TESIS DEFENDIDA POR

César Olea Aguilar

Y aprobada por el siguiente comité:

Dr. José Antonio García Macías
Director del Comité

Dr. José Alberto Fernández Zepeda
Miembro del Comité

Dr. Jaime Sánchez García
Miembro del Comité

M.C. Raúl Tamayo Fernández
Miembro del Comité

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

Dr. David Hilario Covarrubias Rosales
Director de Estudios de Posgrado

17 de Septiembre de 2008

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE
EDUCACIÓN SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS
EN CIENCIAS DE LA COMPUTACIÓN

Arquitectura Modular para Redes Inalámbricas de Sensores y Actuadores

TESIS

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

MAESTRO EN CIENCIAS

Presenta:

César Olea Aguilar

Ensenada, Baja California, México. Septiembre de 2008.

RESUMEN de la tesis que presenta **César Olea Aguilar**, como requisito parcial para obtener el grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN. Ensenada, Baja California. Septiembre de 2008.

Arquitectura Modular para Redes Inalámbricas de Sensores y Actuadores

Resumen aprobado por:

Dr. José Antonio García Macías
Director de Tesis

En la actualidad, cada vez más dispositivos cuentan con la capacidad de conectarse a Internet. La Internet «de cosas» difiere del modelo anterior de Internet, ya que los dispositivos que la integran pueden monitorear y modificar el ambiente en el que se encuentran. Las redes inalámbricas de sensores y actuadores son una tecnología clave para el desarrollo de la Internet «de cosas» gracias a su capacidad de monitoreo y actuado. Sin embargo, la creación de aplicaciones para este tipo de redes requiere del conocimiento de cada uno de los dispositivos que la integran, dando como resultado la necesidad de programación de bajo nivel y aplicaciones dependientes de la plataforma de implementación. En esta tesis se propone una arquitectura modular para proporcionar soporte dinámico a los distintos dispositivos que integran la red, utilizando una interfaz de programación consistente entre dispositivos y de alto nivel.

Se realizó el diseño de la arquitectura, así como también la implementación de un sistema *middleware* prototipo llamado ActiveCloud. El *middleware* prototipo permitió comprobar la viabilidad de la arquitectura propuesta, así como también realizar pruebas de funcionalidad en cada una de las etapas de diseño. Como parte de la evaluación se diseñó y aplicó un experimento utilizando ActiveCloud, en donde usuarios potenciales utilizaron el *middleware* prototipo para realizar una aplicación de control. El presente trabajo es un paso más hacia la consolidación de una arquitectura agnóstica al *hardware*, permitiendo que los desarrolladores no se preocupen por la manera en la que interactúan los distintos dispositivos que forman la red.

Palabras clave: Redes Inalámbricas de Sensores y Actuadores, Arquitecturas Modulares, Actores, Internet, *Middleware*.

ABSTRACT of the thesis presented by **César Olea Aguilar**, as a partial requirement to obtain the MASTER SCIENCE degree in COMPUTER SCIENCE. Ensenada, Baja California. September 2008.

Modular Architecture for Wireless Sensor and Actuator Networks

Abstract approved by:

Dr. José Antonio García Macías
Thesis director

Currently, an increasing array of devices are being equipped with Internet networking capabilities. The Internet of Things, where connected devices are able to monitor and modify their environment, differs from the traditional Internet model. Wireless sensor and actuator networks, with their monitoring and acting capabilities, are a key technology for the realization of this Internet of Things. However, low-level device knowledge is required in order to program applications for such networks, resulting in platform-dependant implementations. This thesis proposes a modular, high level and device consistent programming interface architecture, providing dynamic support for those devices that comprise the network.

An architecture and prototype middleware system called ActiveCloud was designed and implemented. The prototype middleware system allowed to verify the feasibility of the proposed architecture and to perform a functional evaluation during each design stage. As part of the evaluation, an experiment using ActiveCloud was designed and implemented. This experiment was conducted with potential users, who developed a control application using the prototype middleware system. This work it's a step further in the consolidation of a hardware agnostic architecture, eliminating the need of device interaction knowledge.

Keywords: Wireless Sensor and Actor Networks, Modular Architectures, Actors, Internet, Middleware.

Dedicatoria

A mi Mamá y a mi Papá. Son mi modelo a seguir de toda la vida.

Los quiero mucho.

Agradecimientos

A mi asesor, Dr. J. Antonio García Macías,por el apoyo mostrado a lo largo de mi estancia en el CICESE. También por crear un ambiente de trabajo muy agradable.

A los miembros de mi comité, Dr. José Alberto Fernández Zepeda, Dr. Jaime Sánchez García y M.C. Raúl Tamayo Fernández, por sus revisiones, comentarios y ayuda para lograr que el presente trabajo refleje mi mejor esfuerzo.

A mis padres y mis hermanos, que siempre mostraron interés en lo que hacía, y nunca dejaron de repetirme "¡echale ganas! ya falta poco"

A mi novia Carolina y su hermana Selene, que hicieron de su casa en Tijuana mi segundo hogar y siempre estuvieron pendientes de mi. Las quiero mucho, gracias por todo.

A todos mis amigos dentro de CICESE. A la generación 2006 por su amistad y todos el tiempo compartido a lo largo de la maestría. También a mis compañeros de la generación 2007 por participar en la evaluación...

