabajo.

Capítulo IV

Arquitecturas Orientadas a Servicios

«*Las cosas deberían hacerse del modo más sencillo posible, pero no más simples*»

– Albert Einstein (1879-1955)

De manera similar a la situación con las WSN, en el desarrollo tradicional de *software* ha existido el problema de proveer mejores abstracciones para la implementación. Si se observa la evolución de la programación tradicional podremos ver cómo ésta ha venido avanzado desde la programación secuencial, pasando por la programación modular, la orientación a objetos y la programación basada en componentes hasta llegar a la orientación a servicios. Como explica Hashimi (2004), al existir solamente la programación secuencial los primeros desarrolladores se percataron de que el escribir *software* se estaba volviendo cada vez más complejo y de que necesitaban de una mejor manera de reutilizar el código que constantemente estaban reescribiendo. Se introdujo el concepto de diseño modular, con él los desarrolladores podrían escribir subrutinas y funciones y reutilizar aquellas que necesitaran. Copiar código ya existente en la creación de nuevas aplicaciones pareció funcionar, sin embargo, se empezaron a dar cuenta de que cuando en una de estas aplicaciones se encontraba un problema, los desarrolladores debían buscar el módulo específico causante de la problemática y modificar su código en cada una de las aplicaciones en las que éste fue utilizado. Un mayor nivel de abstracción era necesario. La orientación a objetos fue propuesta. Con esta nueva abstracción los desarrolladores encontraron que reutilizar y mantener código aún seguía siendo una tarea compleja y que, aunque esta abstracción les permitía reutilizar código, ésta no les permitía reutilizar la funcionalidad, lo cual

es más deseable. Por lo que surgió una nueva capa de abstracción: el *software* basado en componentes. La orientación a componentes es una buena solución para la reutilización y mantenimiento de código, sin embargo, no toma en cuenta las complejidades actuales. El *software* de hoy tiene que enfrentarse a situaciones tales como el *software* distribuido y la integración de aplicaciones, así como a una gran heterogeneidad en plataformas, protocolos y dispositivos, además de Internet entre otros. La orientación a servicios ha surgido como una respuesta a esto y como una solución para facilitar la integración entre componentes de *software*. Diversos autores han considerado a las arquitecturas orientadas a servicios como la siguiente ola de desarrollo de aplicaciones (Pallos, 2001). Se considera que un factor por el cual este tipo de arquitecturas tiene buena recepción es que se adapta a la dinámica de negocios actual, la cual, está sustentada a través de servicios, por lo que son muy útiles en la integración de *software* interempresarial.

Antes de entrar en detalle en la definición de este tipo de arquitecturas, resulta adecuado introducir el concepto de acoplamiento, el cual se define como el grado de dependencia entre sistemas que interactúan entre sí, es decir, el grado en el que un sistema depende de otro para su propio funcionamiento. Existen dos tipos de estas dependencias:

- *Dependencia real*. Se le conoce así al conjunto de características o servicios que un sistema consume de otros sistemas.
- *Dependencia artificial*. Es el conjunto de factores que un sistema debe cumplir para consumir las características o servicios proveídos por otros sistemas.

Un ejemplo en la vida real sería el siguiente: en Europa se utiliza otro tipo de enchufes eléctricos a los que se utilizan en Norteamérica. Si un turista norteamericano viaja a Europa y desea enchufar algún aparato eléctrico que haya llevado consigo, éste debería utilizar un convertidor para poder adaptarse al enchufe europeo. En este ejemplo, la dependencia real estaría representada por la necesidad de energía eléctrica, mientras que la artificial sería que el enchufe del aparato eléctrico debe embonar con la toma eléctrica en la pared.

Figura 5: **Componentes de SOA**

Una arquitectura orientada a servicios (SOA, por sus siglas en inglés) es un estilo arquitectónico cuyo objetivo principal es lograr un bajo acoplamiento entre agentes de *software* que interactúan entre sí (el bajo acoplamiento describe la configuración en la cual la dependencia artificial ha sido reducida a un mínimo, sin afectarse de alguna manera la dependencia real). SOA está conformada por proveedores y consumidores de servicios, así como por un directorio de los mismos. Estos roles están ilustrados en la Figura 5. En la arquitectura existen tres operaciones principales: *publish*, *find* y *bind* (Graham et al., 2004).

Un servicio es una unidad de trabajo realizada por un proveedor de servicios para lograr resultados deseados por un consumidor del mismo. Ambos roles, el proveedor y el consumidor, son realizados por agentes de *software* en representación de sus respectivos dueños. La razón principal para consumir un servicio es que generalmente resulta mucho más barato y efectivo que hacer el trabajo por si mismo. Con esto se logra un principio de ingeniería de *software* conocido como «separación de preocupaciones» (He, 2003).

El descubrimiento de servicios de manera dinámica es una parte importante de SOA, esto se logra gracias al uso del directorio. El directorio de servicios es un intermediario entre proveedores y consumidores, con su uso es posible:

Figura 6: **Detalle de la operación *publish***

- *Una gran escalabilidad en el número de servicios.* Gracias a que en el directorio sólo se encuentra una descripción de los servicios disponibles y a que estos servicios son proveídos por entidades externas al directorio.

- *Desacoplar a los consumidores de los proveedores.* Al presentar al consumidor con el proveedor y preparar la comunicación entre ambos.

- *Permitir actualizaciones de servicios.* Si la actualización de un servicio es necesaria, esta es solamente llevada a cabo en el directorio (mediante el registro del servicio actualizado), por lo que no es necesaria alguna modificación en los consumidores.

- *Proveer un servicio de búsqueda a los consumidores.* Para permitir el consumir sea capaz, por sí mismo, de seleccionar el proveedor que más se adapte a sus necesidades. Todo esto en tiempo de ejecución.

El funcionamiento de los roles en la arquitectura es el siguiente (Chappell y Jewell, 2002): inicialmente, un proveedor de servicios necesita registrar en el directorio el servicio que está listo a proveer. Para ello utiliza la operación *publish* (véase la Figura 6). Una vez registrado, los consumidores potenciales del servicio, utilizando el directorio, pueden localizar dinámicamente al servicio mediante la operación *find* (véase la Figura 7). Cuando el servicio ya fue identificado por el consumidor, el directorio le provee información sobre la forma en cómo puede contactar al proveedor y entonces el consumidor es capaz de preparar la comunicación utilizando la operación *bind* (véase la Figura 8), para posteriormente utilizar el servicio en sus necesidades.

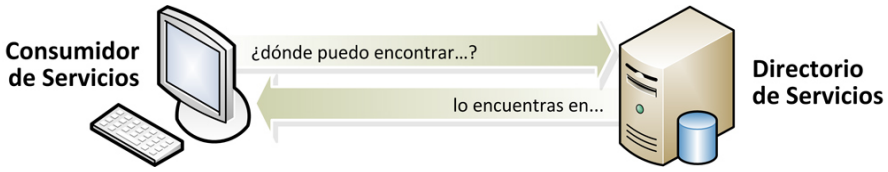


Figura 7: **Detalle de la operación *find***

Figura 8: **Detalle de la operación *bind* y del uso del servicio**

Otro ejemplo en la vida real puede ilustrar el funcionamiento: tomemos el caso de la reproducción de un disco compacto de música. Una persona (en este caso el consumidor) que desee reproducir el disco compacto (el servicio que se necesita) necesitaría saber qué aparato reproductor puede hacerlo (aquí es donde el directorio interviene), posteriormente acude al aparato, introduce el disco (operación *bind*) y lo reproduce. Obsérvese cómo este servicio puede ser ofrecido por otros aparatos, tal vez con diferentes características (por ejemplo tener mejor calidad en la reproducción), sin embargo, el servicio y el consumidor siguen siendo los mismos.

La comunicación entre todos los componentes de la arquitectura es realizada a través de mensajes descriptivos, extensibles e independientes de plataforma o lenguaje. Todo esto con el objetivo de mantener el acoplamiento tan bajo como sea posible. Esto se logra gracias a dos restricciones arquitectónicas:

- Los agentes de *software* participantes deben tener un pequeño conjunto de interfaces simples y ubicuas. En estas interfaces solamente deberá estar codificada semántica

genérica. Además deberán estar disponibles universalmente para todos los proveedores y consumidores.

- Las instrucciones comunicadas a través de estas interfaces deberán estar codificadas en un esquema escalable, además deben ser únicamente descriptivas, es decir, deberán indicar acciones a ser realizadas (volviendo al ejemplo anterior, la persona sólo indica qué pista desea reproducir, no cómo debe leerse el disco, interpretar la información, etc.), aunque, un mínimo de comportamiento de sistema podrá ser enviado. Un esquema limita el vocabulario y la estructura de los mensajes, sin embargo, un esquema escalable permite que nuevas versiones de los servicios sean introducidas sin interferir con servicios existentes.

IV.1. Servicios web

Aunque SOA no es un concepto nuevo (fue introducido hace un poco más de una década) es gracias a los servicios web que nuevamente se le presta atención. Los servicios Web están construidos encima de protocolos bien conocidos e independientes de plataforma. Estos protocolos incluyen HTTP, XML, UDDI, WSDL y SOAP. Es la combinación de estos protocolos lo que hacen a los servicios web tan atractivos.

Los protocolos UDDI, WSDL y SOAP se encargan del descubrimiento dinámico de servicios, con XML se obtiene una interfaz independiente de plataforma así como escalable y con HTTP se obtiene una plataforma de interoperabilidad. Además de cumplir con el patrón de SOA, la tecnología de servicios web puede ser fraccionada en tres pilas de protocolos: la pila física o de intercambio (HTTP), la pila de descripción (XML y SOAP) y la pila de publicación y descubrimiento (UDDI).

IV.2. *Middleware* orientado a servicios para WSN

Trabajos anteriores han logrado realizar una abstracción de alto nivel. Uno de los más relevantes es Madden et al. (2002), en el que la red de sensores es manejada como si se tratara de una base de datos, permitiendo la interacción con la red a través de consultas SQL. Poco a poco se está prestando atención al esquema orientado a servicios como una manera de lograr este tipo de abstracción y además ofrecer nuevas ventajas, entre las que destaca la facilidad de integración con otras infraestructuras ya sean externas a la red o incluso la integración con otro tipo de redes. A continuación se describen algunos primeros trabajos en los que se ha empezado a aplicar el esquema orientado a servicios en el diseño de un sistema *middleware* para WSN.

IV.2.1. Golatowski et al. (2003)

En su trabajo, Golatowski et al., define a un alto nivel un sencillo modelo orientado a servicios en el cual relega la responsabilidad del procesamiento de las solicitudes de los servicios a una entidad externa de la red. Este servidor estaría encargado de servir como «puente» entre las solicitudes entregadas desde el exterior y la funcionalidad interna de la red. En la propuesta no se definen a detalle cómo estos componentes interactuarían en cuanto a qué tipo de protocolos utilizarían, ni explica cuáles serían las características propias de cada uno de los componentes.

IV.2.2. Delicato et al. (2003a;b; 2005; 2006)

Delicato et al. han venido trabajando en una serie de trabajos en los que aplica el modelo orientado a servicios, principalmente la tecnología de servicios web, en diversos sistemas *middleware* con características diferentes. A continuación se hace una breve descripción de las propuestas realizadas en cada uno de los artículos y se hace un análisis de la problemática encontrada en ellos.

Delicato et al. (2003b)

En el artículo Delicato et al. (2003b), se presenta una arquitectura que, haciendo uso de la tecnología de los servicios web, intenta lograr un bajo acoplamiento entre la diseminación de datos en la red y las características específicas a la aplicación. Para ello utiliza un esquema basado en Directed Diffusion en el diseño de la arquitectura. En ella, se definen los componentes y los elementos WSDL necesarios para lograr sus objetivos.

Directed Diffusion (Intanagonwiwat et al., 2000) es un protocolo de diseminación centrado a datos diseñado específicamente para las WSN. En Directed Diffusion, los nodos individuales capturan los parámetros de sensado en descripciones de «eventos». Una descripción contiene un conjunto de atributos. Las aplicaciones que solicitan información envían sus intereses por medio de uno de los nodos *sink* de la red. Estos intereses son también representados como un conjunto de atributos. Si los atributos de la información generada por los nodos de sensado concuerdan con estos intereses, la información es enviada hacia fuera de la red por medio de los nodos *sink*.

El artículo hace tres contribuciones principales. La primera es la arquitectura en sí, en la cual separa la funcionalidad de comunicación de datos del procesamiento específico a la aplicación. La segunda es que define una solución de servicios web para WSN, donde los nodos *sink* están modelados como servicios web que exponen los servicios proveídos por la red. Y por último, la tercera, en la que se propone el uso del lenguaje WSDL y del protocolo SOAP, como mecanismo para describir los servicios y dar formato a los mensajes usados por el protocolo de comunicación de bajo nivel.

En la propuesta de Delicato et al. (2003b) se identificaron las siguientes problemáticas:

- No respalda sus contribuciones mediante la implementación de su modelo en alguna plataforma de sensores. Al no hacerlo, ciertos problemas no podrían ser detectados y por lo tanto influir en la adopción y aplicación de la arquitectura.

- Menciona que se hace un esfuerzo por mantener la cantidad de energía gastada en el mismo nivel que los sistemas WSN actuales, y para ello, hace uso del protocolo XML binario. Sin embargo, podría haber problemas para llevar a cabo la arquitectura, como se puede observar en la Figura 2, las capacidades actuales en cuanto a procesamiento y memoria de las plataformas son demasiado limitadas. Haciendo poco viable el manejo de alguna forma de XML binario en la red, sobre todo por el tiempo extra gastado en el empacamiento y desempacamiento, interpretación y en la agregación de mensajes con este protocolo. Aunque, los autores mencionan que, debido a que el consumo de energía en el procesamiento de datos se asume como de un orden o magnitud más pequeña que en la transmisión de datos, el procesamiento adicional causado por los mensajes SOAP debería ser insignificante en el sistema, sin embargo, no se menciona cómo esto podría lograrse.

Delicato et al. (2003a)

En este otro artículo se presenta una evolución de la arquitectura presentada anteriormente. El objetivo aquí es el de proponer una capa de interoperabilidad genérica y flexible para las WSN, que provee las funcionalidades básicas requeridas por cualquier WSN. Tal capa *middleware* está compuesta del protocolo SOAP e interfaces proveídas por documentos WSDL. Utilizando protocolos de diseminación específicos para WSN y la capa *middleware* basada en servicios, se desea ofrecer una manera flexible de manipular, extraer e intercambiar datos desde las WSN. Las aplicaciones acceden a la red y modifican el comportamiento de la capa de bajo nivel de diseminación de datos a través de una interfaz común e independiente de plataforma proveída por la capa *middleware*.

Algunos problemas encontrados:

- De nuevo, se sigue haciendo uso de protocolos no diseñados para WSN en la lógica interna de la red, aunque, a un bajo nivel se utilizan protocolos de diseminación para WSN. Ahora se propone la utilización del protocolo SOAP en la comunicación interna

mediante una capa de interoperabilidad. Una de las debilidades del protocolo SOAP es el procesamiento adicional que este implica, por lo que su uso en las WSN podría causar gastos en energía excesivos haciendo a la propuesta inviable.

- El trabajo continúa sin tener una comprobación, ya sea por medio de simulaciones o por la implementación de un sistema basado en la arquitectura propuesta en alguna plataforma WSN. Sobre todo en términos de consumo de energía.

Delicato et al. (2005)

Una evolución más en el trabajo de Delicato et al. es este otro artículo en el que se propone un *middleware* reflexivo y orientado a servicios para WSN. Además de proveer un modelo de programación abstracto para aplicaciones WSN, mantiene el balance entre requerimientos de calidad de servicios de la aplicación y el tiempo de vida de la red. Tal *middleware* está encargado de tomar decisiones acerca de los protocolos de comunicación, organización topológica de la red, modos de operación de los sensores y otras funciones infraestructurales típicas de las WSN. El *middleware* monitorea la red y los estados de ejecución de la aplicación realizando una adaptación de la red donde sea necesario, con o sin interferencia con la aplicación.

Se hacen tres contribuciones principales. La primera, el sistema provee una capa de interoperabilidad entre diferentes aplicaciones y la WSN. Las aplicaciones usan la red a través de diferentes protocolos y organizaciones, sin preocuparse sobre los detalles de la operación de la red. Segundo, los servicios proveídos por el *middleware* son accedidos de manera flexible a través de un lenguaje estándar y de alto nivel. El uso de XML y SOAP en la comunicación de la red provee un máximo de flexibilidad e interoperabilidad. Finalmente, los servicios de decisión proveídos sobre la configuración de la red y de su adaptación con conciencia del contexto, ayudan a incrementar el tiempo de vida global de la red, mientras se cumplen con los requerimientos de la aplicación.

El sistema fue evaluado de acuerdo a dos metodologías diferentes: una cualitativa y una cuantitativa. El objetivo de la evaluación cualitativa fue la implementación de un prototipo del *middleware* propuesto como una prueba de concepto para el sistema. El prototipo fue desarrollado utilizando el lenguaje Java[1] y la plataforma de servicios web Axis de Apache[2]. En la evaluación cuantitativa se llevaron a cabo simulaciones con dos objetivos: para mostrar que la adopción de un protocolo de red cumpliendo con los requerimientos específicos de la aplicación podría mejorar el desempeño general de la red en términos de ahorro de energía; y para probar que se pueden obtener mejoras en el tiempo de la vida de la red a través de la adopción de un comportamiento adaptativo, en contraste con la configuración estática de la red.

Problemática encontrada:

- Se continúa utilizando SOAP y XML en la comunicación interna de la red. Los autores llevaron a cabo simulaciones para verificar la validez, en términos de consumo de energía, del uso de un esquema adaptable. Pruebas mediante simulaciones para sustentar el uso de estos protocolos, sobre todo acerca del consumo de energía adicional, habrían sido interesantes.
- Aunque se implementó un prototipo, éste no lo fue en una plataforma WSN, con lo que ciertos problemas específicos de las WSN no podrían haber sido detectados.

Delicato et al. (2006)

En este artículo, los autores presentan una propuesta que tiene como base las ideas y contribuciones de los artículos anteriores. Esta nueva propuesta consiste de tres puntos principales: un modelo de programación orientado a servicios para WSN, una interfaz

[1] Sun Microsystems. `http://java.sun.com/`
[2] Web Services Project Apache. `http://ws.apache.org/`

estándar para acceder a la red y un conjunto de componentes de servicio configurables para apoyar el desarrollo de aplicaciones y para controlar el comportamiento de la red durante la ejecución de tareas de sensado enviadas a la misma. Uno de los aspectos interesantes de la propuesta es la inclusión de parámetros de calidad de servicio que son utilizados para administrar el comportamiento de la red.

La desventaja de la propuesta continúa siendo la dificultad de aplicación actual. Se continúa asignando tareas de procesamiento de contenido XML a los nodos de sensado. Para la evaluación de factibilidad de la propuesta, los autores implementaron emuladores que, aunque están basados en especificaciones de *hardware* de una plataforma WSN, continúan siendo programados en Java y presentan los resultados de simulaciones realizadas con los mismos. Estos resultados se enfocaron en medir aspectos tales como la memoria utilizada y el tráfico de la red.

En el siguiente capítulo se presenta la arquitectura propuesta por este trabajo de tesis en la cual se hace un diseño cuidadosamente enfocado a las capacidades limitadas actuales de las plataformas WSN. La propuesta tiene el objetivo principal de lograr una solución viable para reducir la brecha actual entre los requerimientos de una aplicación y los temas específicos de la red.

Capítulo V

Arquitectura Orientada a Servicios para WSN

«*Debe ser simple para ser cierto. Si no es simple, probablemente no prodremos descifrarlo*»

– Albert Einstein (1879-1955)

Después de publicarse el libro de Charles Darwin, 'El Origen de las Especies', el zoólogo alemán Ernst Haeckel estableció que «la ontogenia recapitula la filogenia». Con esto quiso decir que el desarrollo de un embrión (ontogenia) repite (es decir, recapitula) la evolución de la especie (filogenia). En otras palabras, después de la fertilización, un embrión humano pasa por las etapas de pez, cerdo, etc., antes de convertirse en un bebé humano. Los biólogos modernos consideran esto una burda simplificación, pero no deja de tener su núcleo de verdad. Algo análogo ha sucedido en la industria de la computación. Tanenbaum (2001) menciona que cada vez que aparece un nuevo tipo de computadora (*mainframe*, miniordenador, ordenador personal, etc.) esta pasa por el mismo desarrollo que sus antepasados, estos dispositivos empiezan a ser programados utilizando lenguaje ensamblador hasta llegar a lenguajes de alto nivel. Este patrón también puede ser observado en el desarrollo de la programación en las WSN.

Hasta ahora la mayor actividad de investigación se ha conducido en las capas de bajo nivel de las WSN, sin embargo, aún falta mucho trabajo en la capa de aplicación. Todo esto nos lleva a este trabajo, en el cual el objetivo principal es el de explorar la factibilidad de utilizar un enfoque orientado a servicios en el diseño de una arquitectura para el desarrollo de sistemas *middleware* para WSN. Para el cual se hace un diseño cuidadosamente

enfocado a las capacidades limitadas de las plataformas de sensores disponibles actualmente. El estudio se centró en la aplicación de una WSN para actividades de visualización o monitoreo solamente.

V.1. Requerimientos de la arquitectura

En el Capítulo III se mencionan las características que presentan las aplicaciones y los sistemas *middleware* para WSN así como los retos a los que estos últimos deben enfrentarse. Para poder diseñar una arquitectura que tenga éxito ante estos retos, se debe realizar un diseño fuertemente enfocado en las características propias de las WSN. Para ello, los requerimientos para el diseño de la arquitectura presentada en este trabajo son en dos rubros: los requerimientos específicos para los sistemas *middleware* para WSN y los requerimientos de los desarrolladores de aplicaciones WSN que utilizarán estos sistemas.

Los requerimientos para sistemas *middleware* para WSN que se utilizaron en el diseño de la arquitectura propuesta son (Römer, 2004, Marrón, 2005):

- *Abstracción.* El sistema *middleware* debe permitir la especificación de las tareas a un alto nivel. También debe esconder en lo más posible los detalles de *hardware* y de distribución al programador.
- *Eficiencia.* Debe tener especial cuidado en el consumo de energía.
- *Programabilidad.* Debe ser flexible y para ello, permitir la configuración o reconfiguración de sus características y funcionamiento.
- *Adaptabilidad.* El sistema debe ser capaz de adaptarse a los cambios de la red para continuar con su correcto funcionamiento.
- *Escalabilidad.* Debe permitir el crecimiento en el número de nodos de la red y estar preparado para ello.