...en particular al M.C. y próximo DR. Edgardo Avilés, gran amigo, gran compañero y gran persona. Siempre dispuesto a ayudar desde el primer día de clases hasta el último.

También agradezco a mi buen amigo Rafael Pérez, por su ayuda con algunas de las imágenes que aparecen en la tesis, sus muchos comentarios, revisiones y sugerencias. Su ayuda siempre es invaluable y por supuesto, bienvenida.

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.

Ensenada, México
17 de Septiembre de 2008.

César Olea Aguilar

Tabla de Contenido

Capítulo Página

Tabla de Contenido (Continuación)

Tabla de Contenido (Continuación)

Lista de Figuras

Lista de Figuras (Continuación)

Lista de Tablas

Capítulo I

Introducción

En sus orígenes, la Internet se concebió para habilitar la transferencia de datos no solo entre computadoras incompatibles, sino también entre distintas redes de computadoras (Hauben, 1998). Esta red de redes proporciona un medio adecuado para la comunicación de datos entre dos computadoras que se encuentran, posiblemente, distribuídas alrededor del mundo. Sin embargo, al día de hoy ha emergido un «nuevo modelo» de Internet, la Internet «de cosas» (*Internet of Things*) impulsado principalmente por la adopción masiva de la tecnología de identificación por radio frecuencia (RFID por las siglas en Inglés de *Radio Frequency Identification*) y la proliferación de dispositivos de cómputo portátiles como teléfonos celulares y nodos de redes inalámbricas de sensores. Bajo este modelo, los dispositivos de cómputo conectados a Internet pueden monitorear, reaccionar y afectar el estado del ambiente que les rodea (Dickerson *et al.*, 2008).

Una de las tecnologías clave para este Internet «de cosas» son las redes inalámbricas de sensores y actuadores (WSAN por las siglas en Inglés de *Wireless Sensor and Actuator Networks*). Dichas redes cuentan con una gran cantidad de nodos de sensado (potencialmente en el orden de los cientos o miles) y nodos actuadores en menor medida. Las redes WSAN tienen la capacidad de monitorear el ambiente y modificarlo de acuerdo a los datos obtenidos. Sus características hacen de las redes WSAN un medio muy atractivo para controlar procesos distribuídos en áreas amplias, integrando los datos generados por estos procesos a otras aplicaciones a través de la red. Estas redes presentan una convergencia entre la obtención de datos y la teoría de control (Melodia, 2007); los actuadores que forman parte de la red le permiten modificar el ambiente en el que se encuentra, mientras que los sensores proporcionan la información de retroalimentación generada por la acción de

los actuadores, misma que es necesaria para controlar un proceso.

Sin embargo, el desarrollo de aplicaciones para redes WSAN requiere de programación a distintos niveles de abstracción entre dispositivos heterogéneos, lo cual representa un problema. En (Römer y Mattern, 2004) los autores describen que "programar sensores involucra el uso de lenguajes de programación de bajo nivel para comunicar las instrucciones al *hardware*. Por lo tanto existe una fuerte necesidad de abstracciones de programación que simplifiquen el asignar tareas a los sensores y de *middleware* que soporte tales abstracciones". Para desarrollar aplicaciones que proporcionen la funcionalidad de una WSAN aún es necesario ser experto en el área, desarrollando con herramientas especializadas para redes WSAN.

Como se mencionó anteriormente, la principal aplicación que se le da a las redes WSAN es la de crear sistemas de automatización y control distribuído. En un ambiente limitado (por ejemplo, una red de prueba emplazada en un laboratorio) es posible contar con nodos del mismo tipo, modelo y fabricante, además de los mismos sensores y actuadores. Sin embargo, en aplicaciones prácticas este escenario es muy improbable, haciendo el uso y mantenimiento de estos tipos de nodos distintos (es decir, una red con dispositivos heterogéneos) una limitante mayor debido a la densidad de la red (número de nodos contenidos en un área específica) y su extensión (área total que cubre la red). La arquitectura de las redes WSAN debe soportar el crecimiento de la red a gran escala, lo cual es limitado debido en parte a la dificultad de su integración con sistemas existentes.

También, las redes WSAN son de naturaleza dinámica. Por dinámica se entiende que la topología de la red puede cambiar por varias razones: una estrategia de ahorro de energía, una falla en el dispositivo o por la integración de nodos móviles, entre otras posibilidades. Además, el bajo costo de los nodos que la componen aunado a la escasa fuente de alimentación en cada uno de ellos, propicia el uso de nodos redundantes.

I.1. Planteamiento del problema

Debido a las grandes restricciones del *hardware* que compone a los nodos de una red WSAN, así como a la necesidad de dar soporte a una variedad de actuadores en la red, la programación de aplicaciones involucra conocer a fondo el *hardware* a manejar, haciendo este proceso difícil, tardado y propenso a errores. Además, debido al dinamismo de las redes WSAN, existe una necesidad de integración de nuevos dispositivos a la red una vez que ésta ya ha sido emplazada y puesta en funcionamiento sin tener que apagar la red o retirar los nodos de su lugar de emplazamiento (soporte dinámico). Esto representa un problema debido al gran número de dispositivos que la componen y a la dificultad de cambiar o incluir nuevos dispositivos una vez que se ha hecho la programación de los mismos (dependencia de la plataforma).

También existe una necesidad de herramientas que simplifiquen la programación de aplicaciones, tema que ha sido abordado por diversos trabajos de investigación (Sgroi *et al.*, 2005, Aberer *et al.*, 2006, Avilés-López y García-Macías, 2007) como se ve en el Capítulo IV. El manejo de *hardware* heterogéneo, la estandarización de las interfaces de programación, el emplazamiento del *hardware* de la red y las aplicaciones que se ejecutan dentro de los nodos, así como la dependencia de la plataforma son problemas del área que han sido poco estudiados.

Finalmente, la programación de *hardware* heterogéneo representa un problema para los desarrolladores de aplicaciones debido a diferencias e incompatibilidades entre las interfaces de programación de los dispositivos (sus interfaces de programación no son consistentes entre sí). Este problema ha sido poco explorado en trabajos anteriores. La principal problemática estudiada en el presente trabajo es el soporte dinámico y de alto nivel a *hardware* heterogéneo en redes WSAN.

I.2. Objetivo

El objetivo general del presente trabajo es desarrollar una arquitectura que permita la programación en alto nivel de aplicaciones para redes WSAN, brindando soporte dinámico a *hardware* heterogéneo.

I.2.1. Objetivos específicos

- Implementar la arquitectura propuesta.
- Proveer un mecanismo para permitir que las aplicaciones creadas con esta arquitectura utilicen un formato de datos estándar, mismo que habilita el intercambio de información entre distintos sistemas.
- Proveer una plataforma para el uso de *hardware* heterogéneo y la obtención de datos provenientes de distintas fuentes utilizando una interfaz de programación consistente.
- Permitir la programación de aplicaciones para redes WSAN ocultando detalles de bajo nivel de la red y del *hardware* que la compone.
- Evaluar el nivel de aceptación de la arquitectura propuesta con usuarios potenciales del sistema.

I.3. Preguntas de investigación

Con la realización de este trabajo se pretende responder las siguientes preguntas de investigación:

- ¿Es posible subir el nivel de abstracción al desarrollo para redes WSAN, de tal manera que sea posible crear aplicaciones desconociendo detalles de bajo nivel de la red y

del *hardware* que la compone?

- ¿Cuál es el grado de utilidad de un sistema que permita la programación de aplicaciones para redes WSAN con soporte dinámico a *hardware* heterogéneo?
- ¿Cuál es el grado de esfuerzo que se requiere para crear aplicaciones con la arquitectura propuesta?

I.4. Metodología

El presente trabajo se llevó a cabo en varias etapas, siguiendo una metodología de investigación propuesta originalmente para sistemas de cómputo ubicuo en el ámbito médico (González *et al.*, 2004), modificada parcialmente para adecuarla a los requerimientos del diseño de aplicaciones con redes WSAN y a los recursos con los que se contaron para la realización de este trabajo.

I.4.1. Familiarización inicial

Inicialmente se hizo una revisión exhaustiva de la literatura disponible sobre el tema de las redes inalámbricas de sensores y las redes inalámbricas de sensores y actuadores con la finalidad de conocer las similitudes, diferencias y los requerimientos únicos de las redes WSAN. Durante esta etapa se determinaron áreas de oportunidad así como también los requerimientos que debe cumplir una arquitectura para redes WSAN, provenientes tanto de la literatura revisada como por experiencia previa en el área de las redes WSN.

I.4.2. Escenarios de uso

Una vez identificadas las áreas de oportunidad y comprendidos los requerimientos que debe cumplir una arquitectura para redes WSAN, se propusieron algunos escenarios de uso. Los escenarios de uso representan un proceso que se puede controlar mediante el

uso de una red WSAN. Un ejemplo de escenario de uso propuesto es: “Se tiene una red WSAN instalada en un invernadero y se desea controlar el nivel de luminosidad que reciben las plantas al medio día; para lograrlo, se cuenta con unas compuertas eléctricas que se pueden abrir o cerrar según se necesite, así como también con varios sensores de luminosidad. Los dueños del invernadero desean recibir un correo electrónico cada vez que las compuertas se abran o se cierren, además de llevar un registro histórico de los niveles de luz a cada hora del día”.

I.4.3. Información obtenida

Con la información obtenida de la familiarización inicial y los escenarios de uso, se realizó el planteamiento del problema y, en consecuencia, se definió el objetivo general así como también los objetivos específicos del presente trabajo. El objetivo intenta resolver la problemática identificada en el planteamiento del problema.

I.4.4. Diseño e implementación de un prototipo

Una vez definido el objetivo, se comenzó a diseñar la arquitectura y, en paralelo, a realizar la programación de la misma. La arquitectura se diseñó analizando trabajos anteriores y retomando conceptos que han demostrado ser útiles para el problema a resolver. El contar con una implementación de referencia a la vez que se diseñaba la arquitectura permitió realizar pruebas funcionales en cada etapa del diseño, aunque esto resultó en un período extendido de desarrollo, debido a los constantes cambios en el diseño de la arquitectura.

I.4.5. Evaluación

Finalmente se diseñaron y realizaron pruebas ante un grupo de usuarios potenciales. Estos usuarios utilizaron el prototipo desarrollado de la arquitectura propuesta para crear

una aplicación con una red WSAN, con el fin de determinar si se lograron los objetivos establecidos. Lo anterior permitió conocer errores no previstos en el diseño y la arquitectura propuesta, obteniendo información con la cual se establecieron propuestas de trabajo futuro.