- *Topología*. Debe proveer mecanismos para soportar la topología siempre cambiante de la red y garantizar robustez en la misma.
- *Integración*. Debe permitir la integración con infraestructuras externas o incluso con otras redes.

Existen otros dos requerimientos los cuales, por requerir de más tiempo del disponible para el desarrollo de este trabajo de tesis, no son considerados. Estos son:

- *Seguridad*. Se deben proveer mecanismos de seguridad que permitan mantener a salvo la información durante su procesamiento y sobre todo en su comunicación.
- *Calidad de servicio*. Se deben tener en cuenta las propiedades no funcionales de la red, tales como la disponibilidad, tolerancia a fallos, puntualidad, entre otras.

En su trabajo, Buonadonna et al. (2005), han identificado algunos requerimientos de los desarrolladores de aplicaciones WSN. Estos requerimientos también son importantes para este trabajo ya que uno de nuestros objetivos es facilitar la creación de nuevas aplicaciones WSN, y para ello, debemos satisfacer las necesidades de los desarrolladores. Los requerimientos son:

- *Reconfigurabilidad*. Para que las aplicaciones puedan cambiar de tasa de datos, tipos de datos recolectados, etc.
- *Monitoreo de vitalidad*. Para conocer la vitalidad de la red. Por ejemplo mediante el conocimiento del número de nodos con energía.
- *Lecturas de sensado interpretables*. Las lecturas deben ser reportadas en unidades de ingeniería. Por ejemplo en grados centígrados.
- *Integración*. No solamente se deben dar medios para permitir la integración con otras herramientas, esta también debe ser sencilla.

- *API familiar*. Se debe proveer un API que sea de fácil uso para el desarrollador, idealmente nativo al lenguaje de programación que este utiliza habitualmente.
- *Extensibilidad en el software del nodo*. Para por ejemplo poder agregar nuevos tipos de sensores o nuevos operadores de datos.
- *Modularidad en los servicios de red*. Para permitir la experimentación con diferentes soluciones para enrutamiento, calendarización, etc.

V.2. Análisis y diseño de la arquitectura

En base a los requerimientos antes mencionados, y como solución para alcanzar los objetivos de esta tesis, se realizó el diseño de una arquitectura para sistemas *middleware*. Posteriormente, sus componentes se fueron refinando a través de cada una de las iteraciones que se realizaron de acuerdo a la metodología conducida. La arquitectura resultado de estas iteraciones es presentada a continuación.

V.2.1. Descripción de la arquitectura

En la Figura 9 se puede observar el escenario de ejecución de la arquitectura. Todas las operaciones tienen lugar en tres secciones o áreas:

- *Captura*. En esta área se encuentra la WSN en operación, la cual mediante un modelo *publish/subscribe*, comunica lecturas de sensado e información de mantenimiento al exterior a través de un nodo *sink* que actúa como puente de comunicación. De igual manera, a través de este nodo se permite la «inyección» de solicitudes a la red.
- *Concentración*. En esta área se realiza la mayor parte del procesamiento. Las lecturas e información enviada por los sensores es recabada y clasificada para su posterior uso. En esta área también se proporciona acceso a los datos y al control de la red por medio de un servidor de servicios web. Esta actividad puede ser realizada ya sea por

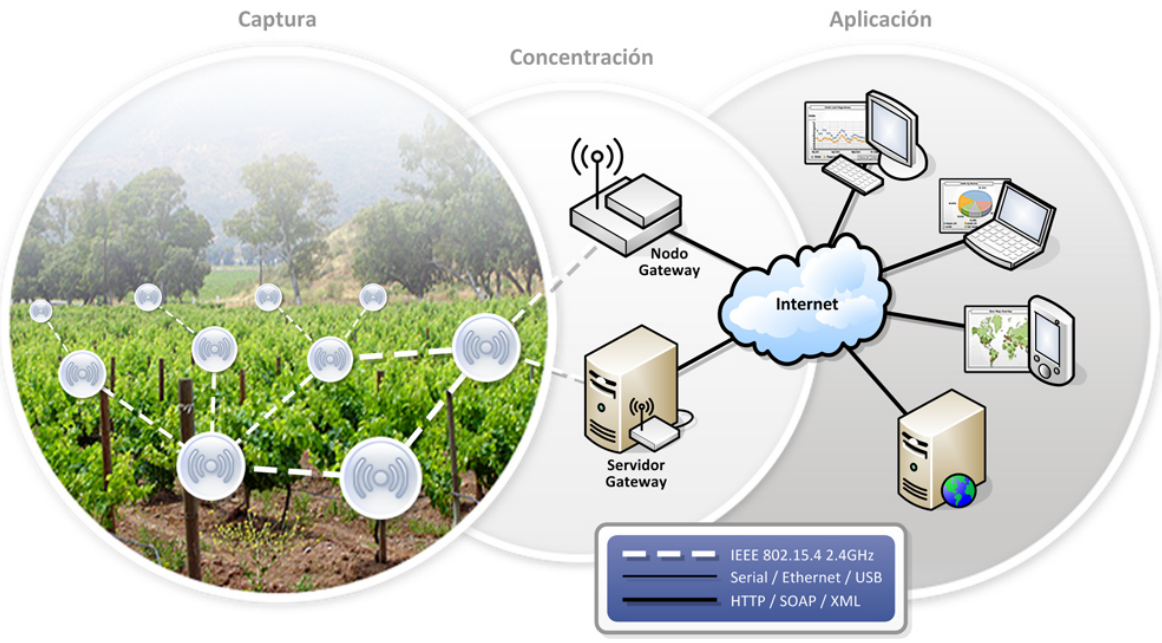


Figura 9: **Escenario de ejecución de la arquitectura**

una computadora conectada a la red por medio de un nodo de sensado (Servidor Gateway, también referido como «nodo *sink*») o por un nodo de sensado especializado con capacidades de soporte para servidores (Nodo Gateway).

- *Aplicación.* En esta área se encuentran las aplicaciones de monitoreo y/o visualización creadas con el uso de los servicios proveídos en el área de concentración.

La arquitectura provee dos tipos de servicios: internos y externos. Para lograrlo, intervienen sus cuatro componentes: Nodo, Gateway, Registro y Servidor (véase la Figura 10). A continuación se describen cada uno de estos componentes.

Componente Nodo

Este componente encapsula toda la funcionalidad de un nodo de sensado. Reside en cada uno de los nodos de la red. Estos nodos conforman los servicios internos. Los subcomponentes del nodo son:

- *Control.* Es el responsable de la orquestación de los demás subcomponentes en el

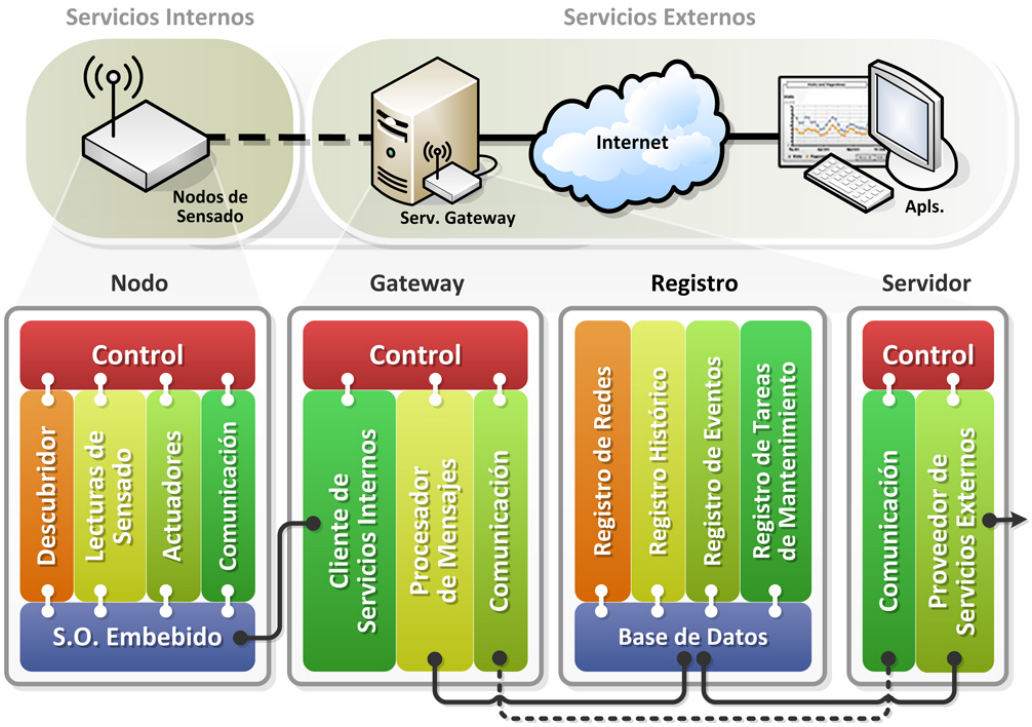


Figura 10: **Componentes de la arquitectura propuesta**

nodo de sensado. También es el que inicializa, recibe, y procesa las acciones.

- *Descubridor*. Durante la inicialización, es el responsable de identificar los servicios ofrecidos por el nodo de sensado. Estos servicios comprenden: los parámetros de sensado que el nodo puede capturar (dispositivos transductores del nodo), actuadores que el nodo tiene a su disposición y lecturas de control (como por ejemplo la lectura de la carga restante en batería).

- *Lecturas de Sensado*. Realiza las lecturas de los diferentes parámetros de sensado del nodo, interactuando para ello con los dispositivos transductores del mismo.

- *Actuadores*. Controla los actuadores en el nodo de sensado. Esto permite al nodo encender o apagar una bocina, un ventilador, etc.

- *Comunicación*. Es encargado de comunicar las lecturas y de recibir comandos enviados desde el exterior de la red a través del componente Gateway.

- *Sistema Operativo Embebido*. El objetivo principal de este subcomponente es el de abstraer la comunicación con el *hardware* del nodo de sensado. Además se puede hacer uso de una gran cantidad de soluciones ya disponibles para: el control de topología, protocolos de comunicación, protección a fallas, etc. Por ejemplo, una de estas soluciones es TinyOS (Hill et al., 2000), la cual ofrece abstracción al *hardware* de una gran diversidad de plataformas WSN y provee un conjunto de mecanismos suficientemente robustos para una aplicación real de la red.

Componente Gateway

Este otro componente es el encargado de fungir como «puente» entre los servicios internos y los externos. Reside en una computadora con acceso a la red por medio de uno de los nodos de sensado, o bien, en un nodo especializado. Es importante mencionar que en la arquitectura propuesta está contemplado el soporte para el funcionamiento de varias redes de sensores a la vez en la misma infraestructura. Para ello, cada una de estas redes debe tener asignado su propio componente Gateway.

- *Control*. Es el encargado de administrar los demás subcomponentes. También es el encargado de inicializar las actividades propias del Gateway.
- *Cliente de Servicios Internos*. Se encarga de la interacción con los servicios internos ofrecidos por los nodos de la red de sensores. Esta interacción consiste en recibir la información generada en la red y en el envío de comandos a la misma.
- *Procesador de Mensajes*. En este subcomponente se procesan los mensajes recibidos de la red. Estos mensajes pueden ser de dos tipos: de registro, o de comunicación de lecturas. Asimismo se encarga de comunicase con el componente Registro para:
  - Registrar la red en el Registro de Redes.
  - Interpretar y añadir las lecturas recibidas al Registro Histórico.
  - Detectar, con la información recabada en los mensajes recibidos, los eventos esperados en el Registro de Eventos.

    - Verificar si en el Registro de Tareas de Mantenimiento se ordena el envío de algún comando a la red y de hacerlo, si es el caso, mediante el Cliente de Servicios Internos.

- *Comunicación*. Este subcomponente es opcional. Está pensado para que sea usado en el caso en el que sea necesaria la recepción de solicitudes de envío de comandos a la red de manera inmediata, sin pasar por el Registro de Tareas de Mantenimiento.

Componente Registro

En este componente, se encuentra almacenada toda la información de la infraestructura en general. Desde las redes de sensores que la han utilizado hasta, entre otro tipo de información, las lecturas que se han recibido de las mismas. Por la misma naturaleza del componente, este reside en una computadora con una buena capacidad de almacenamiento. Los subcomponentes de Registro son:

- *Registro de Redes*. En él se encuentran registradas las redes de sensores que han estado disponibles en la infraestructura. Cada una de ellas con su nombre y una descripción de su aplicación.

- *Registro Histórico*. En este registro se encuentran almacenadas todas las lecturas de sensado recibidas para alguna red en particular, así como también información de control de la misma.

- *Registro de Eventos*. En este otro registro se encuentran almacenados una serie de eventos, representados por medio de criterios que los usuarios de los servicios externos han registrado para su detección.

- *Registro de Tareas de Mantenimiento*. Este registro actúa como un calendarizador de tareas. En él están especificados los comandos que han sido registrados, por medio de los servicios externos, para su envío a la red. Estos comandos pueden indicar las siguientes acciones:

- Encender o apagar algún actuador.
- Cambiar la tasa de muestreo de datos.
- Entrar o salir de un modo de espera a algún nodo de sensado.

Las tareas pueden estar dirigidas a todos los nodos en la red o a un nodo en específico. Además, pueden ser programadas para:

- Ser enviadas inmediatamente.
- Ser enviadas en una fecha y hora específicas. En este caso, adicionalmente se puede especificar que la tarea sea repetida cada cierto tiempo.
- Esperar a la detección de algún evento (previamente registrado en el Registro de Eventos) para su envío.

- *Base de Datos*. Este subcomponente es un manejador de bases de datos del cual dependen todos los registros del componente.

Componente Servidor

En este componente se encuentra la funcionalidad principal de los servicios externos. Básicamente consiste de un servidor de servicios web que permite interactuar con las redes registradas en la infraestructura. Este reside en una computadora con capacidades suficientes para funcionar como servidor.

- *Control*. Es el subcomponente que inicializa y prepara al servidor.
- *Comunicación*. Este es un subcomponente opcional que, junto con su equivalente en el componente Gateway, permite el envío inmediato de comandos a la red sin necesidad del uso del Registro de Tareas de Mantenimiento.
- *Proveedor de Servicios Externos*. Este es el subcomponente principal de Servidor. Es responsable de proveer un servicio web de directorio, con el cual se puede consultar qué redes de sensores se encuentran en el Registro de Redes y los detalles de cada una

de ellas. Además, se provee un servicio web por cada una de las mismas. Un servicio web de red provee una interfaz para:

- Consultar los servicios ofrecidos por la red.
- Acceder a las lecturas en el Registro Histórico.
- Consultar y registrar eventos.
- Consultar y registrar tareas de mantenimiento.

V.2.2. Casos de uso

El caso de uso es una técnica para describir las interacciones de un sistema con sus usuarios potenciales. Es una colección de escenarios iniciados por una entidad llamada actor (una persona, un componente de *hardware* u otro sistema). Un mismo actor puede llevar a cabo varios casos de uso, e inversamente, un caso de uso puede ser llevado a cabo por varios actores. Un caso de uso debería dar por resultado algo de valor ya sea para el actor que lo inició o para otro. Es posible reutilizar casos de uso. Una forma «incluir» es utilizar los pasos de un caso de uso como parte de la secuencia de pasos para otro caso de uso. Otra forma «extender» es crear un nuevo caso de uso mediante la adición de pasos a un caso de uso existente (Schmuller, 2001).

En la Figura 11 se presenta el diagrama de casos de uso global de la arquitectura propuesta. A continuación se presenta la descripción de los casos de uso más significativos.

Caso de uso: *Descubrir Servicios*. Llevado a cabo por el nodo de sensado, tiene como propósito identificar los servicios que el nodo es capaz de proveer.

Descripción: Durante su inicialización, el nodo de sensado identifica los servicios que puede proveer. Esto lo hace al identificar los parámetros que puede leer, los actuadores que tiene a su disposición y las lecturas de control a las que puede acceder.

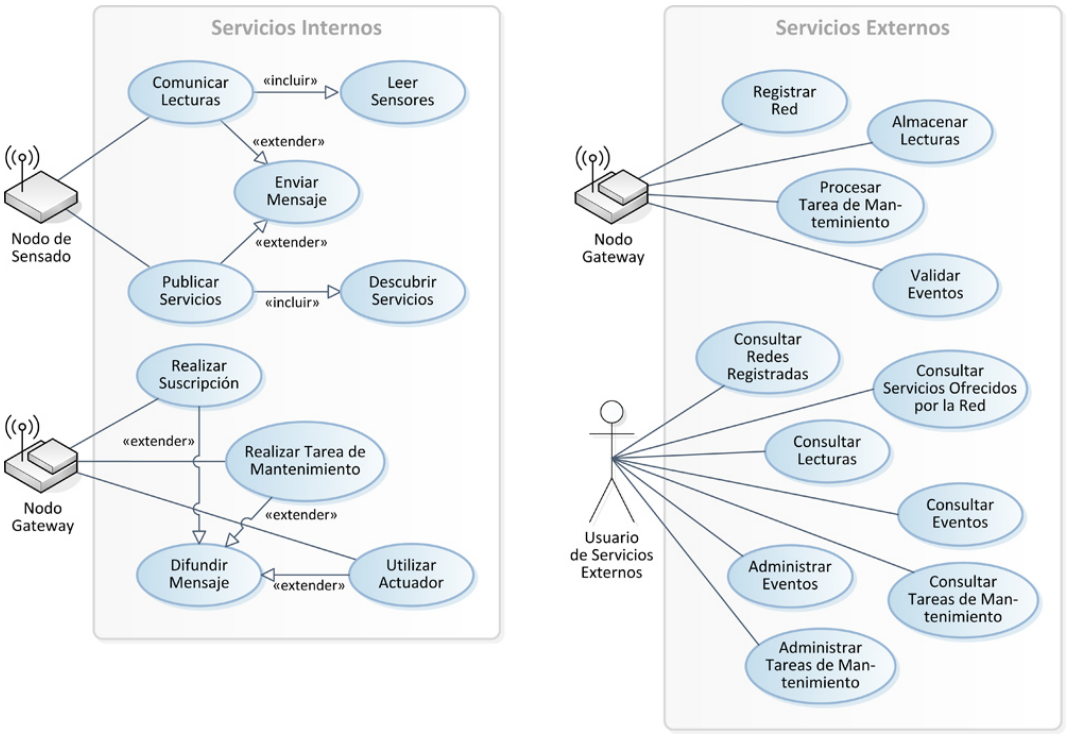


Figura 11: **Diagrama de casos de uso de la arquitectura propuesta**

Caso de uso: *Publicar Servicios*. Llevado a cabo por el nodo de sensado, tiene como objetivo el dar a conocer los servicios con los que cuenta el nodo.

Descripción: El caso de uso inicia una vez que se han identificado los servicios que el nodo puede ofrecer. El nodo entonces puede proceder con el envío de un mensaje de registro con la información del nodo al nodo *gateway* de la red.

Caso de uso: *Registrar Red*. Llevado a cabo por el nodo *gateway*, tiene como objetivo registrar los servicios de la red.

Descripción: El caso de uso inicia cuando se han recibido mensajes de registro desde la red. Con esta información el nodo *gateway* procede a registrar los servicios que son proveídos por la red a la que está asignado. Sin embargo, el nodo *gateway* no registra qué nodo en específico los provee.

Caso de uso: *Realizar Suscripción*. Llevado a cabo por el nodo *gateway*, tiene como objetivo suscribirse a los servicios ofrecidos por los nodos de la red.

Descripción: El caso de uso inicia cuando se han registrado los servicios ofrecidos por la red. Gracias a ello, el nodo *gateway* se suscribe a los servicios que ya sabe que son proveídos por la red. Para ello, difunde un mensaje de suscripción.

Caso de uso: *Leer Sensores*. Llevado a cabo por el nodo de sensado, tiene como objetivo el leer los dispositivos transductores y preparar las lecturas.

Descripción: Este caso de uso inicia cuando se desea obtener las lecturas de los sensores. Para llevarlo a cabo, primeramente inicia los dispositivos transductores para posteriormente realizar la lectura de los mismos.

Caso de uso: *Comunicar Lecturas*. Llevado a cabo por el nodo de sensado, tiene como objetivo el comunicar al nodo *sink* la información recabada por el nodo.

Descripción: Este caso de uso inicia cuando se han suscrito a los servicios ofrecidos por el mismo nodo y cuando se ha realizado la lectura de los sensores. Una vez hecho esto, el nodo envía al nodo *sink* la información correspondiente a los servicios para los cuales hay una suscripción.

Caso de uso: *Almacenar Lecturas*. Llevado a cabo por el nodo *gateway*, tiene como objetivo el almacenar las lecturas recibidas en el registro histórico.

Descripción: Este caso de uso inicia cuando se recibe un mensaje de lecturas desde la red. Estas lecturas son clasificadas y almacenadas en el registro histórico correspondiente a la red de la que se recibió el mensaje.

Caso de uso: *Validar Eventos*. Llevado a cabo por el nodo *gateway*, tiene como objetivo el verificar si los criterios para un cierto evento son cumplidos.

Descripción: Este caso de uso inicia cuando se recibe un mensaje de lecturas desde la red y cuando hay al menos un evento registrado en el registro de eventos. Con la información recabada de la información recibida en mensajes pasados, se verifica si los criterios para un cierto evento en el registro de eventos son satisfechos. Si es así, este evento es modificado en el registro de eventos como «detectado».

Caso de uso: *Consultar Redes Registradas*. Llevado a cabo por el usuario de servicios externos, tiene como objetivo el obtener información acerca de qué redes se encuentran registradas en el registro y de obtener detalles sobre las mismas.

Descripción: Este caso de uso inicia cuando el usuario de servicios externos desea consultar el registro de redes. Para ello, cuando el usuario envía la solicitud, el sistema, comunicándose con el registro de redes, le devuelve un listado con los detalles de cada una de las redes de sensores en el registro.

Caso de uso: *Consultar Servicios Ofrecidos por la Red*. Llevado a cabo por el usuario de servicios externos, tiene como objetivo la consulta de los servicios ofrecidos por una red en específico.

Descripción: Este caso de uso inicia cuando el usuario de servicios externos desea consultar información sobre los servicios ofrecidos por una red. Para ello, el sistema le devuelve un listado con los parámetros de sensado, los actuadores y la información de control que hay disponibles para la red indicada.

Caso de uso: *Consultar Lecturas*. Llevado a cabo por el usuario de servicios externos, tiene como objetivo la consulta de la información de sensado almacenada en el registro histórico.

Descripción: Este caso de uso inicia cuando el usuario de servicios externos desea consultar información de sensado existente en el registro histórico de una red en específico. Para ello, el sistema devuelve para un cierto periodo de tiempo especificado, las lecturas de sensado o información de control disponible para la red indicada.

Caso de uso: *Administrar Eventos*. Llevado a cabo por el usuario de servicios externos, tiene como objetivo el permitir agregar, modificar o eliminar un evento en el registro de eventos para una red en específico.