I.5. Contenido

El documento se encuentra organizado de la siguiente manera: en el Capítulo II se da una descripción a fondo sobre las redes inalámbricas de sensores, sus componentes, su funcionamiento, aplicaciones y trabajo futuro en dicha área. También en el Capítulo II se da una introducción a las redes WSAN con el objetivo de hacer una correlación entre ambas redes. El Capítulo III se centra sobre las redes WSAN y el proceso de desarrollo de aplicaciones para estas redes. En este capítulo se discute la problemática existente en las redes WSAN, lo cual lleva a la definición de los objetivos del presente trabajo. La arquitectura propuesta se describe en el Capítulo IV, detallando los requerimientos a cubrir. Se hace una revisión de trabajos anteriores relevantes y se presentan los componentes principales de la arquitectura. El Capítulo V trata sobre la implementación de un *middleware* prototipo basado en la arquitectura propuesta, la plataforma de *hardware* utilizada y los componentes que se programaron. La evaluación de la arquitectura se presenta en el Capítulo VI, en donde se da a conocer la metodología que se siguió para realizar la evaluación, así como los resultados obtenidos. Por último, en el Capítulo VII se dan a conocer las conclusiones generales, las aportaciones hechas por la realización de este trabajo y una guía para aquellas personas que deseen seguir por la misma línea de investigación aquí presentada.

Capítulo II

Redes Inalámbricas de Sensores

Las redes inalámbricas de sensores se están convirtiendo cada vez más en un medio atractivo para monitorear condiciones ambientales, así como también acortar la brecha entre el mundo físico y el virtual. Una red de sensores consiste en un gran número de nodos pequeños (llamados motas) con una capacidad de procesamiento muy limitada, comunicación inalámbrica y sensado de parámetros físicos. Las áreas de aplicación para las redes de sensores incluyen monitoreo geofísico, agricultura de precisión, monitoreo de hábitat, monitoreo de transporte, sistemas de defensa militares, procesos de negocios, y en el futuro posiblemente la cooperación con objetos inteligentes de la vida diaria (Römer *et al.*, 2002).

El concepto de una red inalámbrica de sensores es muy poderoso. Una red de este tipo es, en teoría, capaz de configurarse de manera automática al ser emplazada. Idealmente la red puede funcionar de manera autónoma, realizando las actividades para las que fue programada con poca o nula intervención humana. Como se verá en la Sección II.5, las aplicaciones para una red inalámbrica de sensores son muchas y muy variadas, sin embargo, existen ciertos retos tecnológicos que impiden una mayor adopción de esta tecnología, particularmente en el área de almacenamiento eficiente de energía, miniaturización, diseminación de datos eficiente en el consumo de energía, así como también en su facilidad de uso, emplazamiento y programación, área que sólo recientemente ha sido investigada.

Existen actualmente varias plataformas de *hardware* y *software* para la construcción de redes inalámbricas de sensores. Los componentes que forman una red inalámbrica de sensores poseen una arquitectura que la guía principalmente las fuertes restricciones en

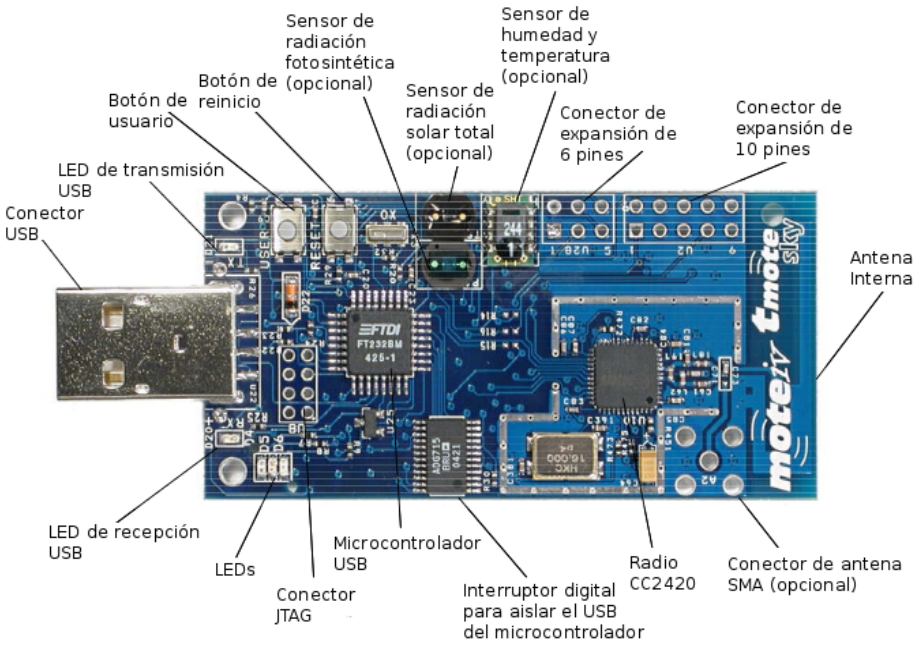
el consumo de energía y el tamaño del *hardware*. Sin embargo, no sólo es necesaria una arquitectura de *hardware* pensada en el ahorro de energía, también se requiere un fuerte acoplamiento en los sistemas de *software* utilizados para «manejar» este *hardware*. El resultado es una arquitectura de *hardware* eficiente en el uso de energía, además de sistemas de *software* con los que se pueden diseñar protocolos eficientes, utilizando distintas estrategias de ahorro de energía como bajar el voltaje de alimentación, apagar dispositivos independientes, distintos estados de «hibernación» del *hardware*, entre otras.