Descripción: Este caso de uso inicia cuando el usuario de servicios externos desea administrar el registro de eventos. El usuario puede agregar, modificar o eliminar los eventos. Estos se especifican mediante un criterio que puede ser cumplido por la información en la red, por ejemplo, este podría ser «temperatura mayor a 30°C».

V.2.3. Diagramas de secuencia

El diagrama de secuencias UML agrega la dimensión del tiempo a las interacciones de los objetos. En el diagrama, los objetos se colocan en la parte superior y el tiempo avanza de arriba hacia abajo. La línea de vida de un objeto desciende de cada uno de ellos. Un pequeño rectángulo de la línea de vida de un objeto representa una *activación* (la ejecución de una de las operaciones del objeto). Los mensajes (simples, síncronos y asíncronos) son flechas que conectan a una línea de vida con otra. La ubicación del mensaje en la dimensión vertical representará el momento en que sucede dentro de la secuencia. Los mensajes que ocurren primero están más cerca de la parte superior del diagrama, y los que ocurren después cerca de la parte inferior. Un diagrama de secuencias puede mostrar ya sea una instancia (un escenario) de un caso de uso, o puede ser genérico e incorporar todos los escenarios de un caso de uso (Schmuller, 2001).

A continuación se presentan algunos de los diagramas de secuencia de los casos de uso más relevantes, descritos anteriormente.

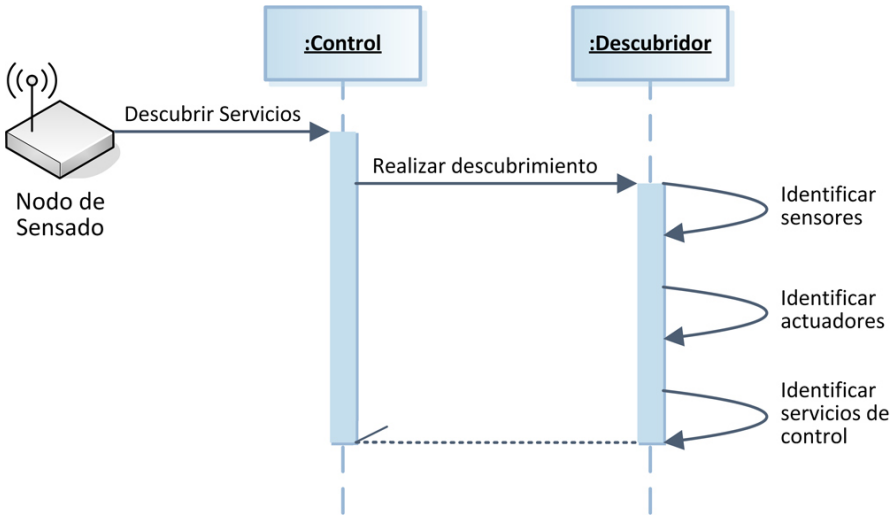


Figura 12: **Diagrama de secuencia para el caso de uso Descubrir Servicios**

La Figura 12 muestra el diagrama de secuencia del caso de uso Descubrir Servicios, el cual es llevado a cabo durante la inicialización del nodo de sensado. El subcomponente Control es el encargado de inicializar las acciones. En este caso, este utiliza el subcomponente Descubridor para solicitar el inicio del descubrimiento de los servicios del nodo. Este a su vez, identifica los dispositivos de sensado, actuadores y las lecturas de control que puede proveer. Para posteriormente devolverle esa información a Control.

La Figura 13 muestra el diagrama de secuencia para el caso de uso Publicar Servicios, el cual se inicia una vez identificados los servicios que el nodo de sensado puede proveer. Posteriormente, a través del subcomponente Comunicación envía un mensaje de registro conteniendo la información de los servicios al nodo *sink*. Para hacerlo, el subcomponente Comunicación utiliza los mecanismos proveídos por el sistema operativo embebido.

La Figura 14 muestra el diagrama de secuencia para el caso de uso Registrar Red, el cual inicia al haberse recibido mensajes de registro de una red. El Procesador de Mensajes es

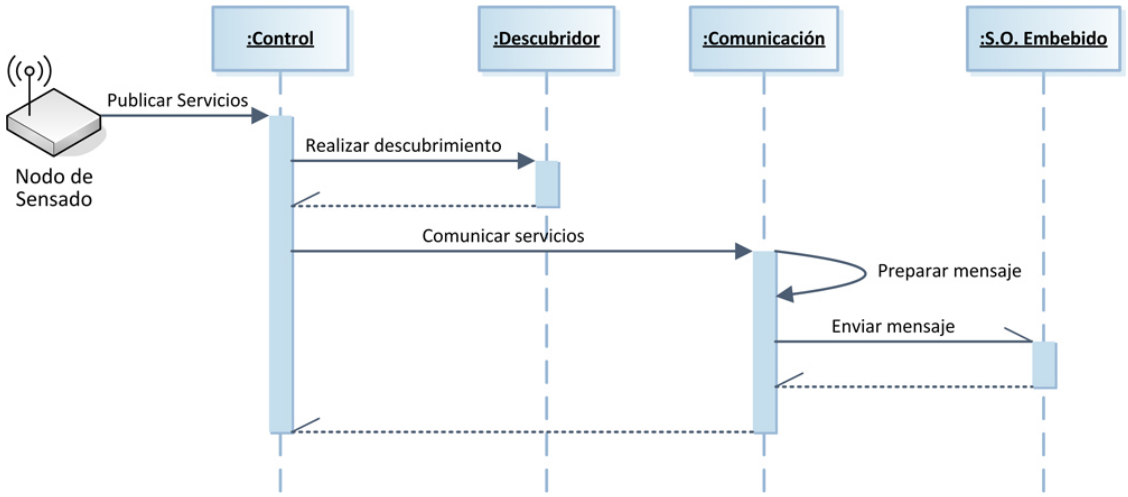


Figura 13: **Diagrama de secuencia para el caso de uso Publicar Servicios**

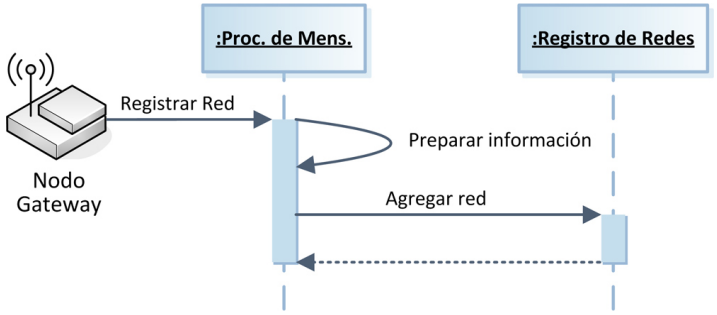


Figura 14: **Diagrama de secuencia para el caso de uso Registrar Red**

el encargado de analizar los mensajes recibidos y de proceder con las acciones correspondientes a estos mensajes. En este caso, procede a preparar la información sobre los servicios que, gracias a los mensajes de registro, sabe que la red provee, para posteriormente agregar un nuevo registro de red, o en su caso, modificar un registro de red (si ésta ya se encontraba registrada) para indicar los servicios que ofrece.

La Figura 15 muestra el diagrama de secuencia para el caso de uso Realizar Suscripción, el cual inicia al haberse registrado los servicios ofrecidos por los nodos de la red. Este caso de uso tiene el objetivo principal de que la red empiece a comunicar la información que capture, para ello, se debe hacer una suscripción a los servicios proveídos por la misma.

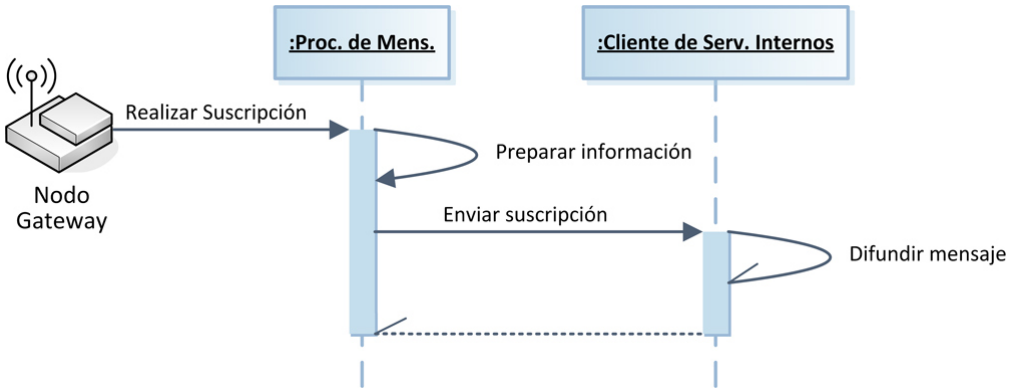


Figura 15: **Diagrama de secuencia para el caso de uso Realizar Suscripción**

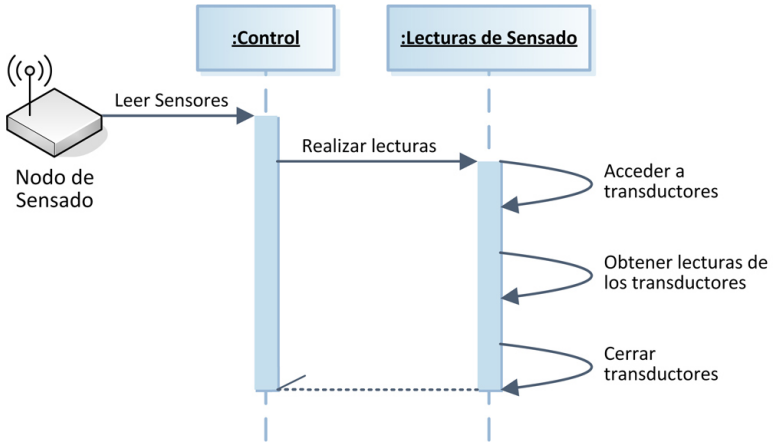


Figura 16: **Diagrama de secuencia para el caso de uso Leer Sensores**

Con este objetivo, el Procesador de Mensajes utilizando el Cliente de Servicios Internos, envía un mensaje de suscripción para los servicios que sabe que la red provee.

La Figura 16 muestra el diagrama de secuencia para el caso de uso Leer Sensores, el cual inicia cuando se desea obtener las lecturas de los mismos. Para lograrlo, el subcomponente Control utiliza la funcionalidad ofrecida por el subcomponente Lecturas de Sensado. Este a su vez, inicializa la interacción con los dispositivos transductores, obtiene las lecturas y finalmente cierra la interacción con los mismos.

La Figura 17 muestra el diagrama de secuencia para el caso de uso Comunicar Lecturas,

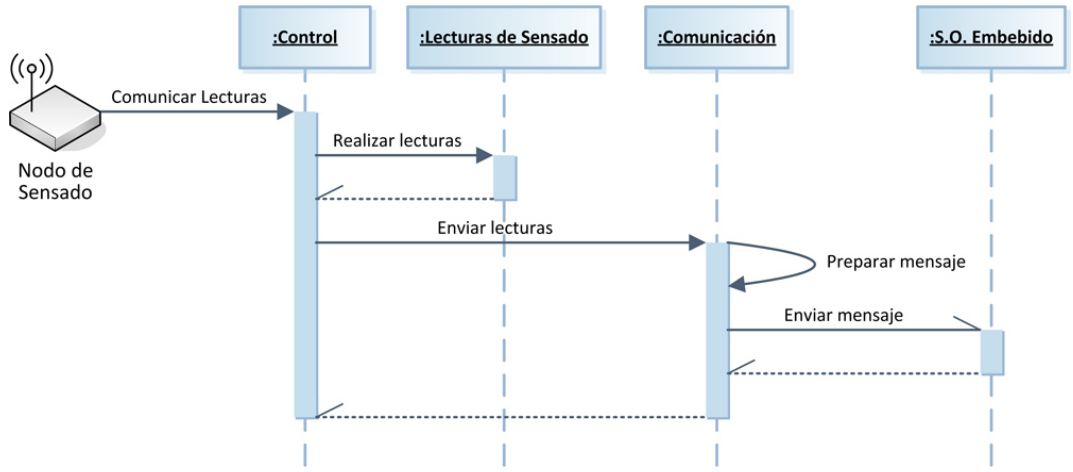


Figura 17: **Diagrama de secuencia para el caso de uso Comunicar Lecturas**

el cual inicia cuando se ha recibido un mensaje de suscripción para alguno de los servicios ofrecidos. Primeramente, el subcomponente Control obtiene las lecturas de los servicios a los que se han suscrito utilizando al subcomponente Lecturas de Sensado. Posteriormente, por medio del subcomponente Comunicación envía las lecturas al nodo *sink*.

La Figura 18 muestra el diagrama de secuencia para el caso de uso Almacenar Lecturas, el cual se inicia al recibirse mensajes de lecturas desde la red. Cuando el subcomponente Procesador de Mensajes recibe un mensaje de lecturas lo interpreta, prepara y registra en el Registro Histórico. Esta información es clasificada como perteneciente a la red para la cual está asignado el componente Gateway que realiza esta acción.

La Figura 19 muestra el diagrama de secuencia para el caso de uso Validar Eventos, el cual se inicia al recibirse mensajes de lecturas de la red y al existir al menos un evento no detectado en el Registro de Eventos. El Procesador de Mensajes además de almacenar las lecturas recibidas, también valida los criterios de aquellos eventos que no han sido detectados. Para hacerlo compara la información recibida hasta el momento con los criterios de estos eventos. En dado caso en que los criterios para un cierto evento sean cumplidos, este es modificado como «detectado» en el Registro de Eventos.

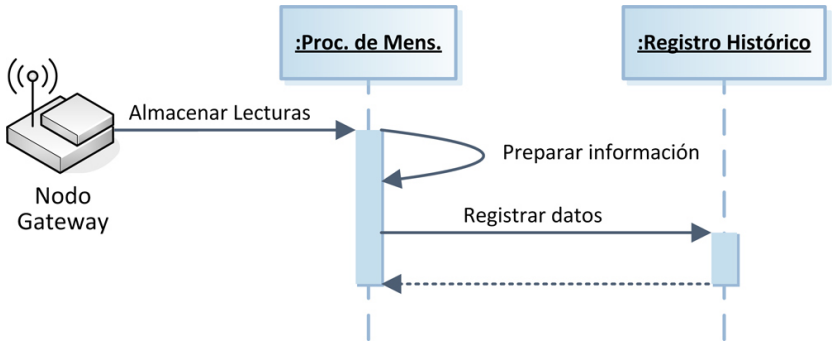


Figura 18: **Diagrama de secuencia para el caso de uso Almacenar Lecturas**

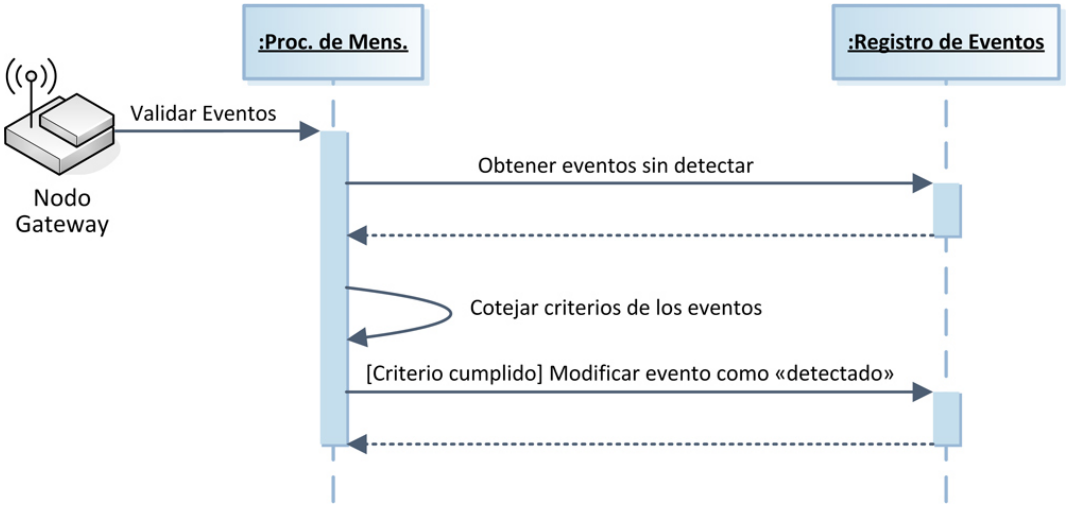


Figura 19: **Diagrama de secuencia para el caso de uso Validar Eventos**

La Figura 20 muestra el diagrama de secuencia para el caso de uso Consultar Redes Registradas, el cual inicia cuando el usuario de servicios externos desea consultar qué redes se encuentran registradas. Para ello, el Proveedor de Servicios Externos utiliza el Registro de Redes para obtener un listado de las redes disponibles en la infraestructura.

La Figura 21 muestra el diagrama de secuencia para el caso de uso Consultar Servicios Ofrecidos por la Red, el cual se inicia cuando el usuario de servicios externos desea consultar información sobre los servicios ofrecidos por una de las redes registradas en la

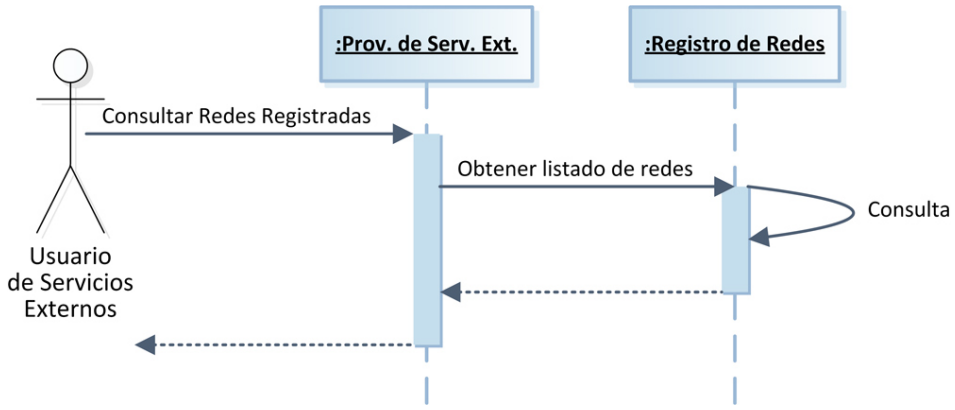


Figura 20: **Diagrama de secuencia para el caso de uso Consultar Redes Registradas**

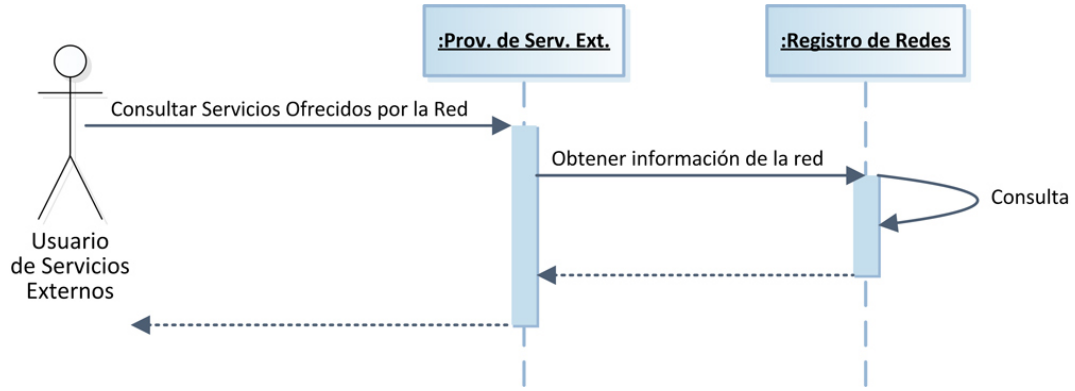


Figura 21: **Diagrama para el caso de uso Consultar Servicios Ofrecidos por la Red**

infraestructura. Para ello, el Proveedor de Servicios Externos hace uso del Registro de Redes para obtener información específica sobre la red para la cual se solicita. Esta información comprende: qué parámetros de sensado es capaz de leer la red, con qué actuadores cuenta, qué información de control está disponible y el rango de fechas para la cual se tiene información.

La Figura 22 muestra el diagrama de secuencia para el caso de uso Consultar Lecturas, el cual se inicia cuando el usuario de servicios externos desea consultar información de sensado. Para ello, el Proveedor de Servicios Externos interactúa con el Registro Histórico para obtener las lecturas de sensado solicitadas por el usuario.

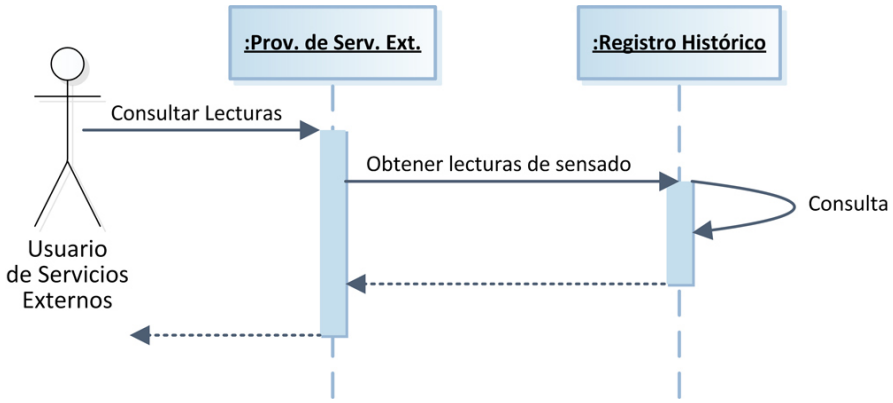


Figura 22: **Diagrama de secuencia para el caso de uso Consultar Lecturas**

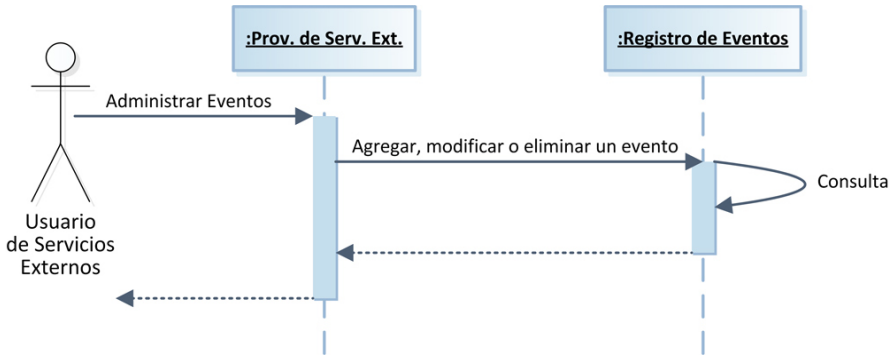


Figura 23: **Diagrama de secuencia para el caso de uso Administrar Eventos**

La Figura 23 muestra el diagrama de secuencia para el caso de uso Administrar Eventos, el cual inicia cuando el usuario de servicios externos desea administrar los eventos registrados. Esta administración puede consistir en agregar un nuevo evento, modificar o eliminar un evento existente. Para ello el Proveedor de Servicios Externos interactúa con el Registro de Eventos.

En el siguiente capítulo se presenta el diseño de un sistema *middleware* prototipo basado en la arquitectura propuesta. El propósito del sistema es comprobar la viabilidad de aplicación de la arquitectura. Además se presenta una pequeña evaluación del sistema prototipo con la que se comprobarán ciertos aspectos de la arquitectura en general.

Capítulo VI

Implementación de un Prototipo

«Una teoría puede probarse mediante experimentos; pero no hay ningún camino que conduzca de los experimentos a la teoría»

– Albert Einstein (1879-1955)

Con el objetivo de comprobar la viabilidad en la aplicación de la arquitectura propuesta se realizó la implementación de un sistema *middleware* prototipo. Su desarrollo permitió conocer eventualidades en la concepción de la arquitectura y obtener información con la cual se mejoró el diseño. En el presente capítulo se describe la implementación de este prototipo de sistema *middleware* para WSN llamado TinySOA.

VI.1. Plataforma de *hardware* utilizada

Para el desarrollo del prototipo, se contó con ocho motes MICAz, siete placas de sensado MTS310CA y una placa programadora MIB510, todos de la compañía Crossbow[1]. Los cuales se describen brevemente a continuación.

MICAz (véase la Figura 24) es un mote para redes de sensores de baja energía. Tiene diversas características en las cuales se mejora el desempeño general de la familia MICA de Crossbow. Este mote incluye: un transmisor/receptor RF compatible con el estándar IEEE 802.15.4/ZigBee; una banda de transmisión de 2.4 a 2.4835 GHz; un espectro ensanchado por secuencia directa resistente a interferencia; una tasa de datos de 250 Kbps; es compatible con TinyOS 1.1.7 o superior; e incluye un conector para placas de sensado, *gateways* y

[1] Crossbow Technology, Inc. `http://www.xbow.com/`

Figura 24: **Mote MICAz (Crossbow Technology, Inc., 2006b)**

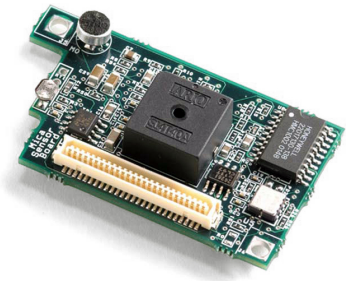


Figura 25: **Placa de sensado MTS310CA (Crossbow Technology, Inc., 2005)**

software de Crossbow. Es alimentado por dos baterías AA y es capaz de funcionar hasta un año en un estado de espera o una semana a una carga de trabajo constante.

MTS310CA (véase la Figura 25) es una placa de sensado con una variedad de modalidades de sensado. Estas modalidades incluyen: un acelerómetro y un magnetómetro, ambos de doble eje, así como un sensor de luminosidad, uno de temperatura, un micrófono y una bocina. Está diseñado para su uso en los motes MICAz y MICA2.

MIB510CA (véase la Figura 26) es una placa programadora que permite la comunicación de datos de una red de sensores a una PC a través de su interfaz serial. Cualquier nodo MICAz o MICA2 puede funcionar como una estación base cuando se encuentra instalada en la placa MIB510CA. Además de la transferencia de datos, también provee una interfaz de

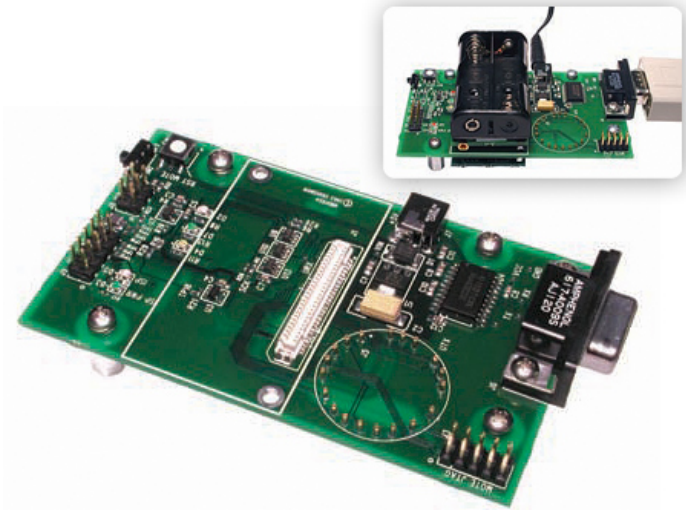


Figura 26: **Placa programadora MIB510CA (Crossbow Technology, Inc., 2006a)**

programación RS-232 serial. MIB510CA tiene un procesador que programa motes MICA2, MICAz y MICA2DOT. El procesador también monitorea el voltaje de energía, no permitiendo la programación si el voltaje no se encuentra en los límites requeridos. También cuenta con dos conectores de 51 pines, lo que permite que sea posible instalar placas de sensado para monitoreo o desarrollo de código. Además es compatible con el conector Atmel JTAG para la depuración de código.

VI.2. Diseño e implementación del prototipo

En esta sección se describen los detalles del diseño e implementación del sistema *middleware* prototipo TinySOA. El cual está basado en la arquitectura presentada en el capítulo anterior. En la Figura 27 se muestra el diagrama global de emplazamiento y de componentes del sistema. Como se puede observar, Gateway, Registro y Servidor se encuentran en la misma computadora, sin embargo, están diseñados para poder residir en localidades diferentes ya que su vínculo es por medio del Registro. En el sistema no se incluyen los componentes de comunicación opcionales entre Gateway y Servidor. Pueden haber variar

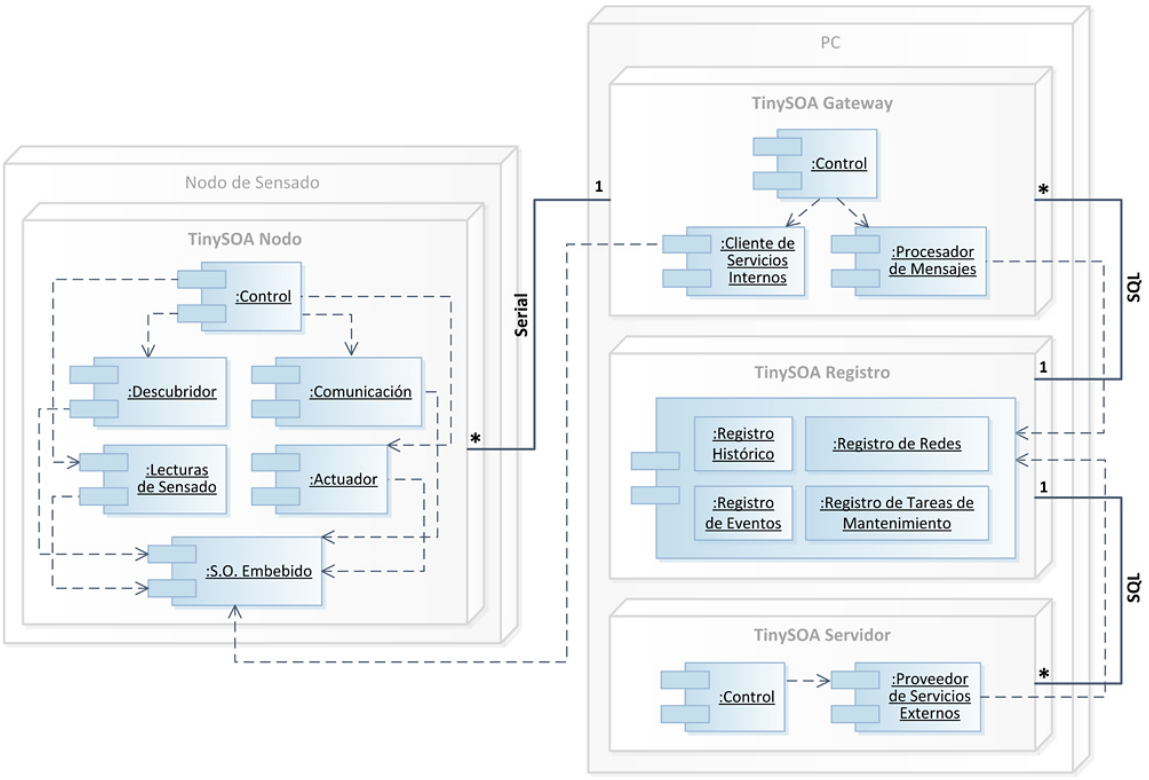


Figura 27: **Diagrama de emplazamiento y de componentes de TinySOA**

redes de sensores (cada una con su Gateway) en el sistema, así como puede haber múltiples instancias de Servidor, pero solamente una de Registro. A continuación se define a detalle el diseño e implementación de cada uno de los elementos de TinySOA.

VI.2.1. TinySOA Nodo

TinySOA Nodo es el *software* en ejecución en cada uno de los nodos de sensado. Para su implementación, se seleccionó el sistema operativo TinyOS (Hill et al., 2000) para servir como abstracción al *hardware* del nodo de sensado. TinyOS se ha convertido en el estándar de facto gracias a su estabilidad, grado de desarrollo y aceptación por varios trabajos de investigación y en desarrollos comerciales. Además provee soporte para una gran variedad de plataformas WSN. Para más información sobre TinyOS, véase el Apéndice A.

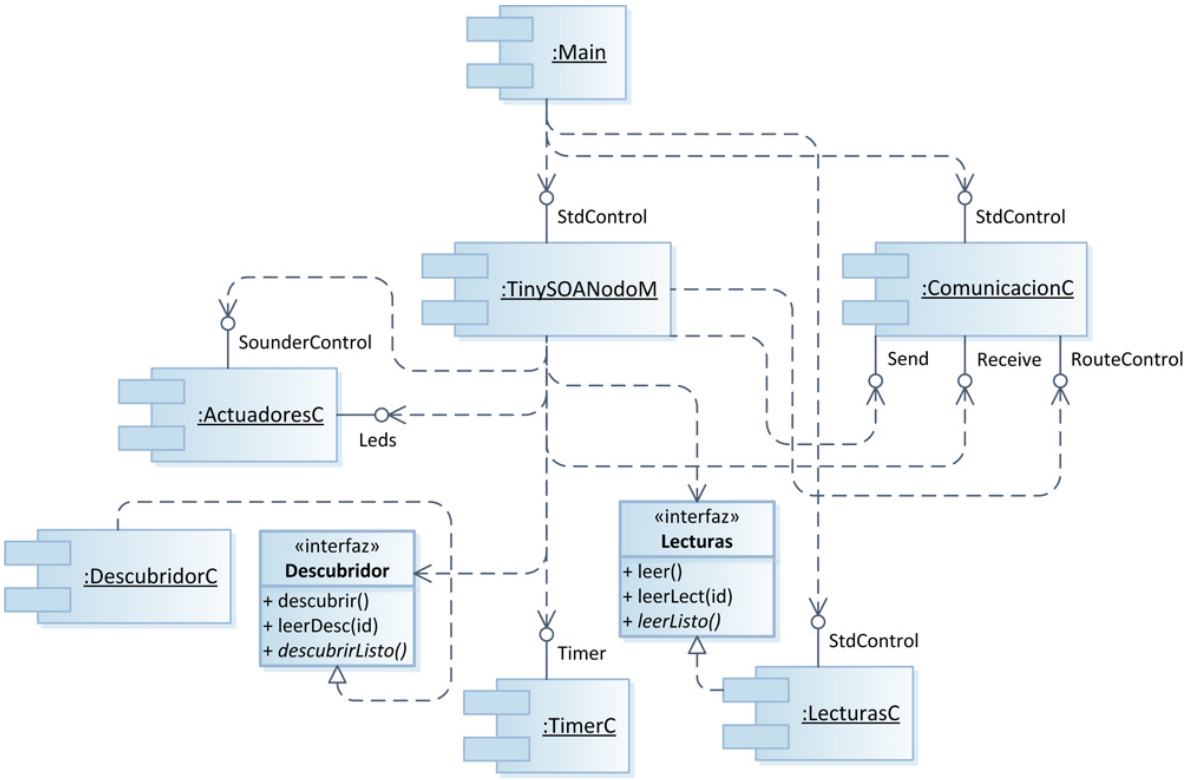


Figura 28: **Diagrama de componentes de TinySOA Nodo**

En la Figura 28 se presenta el diagrama de componentes para TinySOA Nodo. La programación en TinyOS es en base a componentes, simplificando la creación de aplicaciones a la tarea de conectar componentes de sistema y componentes definidos por el usuario. A continuación se describe brevemente cada uno de estos componentes:

- *Main*. Es un componente de sistema, es el encargado de, utilizando la interfaz StdControl, inicializar y controlar la ejecución de la aplicación constituida por los demás componentes en la misma.

- *TinySOANodoM*. Es el componente Control de la arquitectura. Es el encargado de orquestar las actividades de los demás componentes en la aplicación. Al proveer la interfaz StdControl, este puede ser inicializado al encenderse el nodo de sensado.

- *DescubridorC*. Es un componente de usuario. Es el encargado de identificar los sensores disponibles en el nodo. En el prototipo, el descubrimiento de servicios es llevado a

cabo mediante la identificación del tipo de placa de sensado que el nodo de sensado tiene instalada. DescubridorC provee la interfaz de usuario Descubridor, la cual consiste en dos métodos, uno para iniciar el proceso de descubrimiento (`descubrir()`), otra para obtener los resultados (`leerDesc(id)`) y un evento que es señalizado cuando se finaliza el descubrimiento de servicios (*`descubrirListo()`*).

- *LecturasC*. Es un componente de usuario, encargado de leer los dispositivos transductores del nodo. Provee la interfaz Lecturas, en la cual, el método `leer()` inicia el proceso de lectura de los dispositivos disponibles. Al finalizar el proceso, el evento *`leerListo()`* es señalizado. Posteriormente se puede acceder a las lecturas mediante el método `leerLect(id)`, donde `id` es el dispositivo transductor a leer.

- *ComunicacionC*. Es un componente de usuario que encapsula componentes de sistema, los cuales consisten en componentes de envío y recepción de mensajes, control de topología, entre otros. Estos componentes proveen las interfaces Send (para el envío de mensajes), Receive (para la recepción de comandos) y RouteControl (para obtener información de topología). Además se provee la interfaz StdControl para inicializar los componentes de sistema.

- *ActuadoresC*. Es un componente de usuario que encapsula componentes de sistema, que consisten en controladores de los Actuadores disponibles. En este caso se proveen las interfaces SounderControl (para el control de la bocina) y Leds (para el control de los *leds* del nodo de sensado).

- *TimerC*. Es un componente de sistema, que provee la interfaz Timer con la cual se pueden implementar calendarizadores.

Estructura de los mensajes de Nodo a Gateway

En la Figura 29 se puede observar la estructura de los mensajes comunicados por TinySOA Nodo hacia el nodo *sink* (con TinySOA Gateway como destinatario). Estos mensajes

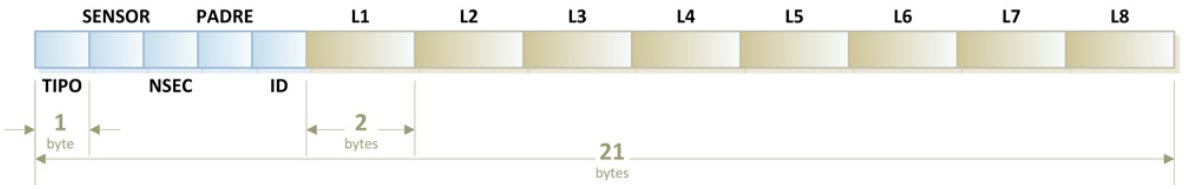


Figura 29: **Estructura de los mensajes TinySOAMsg**

están conformados de la manera siguiente: pueden ser de dos tipos, de registro y de comunicación de lecturas; contienen el número de identificación de la placa de sensado; un número de secuencia; el ID del nodo padre en la topología; el ID del nodo de sensado que envía el mensaje, y ocho contenedores de lecturas o especificaciones de sensado. Se buscó que el tamaño del mensaje fuera lo más pequeño posible, para los campos de control solamente se está utilizando 1 *byte*, típicamente el tamaño de una lectura de un transductor es de 2 *bytes*, se incluyó espacio para 8 lecturas dando un tamaño total de 21 *bytes*. Esta implementación contempla la organización de la red en una forma de árbol.

Cuando se envía un mensaje de registro, además de que el campo `sensor` contiene el número de identificación de la placa de sensado, se especifica, para cada uno de los ocho contenedores, qué parámetro será enviado en ese contenedor cuando se envíen mensajes de lecturas. Por ejemplo, en el caso que se contara con la placa MTS310CA instalada, el campo sensor contendría la constante `SBID_MTS310`, y los valores de los contenedores serían las constantes `TINYSOA_SENSOR_TEMP` (sensor de temperatura), `TINYSOA_SENSOR_LUZ` (luminosidad), etc. De esta manera TinySOA Gateway puede saber qué servicios ofrecen los nodos de sensado (al saber con qué sensores cuenta, gracias a la identificación de la placa de sensado) y cómo, posteriormente, interpretar los mensajes de lecturas. Para un listado de las constantes definidas en el sistema consulte el Apéndice B.

VI.2.2. TinySOA Gateway

TinySOA Gateway es el *software* encargado de la comunicación entre la red de sensores y la infraestructura exterior. Este modulo fue implementado utilizando el lenguaje de

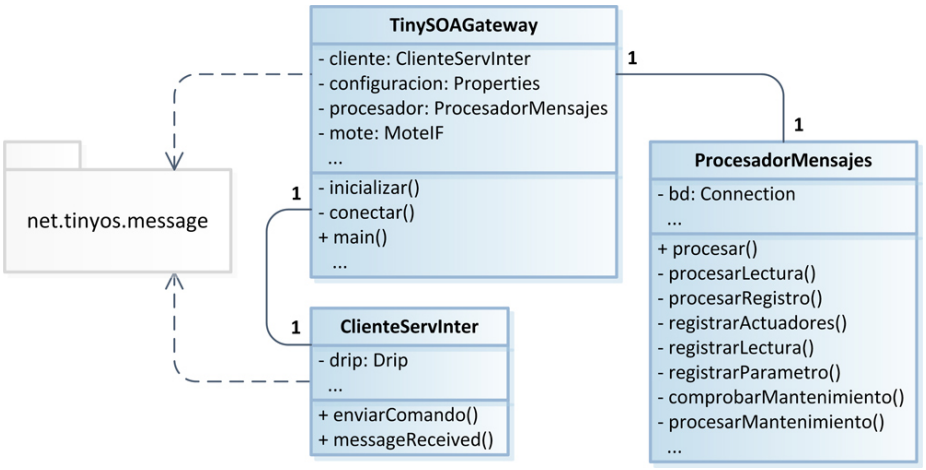
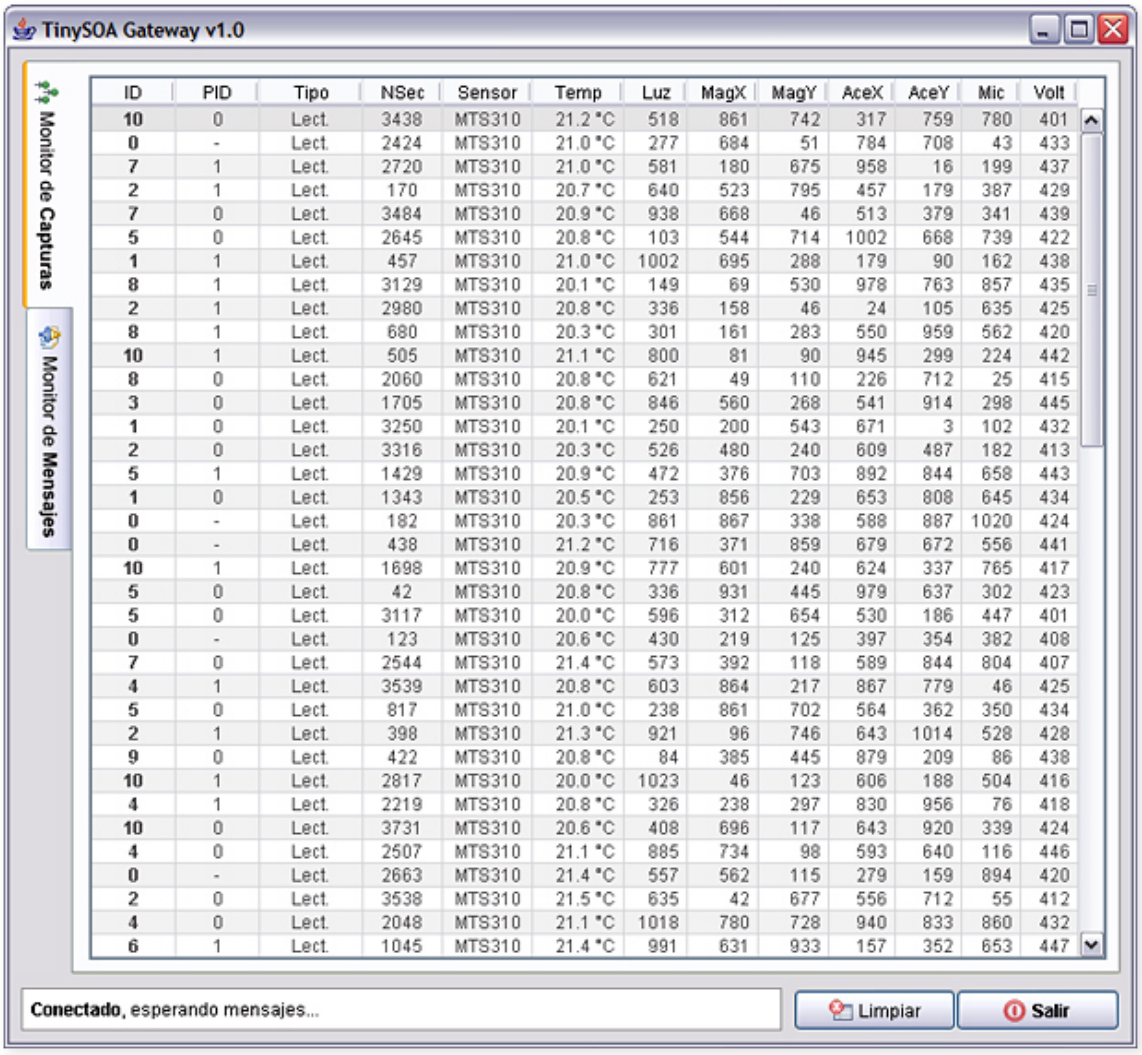


Figura 30: **Diagrama de clases de TinySOA Gateway**

programación Java[2]. La razón para ello es que en la distribución de TinyOS[3] se incluyen herramientas, para su uso por aplicaciones Java, que se encargan de la comunicación con la red de sensores. Facilitando el proceso de recibir, interpretar y procesar mensajes de la red, así como también el de empaquetar y diseminar mensajes en la misma.