II.1. Anatomía de una red inalámbrica de sensores

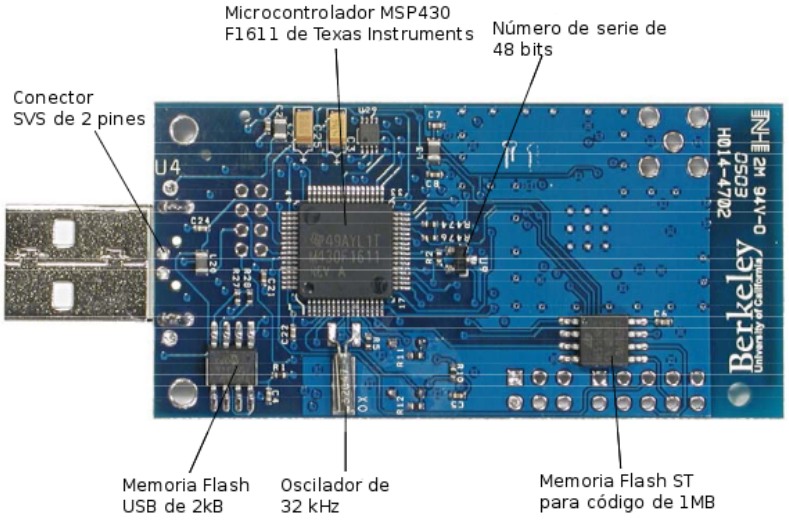
El *hardware* que compone a una red inalámbrica de sensores está compuesto por pequeños nodos con recursos de cómputo muy limitados, los cuales utilizan baterías como fuente de energía. Un nodo típico contiene una unidad de procesamiento, sensores de distintos parámetros analógicos como luminosidad, aceleración, humedad o temperatura, un bloque de memoria RAM y una unidad de radio para fines de comunicación del nodo con otros dispositivos (ver Figura 1).

Por el lado del *software*, cada nodo ejecuta localmente una copia del programa de la red. Dicho programa hace las veces de sistema operativo del nodo manejando los recursos disponibles, además de ejecutar las tareas para las cuales fue programada la red. Normalmente el *software* que se ejecuta en una red de sensores es altamente dependiente de la aplicación, la topología de la red, el protocolo de comunicación y el *hardware* de los nodos, resultando en aplicaciones que son poco portátiles y difíciles de programar.

Cada nodo viene equipado con capacidades de sensado. En algunos casos el nodo y los sensores se encuentran en la misma placa (como en la plataforma «Tmote», ver figura 1(a)) mientras que en otros (como en la plataforma «Mica2», ver Figura 2(a)), los sensores se encuentran en una placa separada (ver Figura 2(b)) lo que permite cambiar de sensores



(a) Parte delantera



(b) Parte trasera

Figura 1: **Componentes que forman un nodo de la plataforma Tmote**

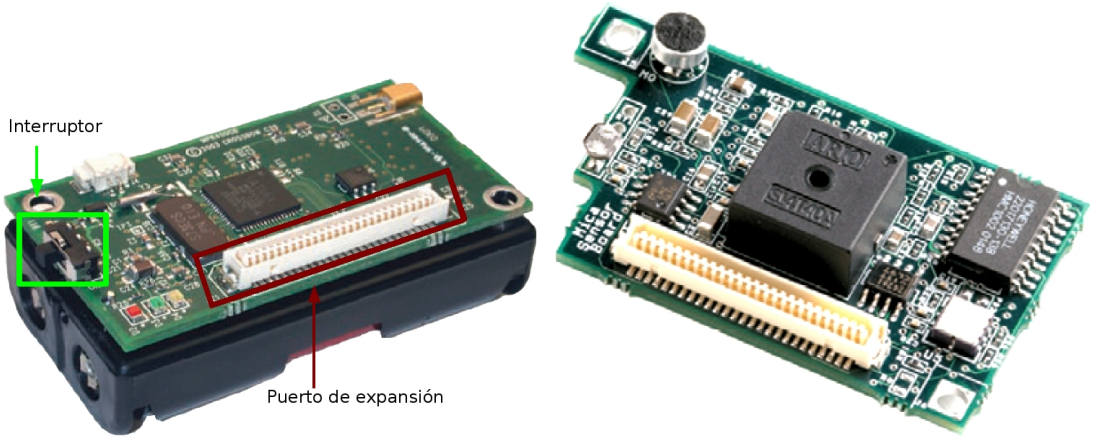
de acuerdo a la tarea a realizar. Algunos de los sensores disponibles en el mercado son:

- Acelerómetro.
- Luminosidad.
- Presión barométrica.
- GPS.
- Campo magnético.
- Humedad.
- Temperatura.

Ciertas placas de sensores cuentan también con actuadores y/o dispositivos de entrada y salida de propósito general como:

- Bocina (*Buzzer*).
- Puerto serie universal (USB por sus siglas en Inglés).
- Convertidor analógico - digital.
- Micrófono.
- Ultrasonido.