En la Figura 30 se puede observar el diagrama de clases de TinySOA Gateway. Básicamente consiste de los módulos indicados por la arquitectura, excepto por el componente opcional de comunicación con el Servidor, el cual no fue implementado para esta prueba. También se puede observar cómo se hace uso de las herramientas brindadas por la distribución de TinyOS (específicamente, en el paquete `net.tinyos.message`) para la recepción (clase MoteIF) y diseminación (clase Drip) de mensajes. Aquí es importante mencionar que la herramienta Drip necesita cierta funcionalidad especial en el nodo *sink* de la red de sensores, esta funcionalidad es proveída por la aplicación TOSBase. Por lo tanto, en la instalación utilizada por este sistema *middleware* en cada uno de los nodos de sensado se encuentra la aplicación TinySOA Nodo, excepto en el nodo conectado a la placa programadora (el nodo *sink*), en el cual se ejecuta la aplicación TOSBase.

[2]Sun Microsystems. `http://java.sun.com/`
[3]TinyOS. `http://www.tinyos.net/`



ID	PID	Tipo	NSec	Sensor	Temp	Luz	MagX	MagY	AceX	AceY	Mic	Volt
10	0	Lect.	3438	MTS310	21.2 °C	518	861	742	317	759	780	401
0	-	Lect.	2424	MTS310	21.0 °C	277	684	51	784	708	43	433
7	1	Lect.	2720	MTS310	21.0 °C	581	180	675	958	16	199	437
2	1	Lect.	170	MTS310	20.7 °C	640	523	795	457	179	387	429
7	0	Lect.	3484	MTS310	20.9 °C	938	668	46	513	379	341	439
5	0	Lect.	2645	MTS310	20.8 °C	103	544	714	1002	668	739	422
1	1	Lect.	457	MTS310	21.0 °C	1002	695	288	179	90	162	438
8	1	Lect.	3129	MTS310	20.1 °C	149	69	530	978	763	857	435
2	1	Lect.	2980	MTS310	20.8 °C	336	158	46	24	105	635	425
8	1	Lect.	680	MTS310	20.3 °C	301	161	283	550	959	562	420
10	1	Lect.	505	MTS310	21.1 °C	800	81	90	945	299	224	442
8	0	Lect.	2060	MTS310	20.8 °C	621	49	110	226	712	25	415
3	0	Lect.	1705	MTS310	20.8 °C	846	560	268	541	914	298	445
1	0	Lect.	3250	MTS310	20.1 °C	250	200	543	671	3	102	432
2	0	Lect.	3316	MTS310	20.3 °C	526	480	240	609	487	182	413
5	1	Lect.	1429	MTS310	20.9 °C	472	376	703	892	844	658	443
1	0	Lect.	1343	MTS310	20.5 °C	253	856	229	653	808	645	434
0	-	Lect.	182	MTS310	20.3 °C	861	867	338	588	887	1020	424
0	-	Lect.	438	MTS310	21.2 °C	716	371	859	679	672	556	441
10	1	Lect.	1698	MTS310	20.9 °C	777	601	240	624	337	765	417
5	0	Lect.	42	MTS310	20.8 °C	336	931	445	979	637	302	423
5	0	Lect.	3117	MTS310	20.0 °C	596	312	654	530	186	447	401
0	-	Lect.	123	MTS310	20.6 °C	430	219	125	397	354	382	408
7	0	Lect.	2544	MTS310	21.4 °C	573	392	118	589	844	804	407
4	1	Lect.	3539	MTS310	20.8 °C	603	864	217	867	779	46	425
5	0	Lect.	817	MTS310	21.0 °C	238	861	702	564	362	350	434
2	1	Lect.	398	MTS310	21.3 °C	921	96	746	643	1014	528	428
9	0	Lect.	422	MTS310	20.8 °C	84	385	445	879	209	86	438
10	1	Lect.	2817	MTS310	20.0 °C	1023	46	123	606	188	504	416
4	1	Lect.	2219	MTS310	20.8 °C	326	238	297	830	956	76	418
10	0	Lect.	3731	MTS310	20.6 °C	408	696	117	643	920	339	424
4	0	Lect.	2507	MTS310	21.1 °C	885	734	98	593	640	116	446
0	-	Lect.	2663	MTS310	21.4 °C	557	562	115	279	159	894	420
2	0	Lect.	3538	MTS310	21.5 °C	635	42	677	556	712	55	412
4	0	Lect.	2048	MTS310	21.1 °C	1018	780	728	940	833	860	432
6	1	Lect.	1045	MTS310	21.4 °C	991	631	933	157	352	653	447

Figura 31: **Captura de pantalla de TinySOA Gateway**

La aplicación TinySOA Gateway fue implementada con una modalidad de depuración mediante una interfaz gráfica, la cual muestra los mensajes recibidos de manera interpretada. En la Figura 31 se puede observar la captura de pantalla de dicha interfaz, así como también algunas de las lecturas recibidas.

Estructura de los mensajes de Gateway a Nodo

Como se mencionaba anteriormente, TinySOA Gateway es el encargado de diseminar mensajes de comando en la red de sensores. Para hacerlo, hace uso de la funcionalidad

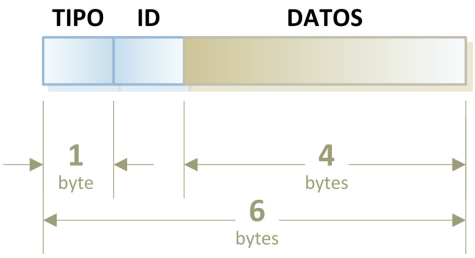


Figura 32: **Estructura de los mensajes TinySOACmdMsg**

proveída por la herramienta Drip. En la Figura 32 se presenta la estructura de estos mensajes. Básicamente consisten en un campo que define el tipo de comando que se envía, el número de identificación del nodo al que va dirigido el mensaje (si es a todos los nodos, su valor es `0`) e información del comando. Por ejemplo, si la orden es encender la bocina, el campo `tipo` estaría definido por la constante `TINYSOA_ACTIVA_ACTUADOR`, `id` sería `0` para que se encienda en todos los nodos de la red y `datos` sería `TINYSOA_ACTUADOR_BOCINA`.

VI.2.3. TinySOA Registro

TinySOA Registro es en realidad una base de datos, en la cual mediante tablas se llevan los registros de redes, histórico, eventos y tareas. Para la implementación del registro, se utilizó el manejador de bases de datos MySQL[4]. En la Figura 33 se muestra el diagrama de base de datos en el cual se pueden observar las características de cada una de las tablas utilizadas para proveer la funcionalidad de los diferentes registros. Las cuales se describen brevemente a continuación:

- *Tabla Redes (Registro de Redes)*. En ella se encuentran registradas las redes que han estado disponibles en la infraestructura. Para cada una de ellas se almacena el nombre y la descripción que fueron indicados durante su registro.

- *Tabla Parámetros*. En esta tabla se encuentra un listado de los parámetros de sensado que puede capturar una red en específico.

[4]MySQL AB. `http://www.mysql.com/`

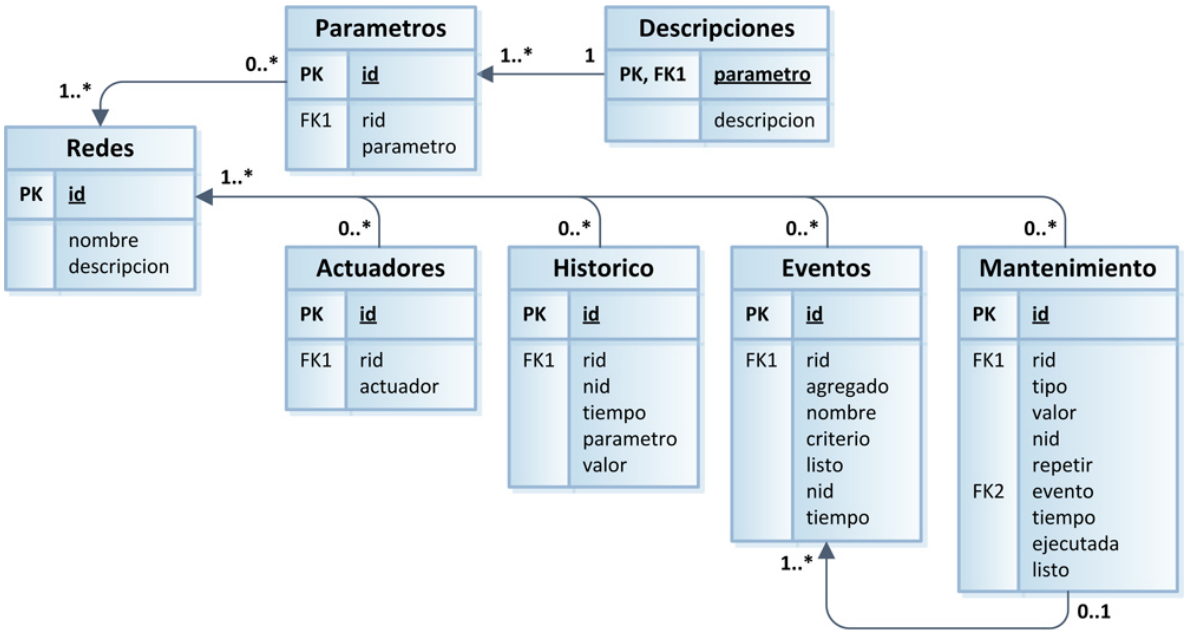


Figura 33: **Diagrama de base de datos para TinySOA Registro**

- *Tabla Descripciones*. En conjunción con la tabla Parámetros, contiene la descripción para los códigos de los parámetros de sensado.

- *Tabla Actuadores*. Similarmente a la tabla Parámetros, almacena un listado de los actuadores disponibles en una red específica.

- *Tabla Histórico (Registro Histórico)*. En esta tabla se encuentran almacenadas las lecturas de sensado que se han reportado para cada una de las redes. Cada registro de esta tabla contiene la lectura de un parámetro de sensado. Para cada lectura se almacenan: el número de identificación del nodo fuente, la fecha y hora en la que ésta fue recibida, el parámetro al que corresponde la lectura y el valor de la misma (ya convertida en unidades de ingeniería).

- *Tabla Eventos (Registro de Eventos)*. En ella se encuentran registrados los eventos para los cuales el usuario ha solicitado su detección. Para cada evento se almacena: la fecha y hora en la que fue agregado, una etiqueta de identificación (nombre), el criterio que debe ser cumplido con base en la información de las lecturas que se van

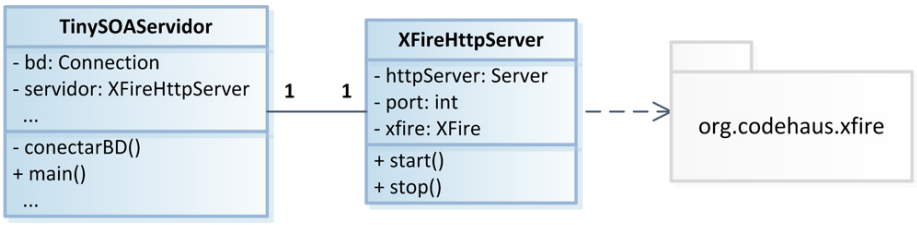


Figura 34: **Diagrama de clases de TinySOA Servidor**

recibiendo, una bandera que indica si éste fue detectado, el número de identificación del nodo que lo provocó y la fecha y hora en la que ocurrió.

- *Tabla Mantenimiento (Registro de Tareas de Mantenimiento)*. En esta tabla se almacenan, para cada una de las redes, las tareas de mantenimiento que se desean enviar a la red de sensores. Para cada una de las tareas se almacena: el tipo de la acción a realizar, la información necesaria para llevar a cabo la tarea, el número de identificación del nodo destinatario (0 si es para todos los nodos de la red), el número de minutos en el que la tarea debe ser realizada de nuevo a partir de que inicie (0 si no se desea repetir la acción), el nombre del evento al que se debe esperar para enviar la tarea (vacío si no se desea esperar a la detección de un cierto evento), la fecha y hora en la que se desea que inicie la tarea, la fecha y hora en la que fue ejecutada la tarea y una bandera que indica si esta ya fue finalizada.

VI.2.4. TinySOA Servidor

TinySOA Servidor es un servidor de servicios web mediante los cuales se provee acceso a la funcionalidad de la infraestructura. Este servidor fue implementado con ayuda del *framework* SOAP Codehaus XFire[5] para Java. El API de XFire, de manera transparente, permite el uso de una gran variedad de estándares (SOAP, WSDL, entre otros) en el desarrollo de aplicaciones que siguen las pautas de la orientación a servicios. En la Figura 34 se muestra el diagrama de clases para TinySOA Servidor.

[5]Codehaus XFire. `http://xfire.codehaus.org/`

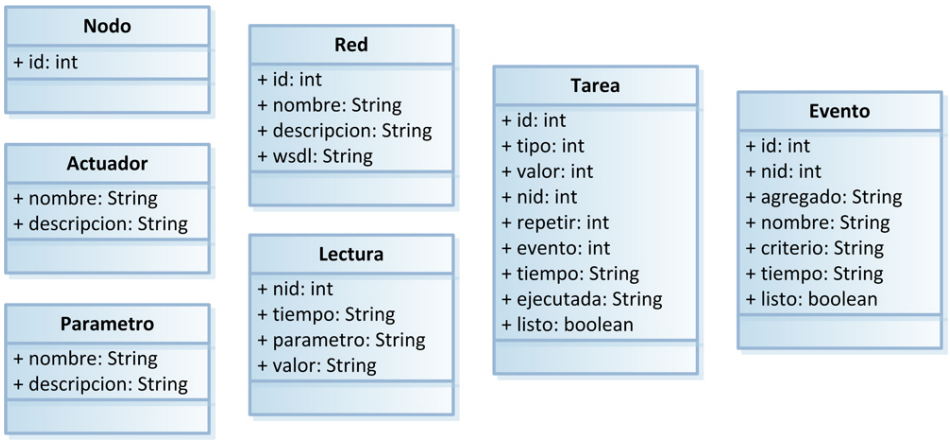


Figura 35: **Diagrama de clases de los objetos devueltos por los servicios**

Los servicios web que provee este servidor consisten en un servicio web de directorio, para la consulta del listado de las redes que se encuentran disponibles en el sistema, y un servicio web de red por cada una de las redes en el sistema.

Para implementar estos servicios, se programaron dos clases que encapsulan la funcionalidad de los dos tipos de servicios. La clase InfoServ para el servicio web de directorio y la clase RedServ para el servicio web de red. Posteriormente, se hizo uso de XFire. Con el que cualquier clase Java, después de agregarles un poco de información extra, son expuestas como servicios web, construyendo automáticamente los documentos WSDL necesarios para su enlazamiento por los usuarios y encargándose del procesamiento de solicitudes y respuestas. La información devuelta por los métodos de los servicios es devuelta en forma de un objeto individual o como un vector de objetos. El diagrama de clases para los distintos objetos que pueden ser devueltos es presentado en la Figura 35.

A continuación se describen los métodos de los servicios InfoServ y RedServ, métodos que comprenden el API ofrecido por TinySOA para la construcción de aplicaciones WSN de monitoreo y/o visualización.

API proveído por el servicio web de directorio

`Vector<Red> obtenerListadoRedes()`

Obtiene un vector de objetos Red conteniendo la información de cada una de las redes de sensores disponibles en la infraestructura.

API proveído por el servicio web de red

`int obtenerIdRed()`

Regresa el ID de la red.

`String obtenerNombreRed()`

Regresa el nombre de la red.

`String obtenerDescripcionRed()`

Regresa la descripción de la red.

`String obtenerTiempoMinimo()`

Regresa la fecha y hora mínima para la cual se tienen lecturas disponibles.

`String obtenerTiempoMaximo()`

Regresa la fecha y hora máxima par la cual se tienen lecturas disponibles.

`Vector<Nodo> obtenerListadoNodos()`

Regresa un vector con la información de los nodos de sensado en la red.

`Vector<Actuador> obtenerActuadores()`

Regresa un vector con información de los actuadores disponibles en la red.

`Vector<Parametro> obtenerParametros()`

Regresa un vector con información de los parámetros de sensado disponibles.

`Vector<Lectura> obtenerLecturasAlTiempo(String tiempo, int limite)`

Regresa un vector con las lecturas disponibles hasta la fecha y hora indicados por

`tiempo`, regresando todas las lecturas si `limite` es igual a `0`, o bien, un número `limite` de las últimas lecturas.

`Vector<Lectura>` **`obtenerUltimasLecturas()`**

Regresa un vector con la última lectura de cada uno de los nodos en la red.

`Vector<Lectura>` **`obtenerLecturas(`**`String` **`desde`**`, String` **`hasta`**`,`
`String` **`parametro`**`, int` **`limite`**`)`

Regresa un vector con las lecturas recibidas en el rango de tiempo indicado por `desde` y `hasta`, regresando todas las lecturas si `limite` es igual a `0`, o bien, un número `limite` de lecturas más recientes para el tiempo `hasta`. De manera similar, se pueden regresar las lecturas para un parámetro determinado, si `parametro` es una cadena vacía se regresan valores para todos los parámetros.

`Vector<Evento>` **`obtenerListadoEventos(`**`int` **`limite`**`)`

Regresa un vector con la información de los eventos, regresando todos los eventos si `limite` es igual a `0`, o bien, un número `limite` de los últimos eventos.

`Evento` **`obtenerEventoPorId(`**`int` **`id`**`)`

Regresa la información de un evento específico.

`boolean` **`agregarEvento(`**`String` **`nombre`**`, String` **`criterio`**`)`

Agrega un nuevo evento al registro de eventos. Regresando `falso`, si existe un error en la especificación del criterio, o `verdadero` si el evento fue agregado correctamente.

`boolean` **`modificarEvento(`**`int` **`id`**`, String` **`nombre`**`, String` **`criterio`**`)`

Modifica un evento, especificado por `id`, del registro eventos. Regresando `falso`, si existe un error en la especificación del criterio, o `verdadero` si el evento fue modificado correctamente.

`void` **`eliminarEvento(`**`int` **`id`**`)`

Elimina el evento especificado por `id`.

`Vector<Tarea>` **`obtenerListadoTareas`**`(int` **`limite`**`)`

Regresa un vector con la información de las tareas de mantenimiento en el registro, regresando todas las tareas si `limite` es igual a `0`, o bien, un número `limite` de las últimas tareas registradas.

`Tarea` **`obtenerTareaPorId`**`(int` **`id`**`)`

Regresa la información de una tarea de mantenimiento específica.

`boolean` **`agregarTarea`**`(int` **`tipo`**`, int` **`valor`**`, int` **`destino`**`,`
`String` **`tiempo`**`, int` **`repetir`**`, int` **`evento`**`)`

Agrega una nueva tarea de mantenimiento al registro. Regresando `falso`, si existe un error en la especificación de la tarea, o `verdadero` si fue agregada correctamente.

`boolean` **`modificarTarea`**`(int` **`id`**`, int` **`tipo`**`, int` **`valor`**`, int` **`destino`**`,`
`String` **`tiempo`**`, int` **`repetir`**`, int` **`evento`**`)`

Modifica una tarea de mantenimiento del registro, especificada por `id`. Regresando `falso`, si existe un error en la especificación de la tarea, o `verdadero` si fue modificada correctamente.

`void` **`eliminarTarea`**`(int` **`id`**`)`

Elimina la tarea especificada por `id`.

VI.3. TinySOA Visor

Durante el diseño del sistema prototipo TinySOA, se implementó una aplicación de monitoreo y visualización con el objetivo de hacer una comprobación continua de la funcionalidad del sistema y de la aplicación de la arquitectura. Este sistema, conocido como TinySOA Visor, hace uso exclusivamente de los servicios proveídos por TinySOA para la interacción con las redes de sensores. Además, sus diferentes características cubren toda la funcionalidad proveída por los servicios. Características que son descritas a continuación.

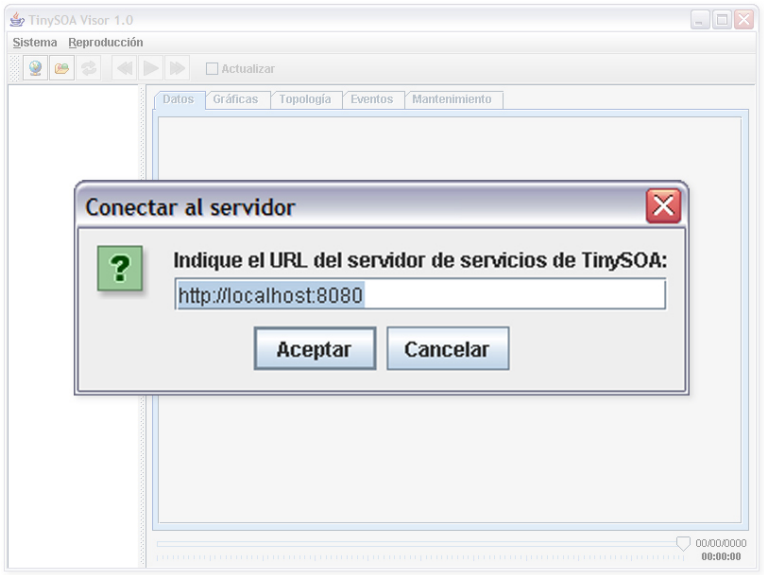


Figura 36: **TinySOA Visor: Diálogo para especificar el URL de TinySOA Servidor**

En la Figura 36 se puede observar el cuadro de diálogo que se muestra al iniciar TinySOA Visor. En este diálogo se debe especificar la dirección URL del servidor de servicios externos (TinySOA Servidor) que se desee utilizar.

Una vez especificado el URL del servidor, la aplicación procede a localizar y utilizar el servicio de directorio. Con el que, obteniendo un listado de las redes de sensores registradas en la infraestructura mediante el uso del método `obtenerListadoRedes()`, presenta un cuadro de diálogo que permite al usuario seleccionar la red que se desea monitorear o visualizar (véase la Figura 37). En este listado se puede consultar el nombre, la descripción y el URL del servicio web de red para cada una de las redes registradas.

Cuando el usuario ya ha seleccionado una red, la aplicación obtiene un enlace con el servicio web de la red seleccionada. Una vez realizada la conexión, se muestra la pantalla principal de TinySOA Visor (véase la Figura 38). Del lado izquierdo de la pantalla se puede observar el listado de los nodos que componen la red, este listado es obtenido a través del método `obtenerListadoNodos()`. La información que es mostrada en el centro

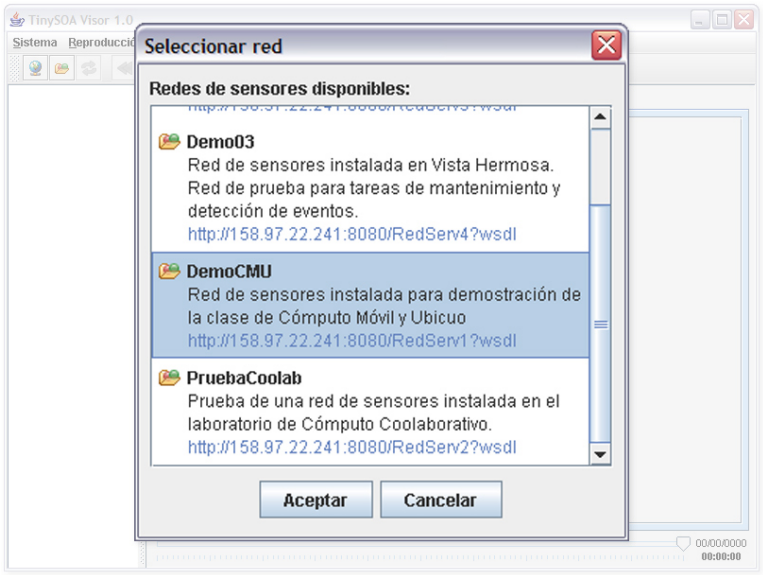


Figura 37: **TinySOA Visor: Diálogo para selección de red de sensores**

de la pantalla es filtrada según los nodos que estén seleccionados en esta lista. En la parte inferior de la pantalla se encuentra la escala de tiempo, los límites de esta escala están definidos por las fechas y horas recibidas al usar los métodos `obtenerTiempoMinimo()` y `obtenerTiempoMaximo()`. La información mostrada en el centro de la pantalla es la más reciente para la fecha y hora seleccionada en la escala. Además, la selección puede ser animada (utilizando los botones de reproducción en el menú), o también puede activarse la modalidad de monitoreo de la aplicación haciendo clic en el cuadro de selección «Actualizar» (que se encuentra a la derecha del menú). En el área del centro de la pantalla se muestra una de las tres modalidades de visualización de datos (tabla, gráfica o topología), la tabla de eventos o la tabla de tareas de mantenimiento.

En la Figura 38 se muestra la vista de datos. Esta vista consiste en una tabla con una fila por cada nodo en la red. En las columnas se encuentran parámetros de sensado (obtenidos con `obtenerParametros()`) e información de control. La información para esta tabla es obtenida mediante el uso del método `obtenerLecturasAlTiempo()`.

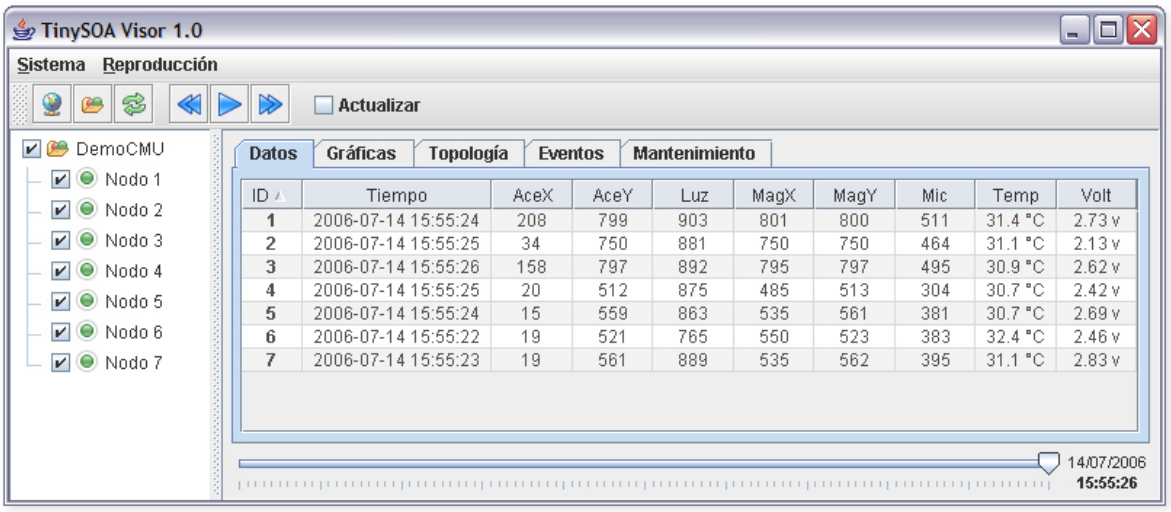


Figura 38: **TinySOA Visor: Vista de datos**

En la Figura 39 se muestra la vista de gráfica. En esta vista se puede obtener una gráfica de la información de los diferentes nodos que componen la red por parámetro de sensado. En la figura se puede apreciar una gráfica para los valores de temperatura.

En la Figura 40 se muestra la vista de topología. En esta vista se puede observar una gráfica de color de acuerdo a los valores de un cierto parámetro de sensado. La posición de los nodos es indicada manualmente, además se puede añadir una imagen al fondo. En la figura se puede observar una gráfica de color del parámetro temperatura y un mapa del laboratorio donde fue instalada una red de prueba como imagen de fondo.

TinySOA Visor además de las características de visualización de datos, también permite administrar los eventos en la infraestructura. En la ficha «Eventos», se puede observar una tabla de los eventos registrados, obtenidos con el método `obtenerListadoEventos()`. En la Figura 41 se puede observar el diálogo para registrar un nuevo evento. Para llevarlo a cabo, la aplicación hace uso del método `agregarEvento()`. Para modificar o eliminar un evento se hace uso de `modificarEvento()` y `eliminarEvento()` respectivamente.

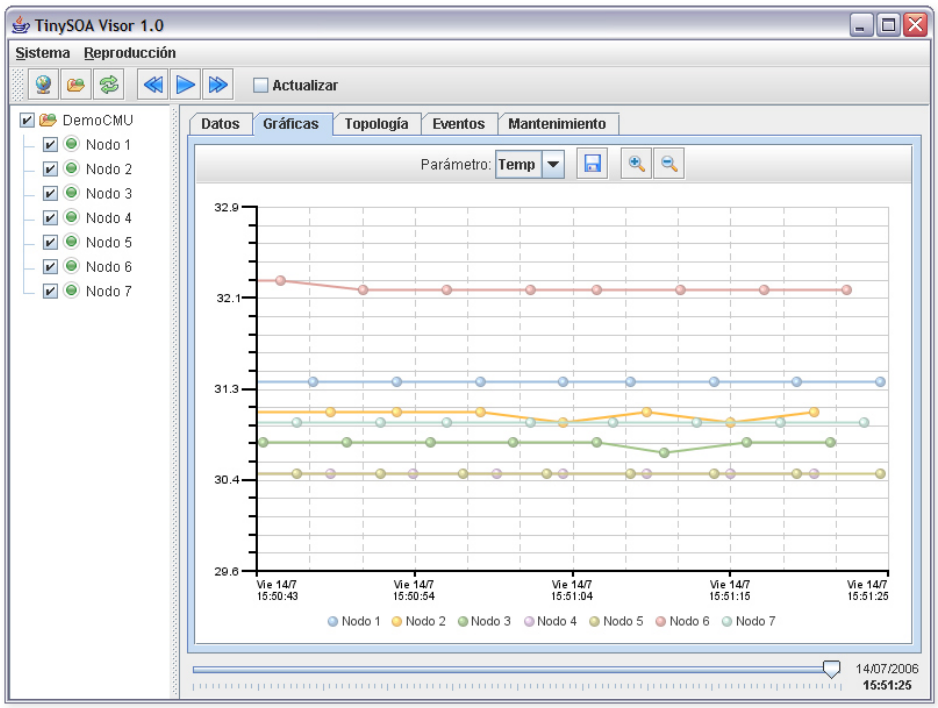


Figura 39: **TinySOA Visor: Vista de gráfica**

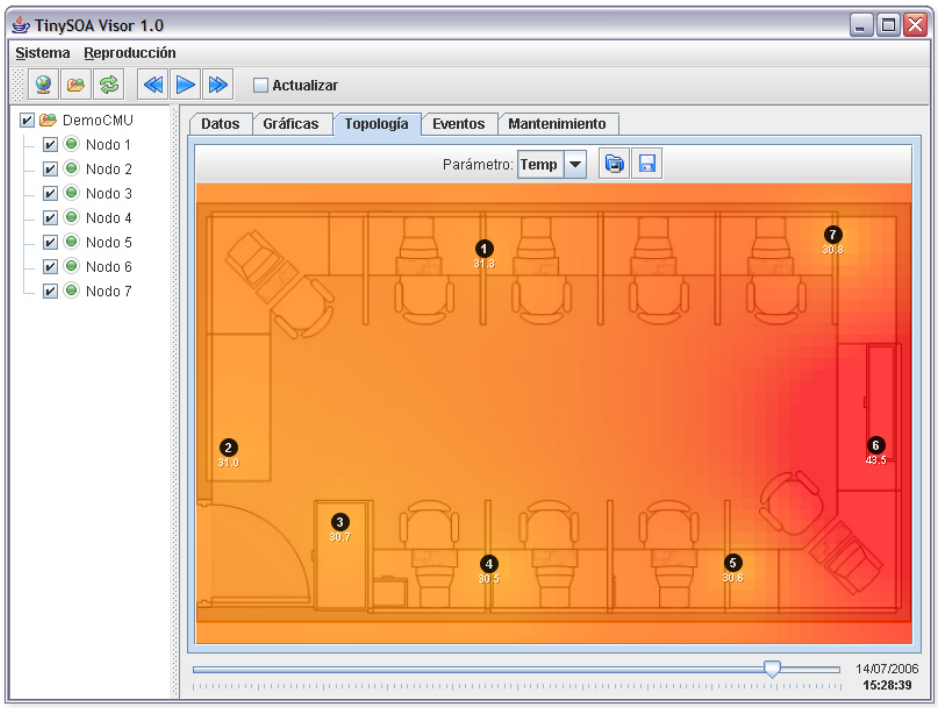


Figura 40: **TinySOA Visor: Vista de topología**

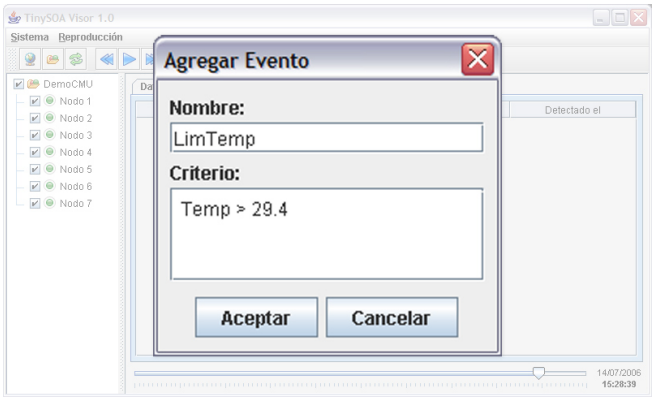


Figura 41: **TinySOA Visor: Administración de eventos**

De manera similar a los eventos, en la ficha «Mantenimiento» se puede observar la tabla de las tareas de mantenimiento registradas (actualizada por el método `obtenerListadoTareas()`). En la Figura 42 se presenta el diálogo para agregar una nueva tarea de mantenimiento, y las tres modalidades de tarea que existen en el sistema (encendido/apagado de actuadores, cambio de tasa de muestreo y entrada/salida del estado de espera).

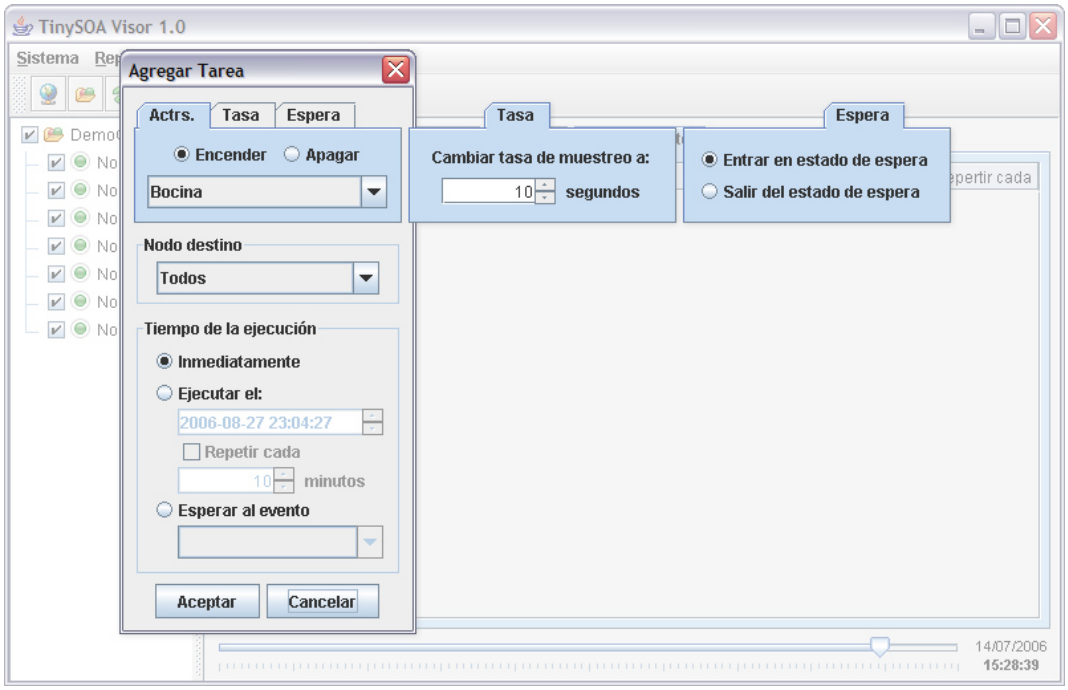


Figura 42: **TinySOA Visor: Administración de tareas de mantenimiento**

Capítulo VII

Evaluación

«*Quien nunca ha cometido un error*
nunca ha probado algo nuevo»
– Albert Einstein (1879-1955)

En este capítulo se presenta la evaluación de este trabajo, la cual fue conducida de dos maneras: la primera mediante un experimento de evaluación con un grupo de usuarios potenciales. Este experimento, en el que se utilizó el prototipo *middleware* TinySOA, tuvo como objetivo principal el estimar el grado en el que la arquitectura es capaz de ayudar a los desarrolladores de aplicaciones WSN; la segunda se efectuó por medio de una evaluación funcional de la arquitectura, en la que los requerimientos del diseño fueron cotejados con la solución propuesta, esto con el objetivo de verificar que efectivamente se está ofreciendo una solución que logra apoyar estos requerimientos.

VII.1. Evaluación del prototipo

Con el fin de poder estimar el grado en el que el sistema, y por ende la arquitectura, es capaz de ayudar a los desarrolladores de aplicaciones WSN, así como también verificar que algunos de los requerimientos de la arquitectura hayan sido suficientemente cumplidos, se llevó a cabo un experimento de evaluación. El diseño y los resultados del mismo son presentados en esta sección. Para ello, primeramente se inicia describiendo el diseño del experimento, en donde se describe el escenario en el cual se efectuó, las variables medidas y la mecánica de las sesiones de evaluación. Para continuar con una descripción de la

población de participantes del experimento. Finalmente, se discuten los resultados obtenidos, los cuales son utilizados en la validación de ciertos aspectos en la evaluación funcional de la arquitectura, que se presenta en la sección VII.2.

VII.1.1. Diseño del experimento

Para el desarrollo del experimento se contó con la participación de 13 personas, todos estudiantes de posgrado, de sexo masculino en un rango de edad entre 23 y 34 años, todos con previa experiencia en el desarrollo de aplicaciones tradicionales. La población incluyó personas que ya contaban con cierta experiencia con las WSN y personas noveles en el área. En el experimento se les especificó una tarea, la cual consistía en la programación de una sencilla aplicación que hiciera uso de los servicios ofrecidos por TinySOA. Posteriormente se les solicitó que contestaran un cuestionario, cuyos resultados son discutidos al final de la presente sección.

Hipótesis

Para guiar el proceso de evaluación, se definieron las siguientes hipótesis:

- ***H1:*** *El API proveído por el sistema es de fácil uso, además provee una abstracción adecuada y sencilla para el desarrollo de aplicaciones WSN de monitoreo y/o visualización.*

  En esta hipótesis se establece que el API proveído por el sistema puede ser utilizado por los desarrolladores de aplicaciones WSN sin mayor problema, esto debido a que gracias al uso de los servicios web los desarrolladores pueden seguir utilizando las mismas herramientas y paradigmas que utilizaban habitualmente en el desarrollo de aplicaciones comunes. Además establece que la funcionalidad proveída es suficiente para el diseño de nuevas aplicaciones de monitoreo y/o visualización.

- ***H2:*** *El sistema permite la fácil integración con otras herramientas.*

  Esta hipótesis establece que los mecanismos ofrecidos por el sistema son suficientes para lograr una fácil integración con otros sistemas e incluso con otras WSN.

- ***H3:*** *El sistema permite la fácil extensión del* software *en el nodo de sensado.*

  Es decir, el sistema está diseñado de tal manera que si se desea agregar nueva funcionalidad, por ejemplo nuevos tipos de sensores, pueda ser realizado fácilmente.

- ***H4:*** *Es sencillo reconfigurar el sistema para su uso en otros escenarios.*

  Esta última hipótesis se refiere a que es sencillo el adaptar la funcionalidad del sistema para poder utilizarlo en diferentes escenarios de aplicación. Por ejemplo, al poder cambiar la tasa de datos, el tipo de datos que se recolectan, etc.

Variables del experimento

Como variable independiente se definió el lenguaje y entorno de desarrollo utilizado en el ejercicio. Cada uno de los desarrolladores utilizó el entorno de su preferencia. Siendo seleccionado por la mayoría aquél que habitualmente utilizan en sus actividades de desarrollo de aplicaciones.

Las variables dependientes son el tiempo que tomó al desarrollador crear una aplicación de monitoreo WSN con el sistema y cuántas líneas de código le llevó hacerlo. Estas variables fueron medidas durante el transcurso de las sesiones y al final de las mismas, respectivamente.

Al término del experimento se aplicó un cuestionario (véase el Apéndice C), el cual está orientado a medir la percepción de los desarrolladores hacia el sistema y obtener sus comentarios respecto a su experiencia general en el desarrollo de aplicaciones WSN utilizando TinySOA. Además, el cuestionario mide el nivel de aceptación de los usuarios sobre la tecnología. Para esto se utilizó el formato de cuestionario propuesto en el Modelo de Aceptación de Tecnología (TAM por sus siglas en inglés), el cual consiste de tres factores: la utilidad de la aplicación, la percepción de facilidad de uso y la intención de uso (Davis y Venkatesh, 1996).

Figura 43: **Aplicación a ser desarrollada en el experimento**

Tarea a realizar

La tarea a realizar consistió en desarrollar una aplicación sencilla, tal y como se muestra en la Figura 43. Esta aplicación consistiría de solamente un botón y una etiqueta. Al hacer clic sobre el botón, la aplicación utilizaría el servicio web de red de una red de sensores para obtener la última lectura de temperatura registrada por cada uno de sus nodos (método `obtenerUltimasLecturas()`), calcularía el promedio y lo desplegaría en la etiqueta.

Condiciones del experimento

Para el experimento se utilizaron computadoras de escritorio con acceso a Internet y con la instalación necesaria para el desarrollo de aplicaciones (Java[1] y .NET[2]). Estas computadoras se encontraban en un laboratorio. Mientras tanto, en un laboratorio diferente, se encontraba instalada y en funcionamiento una red de sensores con el sistema TinySOA en ejecución. Un servidor en ese mismo laboratorio (conteniendo el *software* TinySOA Gateway, Registro y Servidor) era utilizado por los desarrolladores en el otro laboratorio.

Procedimiento

Se efectuó una sola sesión de evaluación. Durante su inicio, se introdujo brevemente a los participantes al área de redes inalámbricas de sensores. Posteriormente se les explicó el sistema TinySOA, indicando para ello los componentes que lo conforman, los servicios que

[1] Sun Microsystems. `http://java.sun.com/`
[2] Microsoft Corporation. `http://msdn.microsoft.com/vstudio/`

Familiarización con:	Expertos		Noveles		Ambos	
	Media	Desv. Est.	Media	Desv. Est.	Media	Desv. Est.
Programación orientada a objetos.	100%	0%	93%	8%	95%	7%
Desarrollo de aplicaciones con servicios web.	43%	49%	67%	23%	65%	31%
Área de redes inalámbricas de sensores.	90%	8%	56%	21%	64%	24%
Desarrollo de aplicaciones WSN.	81%	16%	29%	21%	41%	30%

Tabla I: **Resumen del perfil de los desarrolladores participantes**

éste ofrece y cómo utilizarlo en el desarrollo de aplicaciones WSN. Una vez realizada la introducción, se procedió a indicarles la tarea a realizar. Para lo cual los participantes utilizaron el lenguaje y el entorno de programación de su preferencia. Se les brindó apoyo al proveerles de la documentación de TinySOA, en la que se definen cada uno de los servicios ofrecidos, los métodos que posee cada uno de ellos y las clases de los objetos que son devueltos por los mismos.

VII.1.2. Descripción de la población de participantes

En el ejercicio participaron 13 personas, de los cuales 11 son estudiantes de maestría y 2 de doctorado. En el cuestionario presentado al finalizar el experimento, se encontraba una sección sobre el perfil del participante. Los resultados de esa sección permitieron dividir a los participantes en dos grupos, los desarrolladores «expertos» y los «noveles».

Los desarrolladores «expertos» están definidos como aquellas personas que manifestaron tener un conocimiento sobre el área de redes inalámbricas de sensores en más del 80 % y en la programación de las mismas en un 50 %.