Además de los nodos regulares, una red inalámbrica de sensores se compone por uno o más nodos pasarela (*gateway*). El nodo pasarela es un dispositivo con menos limitantes que los nodos regulares, ya que se encuentra fijo y conectado a una fuente de alimentación contínua. El objetivo de contar con un nodo pasarela es servir como punto de entrada a los servicios de la red, traduciendo las peticiones del cliente externo y ejecutando las instrucciones necesarias en los nodos de la red de sensores. Otra de las funciones de un nodo pasarela es actuar como un nodo sumidero (*sink*), en donde toda la información proveniente de la red se agrega y posiblemente se procesa antes de ser puesta a disposición



(a) Nodo de la plataforma Mica2 (b) Placa de sensores para un nodo Mica2

Figura 2: **Componentes de un nodo de la plataforma Mica2**

del usuario final. Un nodo pasarela puede ser un nodo regular conectado físicamente a otro dispositivo fijo (como una computadora) el cual le proveé una fuente de alimentación y conectividad al mundo exterior, o también puede tratarse de un dispositivo completamente independiente.

Actualmente el *hardware* disponible para nodos pasarela va más allá de simples nodos regulares conectados a otro dispositivo. La pasarela «Stargate NetBridge» producida por la compañia Crossbow Technology ejecuta el kernel Linux como sistema operativo. Este dispositivo tiene la capacidad de conectar la red de sensores a una red Ethernet existente, y posee herramientas de *software* para el manejo de los nodos de la red y para visualización de datos, todo integrado en el mismo dispositivo.

La arquitectura del *hardware* diseñado para formar una red inalámbrica de sensores hace énfasis especial en el manejo eficiente de la energía. Un ejemplo del diseño enfocado al manejo eficiente de la energía es la alimentación independiente de cada uno de los subsistemas, de tal manera que la interfaz de radio se enciende solamente cuando se va a transmitir un dato, los sensores sólo consumen energía cuando es necesario capturar datos, etc (Hill *et al.*, 2004).

Uno de los componentes que merece mayor atención es la interfaz de radio. La función principal de la interfaz de radio es convertir bits a ondas de radio para su transmisión por el aire, así como también recibir por medio de una antena dichas ondas para su conversión de vuelta a bits. Gracias a la interfaz de radio, los nodos pueden comunicarse de manera inalámbrica; sin embargo, esto representa un problema: activar la interfaz de radio es una de las operaciones que consumen más energía del nodo, además de la memoria y algunos tipos de sensores (Karl y Willig, 2005).

II.2. Funcionamiento

Similar a lo que sucede con la investigación en cómputo ubicuo, en la literatura sobre redes inalámbricas de sensores se hace uso recurrente de escenarios «típicos» que pueden llegar a ser posibles en un futuro, gracias a los frutos obtenidos de la investigación en dicha área. Uno de los escenarios propuestos es el de la red emplazada via aérea o *air-dropped network*. En este escenario, los cientos o miles de nodos que componen la red son emplazados en el área de interés por medio de un avión que sobrevuela el terreno, simplemente «tirando» los nodos. Una vez en el suelo, los nodos comienzan una fase de descubrimiento y configuración para así comenzar a reportar los datos sensados a su estación base o nodo pasarela.

Sin embargo, el escenario descrito anteriormente está muy lejano de ser posible y, otra vez de manera similar a lo que sucede en el cómputo ubicuo, la investigación que se ha desarrollado en torno a dicho escenario es beneficiosa por los aciertos intermedios y no tanto por llevar a la realidad lo descrito. Actualmente para poder utilizar una red inalámbrica de sensores (sin ayuda de *middleware* especializado, ver Capítulo III), primero es necesario programar los nodos que componen la red con una copia del programa a ejecutar, para después colocar manualmente los nodos en el lugar de interés (a esto se le llama «emplazar»

o «desplegar» la red), encender los nodos y verificar que se puedan comunicar con el nodo pasarela. Por último los datos recopilados en el nodo pasarela deben ser accesibles al usuario final para ser explotados típicamente por otra aplicación. Sin duda los pasos necesarios para poner en funcionamiento una red inalámbrica de sensores son más complejos que con la *air-dropped network*, inclusive algunos autores (Tanenbaum *et al.*, 2006) concluyen que la «mítica» red emplazada vía aérea es inviable.

Como se mencionó anteriormente, el primer paso para utilizar una red de sensores es diseñar la topología de la red. Tanto la densidad de la red (la relación entre el número de nodos y el área cubierta) como el patrón físico en el que son dispuestos los nodos (topología) va a tener un impacto directo en el rendimiento de la red. Una vez decidida la topología y establecidos los valores de sus variables, cada uno de los nodos se programan individualmente. Debido a la densidad de la red, la programación de los nodos puede ser una tarea muy demandante.

Una vez que se tiene la topología y los nodos han sido programados, se inicia el proceso de emplazamiento de la red, esto es, colocar los nodos en el lugar que se desea monitorear. Si la densidad de la red es muy grande, esta tarea puede llegar a ser muy demandante, de ahí que se busquen alternativas tales como la red emplazada desde el aire de la que se habla en (Tanenbaum *et al.*, 2006). Al contar con la red emplazada, ésta puede comenzar a sensar y reportar los datos al nodo pasarela o estación base. En la Figura 3 se observa a grandes rasgos cada uno de los componentes de una red inalámbrica de sensores.

Todo el trabajo necesario para poner en funcionamiento una red inalámbrica de sensores ha llevado a realizar investigación para lograr un emplazamiento rápido y sencillo. La idea es lograr que un usuario inexperto en el tema de redes inalámbricas de sensores sea capaz de poner una en funcionamiento sin conocer detalles de muy bajo nivel, similar a lo que sucede actualmente con las redes IP. Destacan en esta área los siguientes desarrollos:

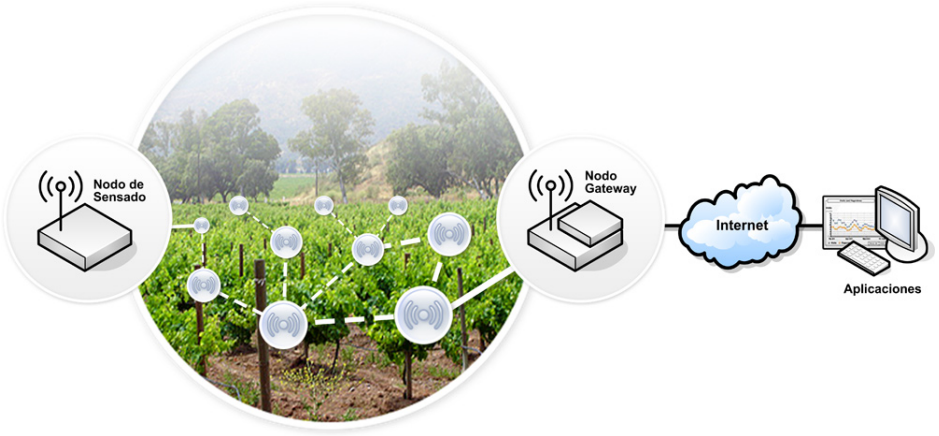


Figura 3: **Arquitectura típica de una red inalámbrica de sensores**

- *GSN. Global Sensor Networks* es un *middleware* que ataca los problemas de emplazamiento, disponibilidad y consulta de datos. GSN provee una plataforma uniforme para la integración de redes inalámbricas de sensores heterogéneas (integrada por motas de *hardware* distinto). En GSN se utiliza una abstracción denominada «sensor virtual», la cual encapsula los detalles sobre el acceso a los datos de los sensores. Para el emplazamiento se utiliza un documento XML en donde se definen los sensores virtuales disponibles así como también algunos aspectos de control de la red (Aberer *et al.*, 2006).

- *TASK.* El objetivo principal de TASK (*Tiny Application Sensor Kit*) es promover el uso de redes inalámbricas de sensores, proporcionando una manera sencilla de configurar y mantener una red del tipo de monitoreo ambiental, eliminando la necesidad de programar los nodos y de monitorear cada uno de los aspectos de bajo nivel de la red. TASK integra algunas aplicaciones creadas por el grupo de desarrolladores de TinyOS[1], en un paquete modular con una arquitectura por capas que interactúan por medio de interfaces bien definidas. Los nodos que integran una red TASK ejecutan el sistema TinyDB, que presenta a la red como una base de datos que se pueda consultar con un lenguaje similar a SQL llamado TinySQL. También, TASK hace uso de

[1] http://www.tinyos.net

una estación base llamada *Sensor Network Appliance* (SNA) que cumple con la misma funcionalidad del nodo pasarela, pero es totalmente autocontenido y no necesita configuración fuera de una conexión a la red y a una fuente de energía. Por último, TASK incluye herramientas de visualización e interacción con la red (Buonadonna *et al.*, 2005).

- *ArchRock Primer Pack*. De manera similar a TASK, el *Primer Pack* de la empresa Arch Rock[2] incluye herramientas para poner en funcionamiento una red inalámbrica de sensores en poco tiempo. Utiliza una arquitectura orientada a servicios (SOA por las siglas en Inglés de *Service Oriented Architecture*) y permite utilizar su interfaz de programación de aplicaciones (API por las siglas en Inglés de *Application Programming Interface*) para crear programas que exploten las capacidades de la red inalámbrica de sensores. Cada uno de los nodos que forma la red creada con el Primer Pack se encuentra ejecutando el sistema operativo TinyOS, además de ejecutar una pila TCP/IP que le permite a cada nodo formar parte de una red IP existente (como Internet) y utilizar herramientas comunes para el diagnóstico de redes (como *ping* o *traceroute*). Para emplazar una red con el Primer Pack sólo se necesita colocar los sensores en su lugar y encender la estación base (ArchRock, 2007).

II.3. Diferencias de las redes inalámbricas de sensores y las redes ad hoc

Una red inalámbrica de sensores tiene algunas similitudes con las redes ad hoc actuales, sin embargo, las redes inalámbricas de sensores tienen características únicas que impiden aplicar soluciones existentes en las redes ad hoc. Dichas características únicas son (Karl y Willig, 2003):

- Se prevé que la densidad de dispositivos sea mucho mayor en una red inalámbrica de sensores (cientos o miles) que en una red ad hoc tradicional.

[2] http://www.archrock.com

- Las capacidades de cómputo y de fuente de energía de los nodos de la red son mucho menores a aquellas de los dispositivos que integran una red ad hoc tradicional.
- Las redes inalámbricas de sensores son de aplicación especializada. Es muy poco probable que se encuentre una solución homogénea para todas las posibles combinaciones de uso de la red.
- Las características del tráfico de red que generan son distintas a redes más tradicionales, en donde el tráfico es generado principalmente como resultado de una intervención humana y se da en ráfagas aleatorias. Dichas características son distintas debido a que una red de sensores interactúa con el ambiente. Como consecuencia de lo anterior, un escenario típico es muy poco tráfico por períodos de tiempo largos, para ser seguido por períodos de tiempo cortos con gran tráfico.
- Una métrica principal a considerar en las redes de sensores es la energía. Frecuentemente la batería de un nodo no es recargable y el diseño de la arquitectura de la red, así como también los protocolos de la misma, deben de prolongar la vida útil de los sensores.
- Debido al bajo costo de los nodos, así como al ambiente hostil en donde se despliegan éstos, es muy probable que muchas aplicaciones requieran el emplazamiento de nodos redundantes. Como consecuencia, la importancia de un nodo en particular se ve disminuida en comparación a un nodo de una red tradicional. En una red inalámbrica de sensores, son más importantes los datos observados y transmitidos (red centrada en datos).

II.4. Redes Inalámbricas de Sensores y Actuadores (WSAN)

Como su nombre lo indica, una red inalámbrica de sensores y actuadores (WSAN por las siglas en Inglés de *Wireless Sensor and Actuator Network*) es aquella que incluye, además

de sensores, dispositivos capaces de modificar el ambiente que les rodea (actuadores) (Akyildiz y Kasimoglu, 2004). Una WSAN es capaz de observar el mundo físico por medio de sus sensores, procesar los datos observados, tomar decisiones basadas en la observación y tomar las acciones necesarias. Este último punto es crítico, ya que es la principal ventaja que tienen las WSAN sobre las WSN.

El hecho de poder tomar las acciones necesarias, aunque pareciera una diferencia poco significativa con respecto a las WSN, crea un dominio del problema muy distinto. En una WSAN es posible, y a veces necesaria, la comunicación y coordinación «sensor - actuador» o «actuador - actuador», lo que provoca un problema inexistente en las redes sin actuadores y que necesita ser investigado.

Además de lo mencionado anteriormente, las WSAN poseen otras características que las hacen únicas:

- Se utilizan típicamente para aplicaciones de control.
- Los actuadores son una parte integral de la red, así como también los métodos para controlarlos.
- El consumo de energía puede *no* ser la cuestión principal a tomar en cuenta.

Las características de las WSAN y los retos que representan se tratan más a fondo en el Capítulo III.

II.5. Aplicaciones

Las características únicas que poseen las WSN y las WSAN abren una amplia gama de posibilidades para su uso en distintas áreas. Algunas de las aplicaciones propuestas son (Römer y Mattern, 2004, Tanenbaum *et al.*, 2006, Akyildiz y Kasimoglu, 2004, Park *et al.*, 2007):

- *Observación de manadas de animales.* Una red de sensores se utiliza para observar el comportamiento de apareamiento de aves que son fácilmente perturbadas por la presencia humana. Otra red similar se utiliza para observar animales que viven en un hábitat natural muy extenso.

- *Monitoreo de glaciares.* Una red de sensores se utiliza para monitorear el comportamiento subglaciar con el fin de ampliar el conocimiento sobre el clima terrestre.

- *Monitoreo de aguas oceánicas.* El proyecto ARGO se encuentra utilizando una red de sensores para observar la temperatura, salinidad y perfil de la superficie del océano.

- *Monitoreo de viñedos.* Una red de sensores se utiliza para monitorear los factores que tienen influencia sobre el crecimiento de las uvas (por ejemplo temperatura, luminosidad, humedad del suelo y del ambiente). La red permite la recolección de la uva en el momento más adecuado de su crecimiento, así como adaptar la aplicación de agua, fertilizante y pesticida de acuerdo a los parámetros anteriores.

- *Rescate de víctimas de avalancha.* Se utiliza una red de sensores para asistir a grupos de rescate de víctimas de avalancha. La red de sensores le proporciona a los equipos datos sobre el estado de la víctima y ayuda a su localización y a establecer la prioridad de los rescates (basados en el ritmo cardiaco, actividad respiratoria y nivel de consciencia).

- *Campo minado autoajustable.* Minas anti-tanque están siendo equipadas con capacidades de sensado y comunicación para asegurar que un área sea cubierta aún cuando el enemigo modifique la localización de una mina para crear un camino libre a los tanques. Las minas forman una red de sensores capaz de estimar la localización y reorientación de los nodos.

- *Agricultura de precisión.* Una red de sensores se utiliza para monitorear parámetros de importancia para el cultivo en invernaderos. Mediante el estudio de los datos obtenidos, se pueden crear modelos que asistan en la práctica de la agricultura de

precisión, determinando los niveles óptimos del cultivo (llamado «microclima») y proporcionando alertas en tiempo real en caso de que algún parámetro se encuentre en niveles fuera de su intervalo normal.

- *Monitoreo de fronteras*. Se emplaza una red de sensores a lo largo de la frontera entre dos países con el objetivo de detección de intrusos. Para lograrlo se puede hacer uso de sensores de sonido o vibraciones en rangos cortos.

- *Control de microclima*. Una red de sensores y actuadores se emplaza en el lugar al que se le desee controlar el clima, como un hogar, invernadero o edificio. Las reglas para el control del clima varían de acuerdo al lugar, la exactitud requerida, y las preferencias del usuario (en