En el mismo cuestionario se preguntaba el nivel de conocimiento del desarrollador en el paradigma orientado a objetos y en cuanto a qué tanta experiencia tenían en el desarrollo de aplicaciones utilizando servicios web. Los resultados a estos factores pueden ser observados en la Tabla I. De igual manera, se identificaron los lenguajes de programación en los que están familiarizados los desarrolladores (para más detalles véase la Figura 44),

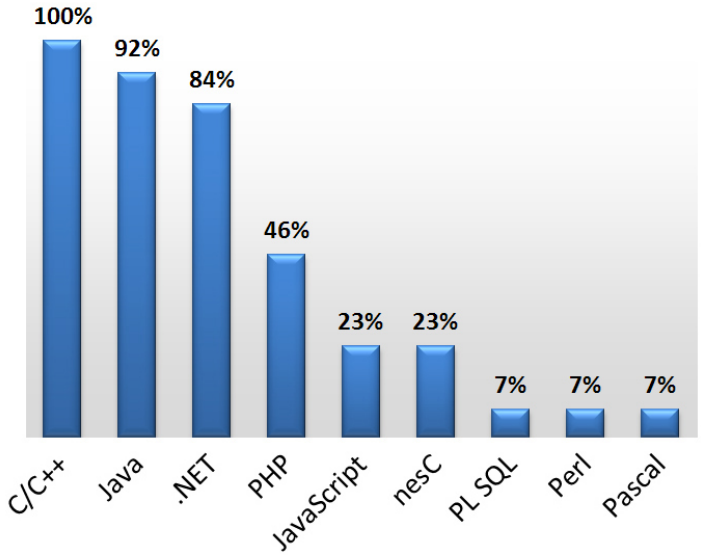


Figura 44: **Dominio de lenguajes de programación de los participantes**

siendo Java y .NET los dos lenguajes que fueron seleccionados por los mismos para el desarrollo de la tarea de la evaluación.

VII.1.3. Resultados y discusión

Con el objetivo de obtener información que permitiera comprobar la validez de las cuatro hipótesis, el cuestionario fue compuesto de preguntas en cinco grupos. Los cuatro primeros para cada una de las hipótesis (5, 2, 1 y 2 preguntas respectivamente) y el último, basado en TAM, nos permitió medir la aceptación general del sistema, en cuanto a facilidad de uso (4 preguntas), utilidad (3 preguntas) e intención de uso (2 preguntas).

Las 19 oraciones están planteadas utilizando una escala Likert (Likert, 1932) con un rango desde «completamente en desacuerdo» (con un valor de 1) hasta «completamente de acuerdo» (de valor 7). Una escala Likert es un tipo de preguntas en donde se pide que se califique el nivel en el cual se está de acuerdo o no con una cierta oración. En las Tablas II-VI se muestra el agrupamiento de preguntas por cada uno de los primeros cuatro grupos.

La **abstracción** ofrecida por TinySOA:	Expertos		Noveles		Ambos	
	Media	Desv. Est.	Media	Desv. Est.	Media	Desv. Est.
El API proveído por TinySOA es de fácil uso.	7.00	0.00	6.30	0.67	6.46	0.66
Puedo utilizar el API de TinySOA de manera nativa a mi entorno de programación.	7.00	0.00	6.10	0.88	6.31	0.85
El API de TinySOA es suficiente para el desarrollo de aplicaciones de monitoreo/visualización.	7.00	0.00	6.20	0.63	6.38	0.65
TinySOA me ofrece una abstracción familiar de acuerdo a mis conocimientos.	7.00	0.00	6.50	0.85	6.62	0.77
TinySOA me oculta los detalles de bajo nivel para el desarrollo de aplicaciones WSN.	6.67	0.58	6.40	1.07	6.46	0.97
Promedio:	**6.93**	0.12	**6.30**	0.69	**6.45**	0.78

Tabla II: **Resultados para el grupo de preguntas sobre abstracción**

La **integración** ofrecida por TinySOA:	Expertos		Noveles		Ambos	
	Media	Desv. Est.	Media	Desv. Est.	Media	Desv. Est.
Considero posible y sencilla la integración de TinySOA con otras aplicaciones.	6.33	0.58	6.20	0.63	6.23	0.44
TinySOA me oculta los detalles de bajo nivel para el desarrollo de aplicaciones WSN.	6.33	0.58	6.20	0.63	6.23	0.60
Promedio:	**6.33**	0.58	**6.20**	0.63	**6.23**	0.52

Tabla III: **Resultados para el grupo de preguntas sobre integración**

La **extensibilidad** ofrecida por TinySOA:	Expertos		Noveles		Ambos	
	Media	Desv. Est.	Media	Desv. Est.	Media	Desv. Est.
TinySOA me permite extender la funcionalidad del sistema para poder adaptarlo a nuevas necesidades.	6.00	1.00	5.60	0.84	5.69	0.85
Promedio:	**6.00**	1.00	**5.60**	0.84	**5.69**	0.85

Tabla IV: **Resultados para el grupo de preguntas sobre extensibilidad**

La **reconfigurabilidad** ofrecida por TinySOA:	Expertos		Noveles		Ambos	
	Media	Desv. Est.	Media	Desv. Est.	Media	Desv. Est.
Puedo reconfigurar (de manera sencilla) a TinySOA para utilizarlo en otros escenarios de aplicación.	6.00	1.00	5.90	0.99	5.92	0.95
Considero que TinySOA provee mecanismos apropiados para su reconfiguración.	6.00	0.00	5.70	0.82	5.77	0.73
Promedio:	**6.00**	0.50	**5.80**	0.91	**5.85**	0.84

Tabla V: **Resultados para el grupo de preguntas sobre reconfigurabilidad**

Grupo/Hipótesis	Expertos		Noveles		Ambos	
	Media	Desv. Est.	Media	Desv. Est.	Media	Desv. Est.
Abstracción	6.93	0.12	6.30	0.69	6.45	0.78
Integración	6.33	0.58	6.20	0.63	6.23	0.52
Extensibilidad	6.00	1.00	5.60	0.84	5.69	0.85
Reconfigurabilidad	6.00	0.50	5.80	0.91	5.85	0.84

Tabla VI: **Resumen de los resultados por grupo**

Factor	Expertos		Noveles		Ambos	
	Media	Desv. Est.	Media	Desv. Est.	Media	Desv. Est.
Facilidad de Uso	6.42	0.83	6.28	0.93	6.31	0.89
Utilidad	6.56	0.38	6.33	0.69	6.38	0.56
Adopción	6.83	0.29	6.15	0.90	6.31	0.85

Tabla VII: **Resultados del Modelo de Aceptación de la Tecnología**

Como se puede ver los resultados de los grupos, cada uno fue calificado favorablemente con valores muy cercanos a 6 (en una escala de 1 a 7) correspondiendo a «ciertamente de acuerdo». Lo que dice que los desarrolladores están generalmente de acuerdo en que el sistema concuerda con las hipótesis presentadas.

Un resultado similar puede ser observado en el resultado del grupo que agrupa las preguntas del modelo TAM. El cual puede ser observado en la Tabla VII.

De los resultados del grupo correspondiente al modelo TAM se puede concluir que la percepción de los desarrolladores es que no les sería difícil aprender a usar el sistema, además de que coinciden en que les podría ayudar en la construcción de aplicaciones de monitoreo y/o visualización. Además por los resultados se puede concluir que existe la disposición para la adopción la tecnología.

En el cuestionario existía la posibilidad de que los desarrolladores dejaran sus sugerencias sobre cómo mejorar el sistema, así como también comentarios generales. Entre las sugerencias recibidas se encuentra el que se desarrolle un paquete de distribución el cual pueda ser incorporado en varios IDEs (entornos de programación) de diferentes lenguajes (como NetBeans[3], Eclipse[4], JBuilder[5], entre otros) que incluya documentación y ejemplos, esto para una mayor facilidad en su uso y adopción.

En cuanto a los comentarios generales, los desarrollares hacen notar el tiempo que les tomó programar una aplicación haciendo uso de una red de sensores sin estar familiarizados con el área. Esto se evidencia al analizar el tiempo que les tomó en promedio (25 minutos) y el número de líneas de código que les llevó hacer la tarea (cerca de 15 líneas en promedio para la conexión, solicitud y procesamiento de la información recibida del sistema). Tradicionalmente, los desarrolladores noveles hubieran tenido que familiarizarse con el paradigma de componentes, preparar el *software* de los nodos y crear una conexión a la red para poder hacer la tarea, claramente tomándoles un tiempo mucho mayor, incluso los desarrolladores expertos hubieran tardado mucho más que 25 minutos. Mencionan también, que la documentación fue lo suficientemente buena como para llevarlos a utilizar el sistema sin contratiempos.

Concluyendo, los desarrolladores mostraron buena disposición a adoptar esta tecnología. Esto al comprobar que con el sistema TinySOA pueden programar aplicaciones que utilicen las características de sensado que le proveen las WSN sin necesidad de preocuparse por sus problemáticas de bajo nivel ni por problemas de conectividad a las mismas.

---

[3] Sun Microsystems. `http://java.sun.com/`
[4] The Eclipse Foundation. `http://www.eclipse.org/`
[5] Borland Software Corporation. `http://www.borland.com/`

VII.2. Evaluación de la arquitectura

En esta sección se presenta la evaluación funcional de la arquitectura propuesta. Este tipo de pruebas tienen como objetivo el encontrar las diferencias entre los requerimientos funcionales y la solución que se propone. Como se menciona en el Capítulo V, se tomaron dos tipos de requerimientos en el diseño de la arquitectura, los requerimientos que todo sistema *middleware* para WSN debe cumplir y los requerimientos de los desarrolladores de aplicaciones WSN que los utilizan. En el resto de la sección se describen brevemente cada uno de estos requerimientos así como también la mecánica considerada en el diseño de la arquitectura que llevó a satisfacerlos.

VII.2.1. Prueba de los requerimientos para sistemas *middleware*

Los requerimientos para sistemas *middleware* para WSN que se utilizaron en el diseño de la arquitectura propuesta son (Römer, 2004, Marrón, 2005):

1. *Abstracción.* El sistema *middleware* debe permitir la especificación de las tareas a un alto nivel. También debe esconder en lo más posible los detalles de *hardware* y de distribución al desarrollador.

2. *Eficiencia.* Debe tener especial cuidado en el consumo de energía.

3. *Programabilidad.* Debe ser flexible y para ello, permitir la configuración o reconfiguración de sus características y funcionamiento.

4. *Adaptabilidad.* El sistema debe ser capaz de adaptarse a los cambios de la red para continuar con su correcto funcionamiento.

5. *Escalabilidad.* Debe permitir el crecimiento en el número de nodos de la red y estar preparado para ello.

6. *Topología.* Debe proveer mecanismos para apoyar la topología siempre cambiante de la red y garantizar robustez en la misma.

7. *Integración*. Debe permitir la integración con infraestructuras externas o incluso con otras redes.

Prueba del requerimiento 1

El requerimiento de abstracción se refiere a que el sistema *middleware* debe permitir la especificación de tareas a un alto nivel, escondiendo en lo más posible los detalles de bajo nivel, tales como especificaciones de topología, enrutamiento, etc.

La arquitectura está diseñada para proveer dos niveles de abstracción. El primero es en el área de captura, en la que se utiliza el patrón *Publish/Subscribe*. Los nodos de sensado descubren y publican sus servicios en el nodo *sink*, y éste a su vez se suscribe a los servicios que conoce que ofrece la red. Esta comunicación se da a través de mensajes simples y descriptivos, por ejemplo «suscribir al servicio luminosidad».

El segundo se da en el área de concentración. El proveedor de servicios externos es el encargado de ello, al hacer posible el uso de un servicio web para la comunicación con la red. De esta manera el envío de comandos y solicitudes a la red se realiza a través de simples llamadas a métodos de un objeto creado con el servicio web de red. Estos métodos comprenden la funcionalidad suficiente como para permitir el desarrollo de aplicaciones de monitoreo o visualización.

Prueba del requerimiento 2

El segundo requerimiento se refiere a la necesidad de eficiencia. Es decir, tener especial cuidado en cuanto a la energía y a los recursos mínimos con los que cuenta la red y en particular un nodo de sensado.

La arquitectura propuesta tiene cuidado con el consumo de energía de dos maneras.

La primera es que solamente una pequeña parte de la infraestructura del *middleware* reside en los nodos de sensado. De esta manera sólo se consume energía para las actividades esenciales del *middleware*.

La segunda es que al mantener encapsulada la funcionalidad del *software* de los nodos de sensado en forma de componentes, es posible utilizar soluciones altamente eficaces y comprobadas tales como algoritmos de enrutamiento, controladores de topología, etc. Disponibles actualmente, gracias al fruto de una activa investigación que se ha realizado en el área en cuanto a las capas de bajo nivel.

Prueba del requerimiento 3

El tercer requerimiento tiene que ver con la reconfigurabilidad del sistema *middleware*. Es decir, que su funcionalidad pueda ser cambiada o modificada para satisfacer nuevas necesidades de aplicación. Por ejemplo, a una aplicación de monitoreo de cultivos le bastaría con recibir información de la red cada cierto número de minutos, pero si se tuviera otra aplicación en la que se monitorea los eventos ocurridos a un hábitat ésta necesitaría de una manera para permitir que se siga utilizando el mismo sistema, simplemente modificando alguna propiedad, en este caso la tasa de muestreo.

Esto es parte integral de la arquitectura, con el envío de tareas de mantenimiento a la red de sensores. En estas tareas se ofrece funcionalidad para cambiar el modo en el que el sistema *middleware* se comporta. Tales como la tasa de muestreo, entre otros.

Prueba del requerimiento 4

El cuarto requerimiento se refiere a la adaptabilidad. Debido a la naturaleza misma de las redes de sensores, un sistema *middleware* para WSN necesita estar consciente de los cambios altamente dinámicos que suceden en las mismas. Estos cambios incluyen la

pérdida de algún nodo (por baja de energía o falla mecánica), la pérdida de calidad de comunicación, entre otras. Un sistema *middleware* para WSN debe ser capaz de mantener su funcionalidad a pesar de estos inconvenientes.

Este requerimiento es satisfecho de dos maneras. La primera es por el componente de sistema operativo del nodo de sensado. Al construir los demás componentes a partir de él, se obtiene la funcionalidad de adaptabilidad a errores y a otros inconvenientes, que ofrece un sistema operativo embebido para WSN. Sistemas operativos que se encuentran disponibles actualmente. Un ejemplo destacado de ellos es TinyOS, estándar de facto en desarrollos de investigación.

La segunda es gracias al empleo del modelo *data-centric* en el área de captura, de esta manera la red conoce qué servicios ofrece la red como un todo, sin importarle qué nodo individual es el responsable. La infraestructura hace referencia a los datos que son proveídos por la red no a quién los provee. Con lo que por ejemplo, si existen dos nodos con capacidad de sensado de temperatura y uno de ellos llegara a fallar, el nodo restante tomaría la responsabilidad del nodo con la falla.

Prueba del requerimiento 5

El quinto requerimiento trata sobre la escalabilidad. Este requerimiento establece que un sistema *middleware* debe estar preparado para poder soportar un número incremental de sensores. Sobre todo a futuro, donde se espera que existan redes con cientos o incluso miles de nodos de sensado.

La arquitectura tiene contemplado este requerimiento de dos maneras. La primera es por el uso del modelo *data-centric*. La segunda es que permite que el componente de comunicación sea fácilmente intercambiado en caso de que no ofrezca soporte al número de nodos que se desea utilizar.

Prueba del requerimiento 6

El sexto requerimiento establece que el sistema *middleware* debe mantener una topología dinámica, y estar consciente de ello.

La arquitectura contempla este requerimiento mediante el uso del componente de sistema operativo embebido. Con el que se utilizan soluciones utilizadas y disponibles actualmente con las que se tiene un buen desempeño en el control de topología.

Prueba del requerimiento 7

El séptimo y último requerimiento de sistemas *middleware* establece que se debe permitir la integración con infraestructuras externas o incluso con otras redes.

Esto está contemplando en la arquitectura en el área de concentración con el proveedor de servicios externos. Los servicios web permiten la fácil integración con otros sistemas y pueden ser utilizados para combinar la información de varias redes de sensores.

VII.2.2. Prueba de los requerimientos de los desarrolladores

Uno de los objetivos principales de la arquitectura es facilitar la creación de nuevas aplicaciones WSN para monitoreo y/o visualización. Para ello, se buscó apoyar las necesidades de los desarrolladores de estas aplicaciones, los cuales han sido identificados por Buonadonna et al. (2005). Estos requerimientos son:

1. *Reconfigurabilidad*. Para que las aplicaciones puedan cambiar de tasa de datos, tipos de datos recolectados, etc.

2. *Monitoreo de vitalidad*. Para conocer la vitalidad de la red. Por ejemplo mediante el conocimiento del número de nodos con energía.

3. *Lecturas de sensado interpretables*. Las lecturas deben ser reportadas en unidades de ingeniería. Por ejemplo en grados centígrados.

4. *Integración*. No solamente se deben dar medios para permitir la integración con otras herramientas, esta también debe ser sencilla.

5. *API familiar*. Se debe proveer un API que sea de fácil uso para el desarrollador, idealmente nativo al lenguaje de programación que este utiliza habitualmente.

6. *Extensibilidad en el software del nodo*. Para por ejemplo poder agregar nuevos tipos de sensores o nuevos operadores de datos.

7. *Modularidad en los servicios de red*. Para permitir la experimentación con diferentes soluciones para enrutamiento, calendarización, etc.

Durante el diseño de la arquitectura, se fue implementando un sistema *middleware* prototipo que tuviera sus características. Debido a la naturaleza cualitativa de los requerimientos para los desarrolladores, en la evaluación funcional de los mismos se hace uso de algunos de los resultados encontrados en la evaluación de TinySOA presentada al inicio de este capítulo.

Prueba del requerimiento 1

El primer requerimiento es el de reconfigurabilidad. Este requerimiento se refiere a la necesidad de cambiar o modificar el comportamiento de la arquitectura, para permitir diferentes escenarios de aplicación. Además esto debe ser fácil para los desarrolladores de esas aplicaciones.

Este requerimiento está contemplado con la introducción de las tareas de mantenimiento. Con las tareas es posible cambiar la forma en la que la red se comporta, llevando a reconfigurarla a la manera que sea de utilidad para la aplicación que se le desee dar. Las tareas de mantenimiento se controlan por medio de métodos del servicio web de red.

En la evaluación de TinySOA, el sistema prototipo *middleware* creado a partir de la arquitectura, se encontró que los desarrolladores consideran fácil el manejo de la reconfiguración con esta tecnología. En la Tabla V de la Sección VII.1.3 pueden observarse los resultados de la evaluación de este aspecto, donde los desarrolladores opinaron que están «ciertamente de acuerdo» (6 en una escala de 7) en la facilidad de reconfiguración.

Prueba del requerimiento 2

El segundo requerimiento trata de la necesidad que tienen los desarrolladores de conocer cuál es la vitalidad de la red, en términos de energía.

La arquitectura provee conocimiento de la vitalidad de la red. Uno de los servicios de control de los nodos de sensado es precisamente la energía restante en batería, con este servicio los desarrolladores pueden tener una idea de la vitalidad de la red al poder hacer cálculos con la información recibida.

Prueba del requerimiento 3

El tercer requerimiento especifica que las lecturas entregadas por la red deben estar codificadas en unidades de ingeniería.

Este requerimiento es satisfecho en la capa de concentración, en donde al arribo de las lecturas, son interpretadas de acuerdo al *hardware* que las capturó. Esto debido a que los diferentes fabricantes de un mismo dispositivo transductor devuelven lecturas en unidades crudas diferentes, al conocer qué *hardware* las capturó se sabe qué método utilizar para interpretarlas. De esto es responsable principalmente el subsistema Gateway, el cual conoce con qué tipo de *hardware* cuentan los nodos gracias a los mensajes de registro que fueron enviados durante la inicialización de los nodos y gracias a que en el mensaje de comunicación de lecturas se adjunta qué dispositivo transductor la capturó.

Prueba del requerimiento 4

El cuarto requerimiento establece que los mecanismos de integración que deben proveer los sistema *middleware* para WSN tienen que ser fácil de usar.

La arquitectura tiene contemplada la integración al proveer servicios web. Para comprobar la facilidad de uso, este aspecto fue evaluado en el sistema TinySOA. Los resultados pueden observarse en la Tabla III de la Sección VII.1.3, en donde los desarrolladores opinaron que están «ciertamente de acuerdo» en que la integración de las redes en aplicaciones ya existentes o en la integración con otras redes es fácil y sencilla.

Prueba del requerimiento 5

El quinto requerimiento establece que el API con el que se hace uso de la infraestructura *middleware* WSN debe ser fácil de usar y familiar al desarrollador.

La arquitectura tiene contemplado esto con los servicios web, los cuales, después de enlazados e inicializados, pueden ser utilizados de manera nativa al lenguaje de programación del desarrollador. Este aspecto fue uno de los principales objetivos del experimento de evaluación de TinySOA. Los resultados en este aspecto pueden apreciarse en la Tabla II de la Sección VII.1.3 en donde los desarrolladores manifestaron estar «ciertamente de acuerdo» en que el API puede usarse de manera nativa al entorno de programación de preferencia y que además es fácil de usar. Además manifestaron que la abstracción proveída por el mismo es suficiente para el desarrollo de aplicaciones de monitoreo y/o visualización y que les oculta exitosamente los detalles de bajo nivel.

Prueba del requerimiento 6

El sexto requerimiento de los desarrolladores es que la funcionalidad de un sistema *middleware* debe ser sencilla de extender. Esto para por ejemplo, agregar nuevos tipos de operaciones o capacidades de sensado.

La arquitectura da soporte a la extensibilidad gracias a la conformación de todos los sistemas en componentes. Estos componentes pueden ser fácilmente mejorados o intercambiados por otros que logren soportar nuevas necesidades originalmente no contempladas. En la evaluación de TinySOA se les preguntó a los desarrolladores si consideraban sencilla la extensión del sistema, en la Tabla IV de la Sección VII.1.3 se muestran los resultados, en donde los desarrolladores consideraron estar «ciertamente de acuerdo».

Prueba del requerimiento 7

El último requerimiento establece que los sistemas *middleware* deben estar diseñados de manera modular, para facilitar el intercambio de funcionalidad específica.

Este requerimiento es uno de los principalmente contemplados en la arquitectura, en donde todos los subsistemas están compuestos por componentes. En cada uno de los componentes se encapsula cierta funcionalidad específica, como por ejemplo Comunicación.

VII.2.3. Resumen

En la fase de análisis de la arquitectura, se encontraron los requerimientos de los sistemas *middleware* WSN así como los de los desarrolladores. Como se puede apreciar en este capítulo, el diseño de la arquitectura propuesta por este trabajo de tesis, tiene contemplado cada uno de estos distintos requerimientos. Además, gracias a la evaluación conducida en el sistema prototipo *middleware* construido a partir de ella, fue posible evaluar algunos aspectos cualitativos, tales como la facilidad hacia los desarrolladores.

Capítulo VIII

Conclusiones

"Un comienzo no desaparece nunca, ni siquiera con un final".
– Harry Mulisch (1927-)

Las redes inalámbricas de sensores ofrecen una nueva gama de aplicaciones. Es importante continuar en el desarrollo de mecanismos, protocolos y dispositivos ahora que el área está aún en etapa de desarrollo. Los avances que hasta el momento se han logrado, hacen posible la construcción de prototipos que nos demuestran sus beneficios. Gracias a la continua evolución y disminución de costos de la tecnología, no es de extrañarse que en un futuro no muy lejano se pueda contar con sensores de un tamaño microscópico, de costos tan bajos que la pérdida de varios de ellos no represente inconvenientes. Incluso podamos pensar en aplicaciones que hagan uso de varios cientos o miles de ellos. Las redes inalámbricas de sensores es una de las principales tecnologías que nos permitirán hacer del cómputo ubicuo una realidad, con los consecuentes beneficios a la vida diaria de las personas que ello implica. Este trabajo es un paso más en la integración de las redes de sensores con Internet y otras aplicaciones, así como también uno más en cerrar la brecha entre las necesidades de aplicación y la lógica de bajo nivel de la red.

VIII.1. Aportaciones

La aportación principal de este trabajo es la exploración realizada en la aplicación de la orientación a servicios en el diseño de una arquitectura que permita la construcción de sistemas *middleware* para WSN. Tal como menciona Mukhi et al. (2004), las arquitecturas

orientadas a servicios son una de las tecnologías que prometen ofrecer las bases para la siguiente generación de sistemas operativos. Aunque ya se había prestado algo de atención en cuanto a la aplicación de este tipo de arquitecturas en las WSN, esto no había sido abordado en concreto. La arquitectura propuesta está cuidadosamente diseñada para las plataformas WSN disponibles actualmente, haciendo de su adopción una realidad y permitiendo su uso en la solución de un problema.

Otra aportación, es el diseño e implementación de un sistema *middleware* prototipo para WSN basado en la arquitectura propuesta. Este prototipo totalmente funcional puede ser utilizado en nuevas comprobaciones y en futuras extensiones a esta investigación.

VIII.2. Trabajo Futuro

Aún existe mucho trabajo por delante, en cuanto al diseño de la arquitectura propuesta y en cuanto al diseño e implementación del sistema prototipo TinySOA. Algunas de las oportunidades de trabajo a futuro identificadas son las siguientes:

- Analizar e integrar los requerimientos no funcionales (de calidad de servicio) en el diseño de la arquitectura.
- Explorar diferentes estrategias de agregación en la arquitectura, con la finalidad de permitir su funcionamiento en ambientes masivos. Tales estrategias podrían ser la conformación dinámica de grupos, la composición de servicios, etc.
- Diseñar un modelo que permita dirigir las transacciones por medio de la localización de los nodos de sensado, además contemplarla como servicio.
- Incluir estrategias de ahorro de energía en el sistema *middleware* prototipo.
- Diseñar una estrategia para la identificación automática de dispositivos de sensado en el nodo para el *middleware* prototipo.

Bibliografía

Abdelzaher, T., Blum, B., Cao, Q., Chen, Y., Evans, D., George, J., George, S., Gu, L., He, T., Krishnamurthy, S., Luo, L., Son, S., Stankovic, J., Stoleru, R. y Wood, A. 2004. EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks. *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04). Tokio, Japón. 24–26 de marzo de 2004*, 582–589 p.

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y. y Cayirci, E. 2002. A Survey on Sensor Networks. *IEEE Communications Magazine. Agosto de 2002*, 40(8):102–114 p.

Bernstein, P. A. 1996. Middleware: A Model for Distributed Services. *Communications of the ACM. Febrero de 1996*, 39(2):86–97 p.

Bonnet, P., Gehrke, J. E. y Seshadri, P. 2000. Querying the Physical World. *IEEE Personal Communications. Octubre de 2000*, 7(5):10–15 p.

Buonadonna, P., Gay, D., Hellerstein, J., Hong, W. y Madden, S. 2005. TASK: Sensor Network in a Box. *European Workshop on Wireless Sensor Neworks 2005. Estambul, Turquía. 31 de enero – 2 de febrero de 2005*, 133–144 p.

Chappell, D. A. y Jewell, T. 2002. *Java Web Services*. O'Reilly Media, Inc. Primera edición. ISBN 0-596-00269-6. 276 pp.

Crossbow Technology, Inc. 2005. Especificación de las placas de sensado MTS/MDA. URL: `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MTS_MDA_Datasheet.pdf` (consultado en diciembre de 2005).

Crossbow Technology, Inc. 2006a. Especificación de la placa programadora MIB510CA. URL: `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB510CA_Datasheet.pdf` (consultado en agosto de 2006).

Crossbow Technology, Inc. 2006b. Especificación del mote MICAz. URL: `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf` (consultado en agosto de 2006).

Culler, D. E. y Hong, W. 2004. Wireless Sensor Networks. *Communications of the ACM*, 47(6):30–33 p.

Curino, C., Giani, M., Giorgetta, M., Giusti, A., Murphy, A. L. y Picco, G. P. 2005. TinyLime: Bridging Mobile and Sensor Networks through Middleware. *In Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom'2005). Isla Kauai, Hawaii, EE.UU. 8–12 de marzo de 2005*, 61–72 p.

Davis, F. D. y Venkatesh, V. 1996. A Critical Assessment of Potential Measurement Biases in the Technology Acceptance Model: Three Experiments. *International Journal of Human Computer Studies. Julio de 1996*, 45(1):19–45 p.

Delicato, F. C., Pires, P. F., Pirmez, L. y da Costa Carmo, L. F. R. 2003a. A Flexible Middleware System for Wireless Sensor Networks. *ACM/IFIP/USENIX International Middleware Conference. Río de Janeiro, Brasil. 16–20 de junio de 2003*, 2672:474–492 p.

Delicato, F. C., Pires, P. F., Pirmez, L. y da Costa Carmo, L. F. R. 2003b. A Flexible Web Service based Architecture for Wireless Sensor Networks. *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference. Rhode Island, EE.UU. 12–22 de mayo de 2003*, 730–735 p.

Delicato, F. C., Pires, P. F., Rust, L., Pirmez, L. y de Rezende, J. F. 2005. Reflective Middleware for Wireless Sensor Networks. *20th Annual ACM Symposium on Applied Computing (ACM SAC'2005). Santa Fe, Nuevo México, EE.UU. 13–17 de marzo de 2005*, 1155–1159 p.

Delicato, F. C., Pirmez, L., Pires, P. F. y de Rezende, J. F. 2006. Exploiting Web Technologies to Build Automatic Wireless Sensor Networks. *8th IFIP IEEE International Conference on Mobile and Wireless Communication Networks (MWCN'2006). Santiago, Chile. 20–25 de agosto de 2006*, 99–114 p.

Estrin, D., Culler, D., Pister, K. y Sukhatme, G. 2002. Connecting the Physical World with Pervasive Networks. *Pervasive Computing, IEEE. Enero–Marzo, 2002*, 1(1):59–69 p.

Fok, C. L., Roman, G. C. y Lu, C. 2005. Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications. *In Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'05). Ohio, EE.UU. 6–10 de junio de 2005*, 653–662 p.

Golatowski, F., Blumenthal, J., Handy, M. y Haase, M. 2003. Service-Oriented Software Architecture for Sensor Networks. *International Workshop on Mobile Computing (IMC 2003). Rostock, Alemania. 17–18 de junio de 2003*, 93–98 p.

Graham, S., Davis, D., Simeonov, S., Daniels, G., Brittenham, P., Nakamura, Y., Fremantle, P., Koenig, D. y Zentner, C. 2004. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI*. Sams Publishing. Segunda edición. ISBN 0-672-32641-8. 816 pp.

Han, Q. y Venkatasubramanian, N. 2001. Autosec: An Integrated Middleware Framework for Dynamic Service Brokering. *IEEE Distributed Systems Online*, 2(7).

Hashimi, S. 2004. Service-Oriented Architecture Explained. URL: `http://dev2dev.bea.com/pub/a/2004/05/soa_hashimi.html` (consultado en diciembre 2005).

He, H. 2003. What is Service-Oriented Architecture? URL: `http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html` (consultado en diciembre de 2005).

Heinzelman, W., Murphy, A., Carvalho, H. y Perillo, M. 2004. Middleware to Support Sensor Network Applications. *IEEE Network Magazine. Enero de 2004*, 18(1):6–14 p.

Hill, J., Horton, M., Kling, R. y Krishnamurthy, L. 2004. The Platforms Enabling Wireless Sensor Networks. *Communications of the ACM*, 47(6):41–46 p.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. y Pister, K. 2000. System Architecture Directions for Networked Sensors. *9th International Conference on Architectural Support for Programming Languages and Operating Systems. Massachusetts, EE.UU. 12–15 de noviembre de 2000*, 93–104 p.

Hines, J. R. 1996. Software Engineering. *IEEE Spectrum. Enero de 1996*, 33(1):60–64 p.

Hollar, S. E. A. 2000. *COTS Dust*. Master's thesis, University of California, Berkeley. URL: `http://www-bsac.eecs.berkeley.edu/archive/users/hollar-seth/publications/cotsdust.pdf` (consultado en julio de 2006).

Intanagonwiwat, C., Govindan, R. y Estrin, D. 2000. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. *Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom'00). Massachusetts, EE.UU. 6–11 de agosto de 2000*, 56–67 p.

Jaikaeo, C., Srisathapornphat, C. y Shen, C. C. 2000. Querying and Tasking in Sensor Networks. *SPIE's 14th Annual International Symposium Aerospace/Defense Sensing, Simulation, and Control (Digitization of the Battlespace V). Florida, EE.UU. 24–28 de abril de 2000*, 4037:184–197 p.

Kang, P., Borcea, C., Xu, G., Saxena, A., Kremer, U. y Iftode, L. 2004. Smart Messages: A Distributed Computing Platform for Networks of Embedded Systems. *The Computer Journal*, 47(4):475–494 p.

Levis, P. y Culler, D. 2002. Maté: A Tiny Virtual Machine for Sensor Networks. *10th International Conference on Architectural Support for Programming Languages and Operating Systems. California, EE.UU. 5–9 de octubre de 2002*, 85–95 p.

Li, S., Lin, Y., Son, S., Stankovic, J. y Wei, Y. 2003. Event Detection Services Using Data Service Middleware in Distributed Sensor Networks. *Proceedings 2nd International Workshop Information Processing in Sensor Networks. California, EE.UU. 22–23 de abril de 2003*, 2634:502–517 p.

Likert, R. 1932. A Technique for the Measurement of Attitudes. *Archives of Psychology*, (140), 1–55 p.

Liu, T. y Martonosi, M. 2003. Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems. *Proceedings of the 9th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'03). California, EE.UU. 11–13 de junio de 2003*, 107–118 p.

Madden, S., M. J. Franklin, J. M. Hellerstein y Hong, W. 2002. TAG: A Tiny AGregation Service for Ad-Hoc Sensor Networks. *5th Symposium on Operating Systems Design and Implementation (OSDI'2002). Massachusetts, EE.UU. 9–11 de diciembre de 2002*.

Marrón, P. J. 2005. Middleware Approaches for Sensor Networks. URL: `http://www.vs.inf.ethz.ch/events/dag2005/program/lectures/marron-2.pdf` (consultado en enero de 2005).

Marrón, P. J., Minder, D., Lachenmann, A. y Rothermel, K. 2005. TinyCubus: An Adaptive Cross-Layer Framework for Sensor Networks. *Information Technology (IT)*, 47(2):87–97 p.

Mukhi, N. K., Konuru, R. y Curbera, F. 2004. Cooperative Middleware Specialization for Service Oriented Architectures. *Proceedings of the 13th International Conference on the World Wide Web (WWW'2004). Nueva York, EE.UU. 17–22 de mayo de 2004*, 206–214 p.

Pallos, M. S. 2001. Service-Oriented Architecture: A Primer. *eAI Journal. Diciembre de 2001*, 32–35 p.

Perrig, A., Stankovic, J. y Wagner, D. 2004. Security in Wireless Sensor Networks. *Communications of the ACM*, 47(6):53–57 p.

Reichardt, D., Miglietta, M., Moretti, L., Morsink, P. y Schulz, W. 2002. CarTALK 2000: Safe and comfortable driving based upon inter-vehicle communication. *Proceddings of the Intelligent Vehicle Symp.*, 2:545–550 p.

Römer, K. 2004. Programming Paradigms and Middleware for Sensor Networks. *GI/ITG Workshop on Sensor Networks. Alemania. Febrero de 2004*, 49–54 p.

Römer, K., Kasten, O. y Mattern, F. 2002. Middleware Challenges for Wireless Sensor Networks. *ACM Mobile Computing and Communication Review (MC2R). Octubre de 2002*, 6(4):59–61 p.

Römer, K. y Mattern, F. 2004. The Design Space of Wireless Sensor Networks. *IEEE Wireless Communications. Diciembre de 2004*, 11(6):54–64 p.

Schmuller, J. 2001. *Aprendiendo UML en 24 Horas.* Prentice Hall. Primera edición. ISBN 968-444-463-X. 448 pp.

Szewczyk, R., Osterweil, E., Polastre, J., Hamilton, M., Mainwaring, A. y Estrin, D. 2004. Habitat Monitoring with Sensor Networks. *Communications of the ACM*, 47(6):34–40 p.

Tanenbaum, A. S. 2001. *Modern Operating Systems.* Prentice Hall. 2da. edición. ISBN 0-13-031358-0. 951 pp.

Whitehouse, K., Sharp, C., Brewer, E. y Culler, D. 2004. Hood: A Neighborhood Abstraction for Sensor Networks. *Proceedings of the 2nd International Conference on Mobile Systems, Applications and Services (MobiSys'04). Massachusetts, EE.UU. 6–9 de junio de 2004*, 99–110 p.

Zhao, F. y Guibas, L. 2004. *Wireless Sensor Networks: An Information Processing Approach.* Morgan Kauffmann. ISBN 1-55860-914-8.

Apéndice A

Acerca de TinyOS

TinyOS es un sistema operativo basado en eventos diseñado para su uso con redes de sensores embebidas. Más específicamente, está diseñado para soportar operaciones concurrentes requeridas por las redes de sensores con requerimientos de *hardware* mínimos.

El sistema TinyOS, las librerías y las aplicaciones están escritas en nesC, un nuevo lenguaje para programar aplicaciones estructuradas basadas en componentes. El lenguaje nesC está diseñado para sistemas embebidos tales como las redes de sensores. nesC tiene una sintaxis parecida al lenguaje de programación C, pero soporta el modelo de concurrencia de TinyOS, así como también mecanismos para estructurar, nombrar y enlazar componentes de *software* en sistemas embebidos de red robustos. La meta principal es permitir a los diseñadores construir componentes que puedan ser fácilmente compuestos en sistemas concurrentes completos.

TinyOS define un varios conceptos importantes que están expresados en nesC. Primero, las aplicaciones nesC están construidas de componentes con interfaces bien definidas y bidireccionales. Segundo, nesC define un modelo de concurrencia, basado en tareas y en manejadores de eventos de *hardware*.

A.1. Componentes

A.1.1. Especificación

Una aplicación nesC consiste de uno o más componentes enlazados entre sí para formar un ejecutable. Un componente provee y usa interfaces. Estas interfaces son el único

punto de acceso al componente y son bidireccionales. Una interfaz declara un conjunto de funciones llamadas comandos que el proveedor de la interfaz debe implementar y otro conjunto de funciones llamadas eventos que el usuario de la interfaz debe implementar. Para que un componente pueda llamar a los comandos en una interfaz, debe implementar los eventos de dicha interfaz. Un componente puede usar o proveer múltiples interfaces y múltiples instancias de la misma interfaz.

A.1.2. Implementación

Hay dos tipos de componentes en nesC: módulos y configuraciones. Los módulos proveen código de aplicación, implementando una o más interfaces. Las configuraciones son utilizadas para ensamblar otros componentes, conectando interfaces utilizadas por componentes a interfaces proveídas por otros. Cada aplicación nesC es descrita por una configuración de alto nivel que enlaza los componentes internos.

A.2. Modelo de concurrencia

TinyOS ejecuta solamente un programa que consiste de los componentes de sistema seleccionados y de los componentes personalizados necesarios para una aplicación. Hay dos hilos de ejecución: tareas y eventos de *hardware*. Las tareas son funciones cuya ejecución es deferida. Una vez inicializadas, se ejecutan hasta terminar y no se interfieren unas con otras. Los eventos de *hardware* son ejecutados en respuesta a interrupciones de *hardware* y también se ejecutan hasta terminar, pero pueden interferir con la ejecución de una tarea o la de otro evento de *hardware*.

Información obtenida de: `http://www.tinyos.net/`

Para ejemplos de aplicación y otras demostraciones:
`http://www.tinyos.net/tinyos-1.x/doc/tutorial/`

Apéndice B

Constantes del Sistema TinySOA

Nombre	Valor	Descripción
Placas de sensado		
SBID_MDA500	0x01 (1)	Placa Crossbow MDA500
SBID_MTS510	0x02 (2)	Placa Crossbow MTS510
SBID_MEP500	0x03 (3)	Placa Crossbow MEP500
SBID_MDA400	0x80 (128)	Placa Crossbow MDA400
SBID_MDA300	0x81 (129)	Placa Crossbow MDA300
SBID_MTS101	0x82 (130)	Placa Crossbow MTS101
SBID_MTS300	0x83 (131)	Placa Crossbow MTS300
SBID_MTS310	0x84 (132)	Placa Crossbow MTS310
SBID_MTS400	0x85 (133)	Placa Crossbow MTS400
SBID_MTS420	0x86 (134)	Placa Crossbow MTS420
SBID_MEP401	0x87 (135)	Placa Crossbow MEP401
SBID_MDA320	0x90 (144)	Placa Crossbow MDA320
SBID_MSP410	0xA0 (160)	Placa Crossbow MSP410
SBID_TMSKY1	0xE1 (225)	Sensores Integrados (TMote Sky)
Tipos de mensajes		
TINYSOA_TIPO_LECTURA	0x01 (1)	Mensaje de lectura
TINYSOA_TIPO_REGISTRO	0x02 (2)	Mensaje de registro
TINYSOA_TIPO_ACTIVA_ACTUADOR	0x03 (3)	Mensaje de activación de un actuador
TINYSOA_TIPO_DESACTIVA_ACTUADOR	0x04 (4)	Mensaje de desactivación de un actuador
TINYSOA_TIPO_DUERME	0x05 (5)	Mensaje de entrada a estado de espera
TINYSOA_TIPO_DESPIERTA	0x06 (6)	Mensaje de salida de estado de espera
TINYSOA_TIPO_CAMBIA_DATA_RATE	0x07 (7)	Mensaje de especificación de cambio de tasa de datos
TINYSOA_TIPO_SOLICITUD_REGISTRO	0x08 (8)	Mensaje de solicitud de registro
TINYSOA_TIPO_SUSCRIBIR	0x09 (9)	Mensaje de suscripción
Parámetros de sensado		
TINYSOA_SENSOR_TEMP	0xB001 (45057)	Sensor de temperatura
TINYSOA_SENSOR_LUZ	0xB002 (45058)	Sensor de luminosidad
TINYSOA_SENSOR_MAGX	0xB003 (45059)	Sensor de magnetismo, eje X
TINYSOA_SENSOR_MAGY	0xB004 (45060)	Sensor de magnetismo, eje Y
TINYSOA_SENSOR_ACEX	0xB005 (45061)	Sensor de aceleración, eje X
TINYSOA_SENSOR_ACEY	0xB006 (45062)	Sensor de aceleración, eje Y
TINYSOA_SENSOR_MIC	0xB007 (45063)	Micrófono
TINYSOA_SENSOR_VOLT	0xB008 (45064)	Voltaje
Actuadores		
TINYSOA_ACTUADOR_BOCINA	0xA001 (40961)	Bocina
TINYSOA_ACTUADOR_LED_ROJO	0xA002 (40962)	*Led* rojo
TINYSOA_ACTUADOR_LED_AMARILLO	0xA003 (40963)	*Led* amarillo
TINYSOA_ACTUADOR_LED_AZUL	0xA004 (40964)	*Led* azul
TINYSOA_ACTUADOR_LED_VERDE	0xA005 (40965)	*Led* verde

Apéndice C

Cuestionario de Evaluación de TinySOA

1. El API proveído por TinySOA es de fácil uso.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

2. Puedo utilizar el API de TinySOA de manera nativa a mi entorno de programación.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

3. El API de TinySOA es suficiente para el desarrollo de aplicaciones WSN de monitoreo/visualización.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

4. TinySOA me ofrece una abstracción familiar de acuerdo a mis conocimientos.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

5. TinySOA me oculta los detalles de bajo nivel para el desarrollo de aplicaciones WSN.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

6. Considero posible y sencilla la integración de TinySOA con otras aplicaciones.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

7. Considero posible utilizar TinySOA en la integración de la lectura de múltiples redes de sensores.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

8. TinySOA me permite extender la funcionalidad del sistema para poder adaptarlo a nuevas necesidades.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

9. Puedo reconfigurar (de manera sencilla) a TinySOA para utilizarlo en otros escenarios de aplicación.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

10. Considero que TinySOA provee mecanismos apropiados para su reconfiguración.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

11. Utilizar TinySOA me permite desarrollar aplicaciones WSN rápidamente.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

12. Utilizar TinySOA me permite desarrollar aplicaciones WSN fácilmente.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

13. Encuentro fácil el uso del sistema TinySOA.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

14. Me resultaría fácil integrar TinySOA en mis desarrollos de *software*.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

15. Mi interacción con TinySOA es sencilla y entendible.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

16. Considero que TinySOA puede ayudarme en el desarrollo de aplicaciones WSN.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

17. En general, encuentro útil el sistema *middleware* TinySOA.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

18. Considero que si tuviera acceso a TinySOA lo utilizaría.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

19. Considero que no tendría mayor problema en la adopción de TinySOA.

en desacuerdo	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	de acuerdo

Perfil personal

¿Cuál es su experiencia en el desarrollo de aplicaciones?

¿Con qué ambientes de programación es familiar? (ordene por grado de preferencia)

¿Qué tan familiarizado se encuentra con el paradigma de programación orientada a objetos?

no familiarizado	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	familiarizado

¿Cuál es su nivel en cuanto al desarrollo con servicios web en el ambiente de programación de su preferencia?

principiante	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	experto

¿Qué tan familiarizado está con el área de las redes inalámbricas de sensores (WSN)?

no familiarizado	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	familiarizado

¿Cuál es su nivel en el desarrollo de aplicaciones WSN?

principiante	[ ] completamente	[ ] ciertamente	[ ] ligeramente	[ ] neutralmente	[ ] ligeramente	[ ] ciertamente	[ ] completamente	experto

Comentarios y sugerencias en cuanto a las características del sistema:

Comentarios generales:

Nombre completo:	Grado de estudios:	Edad: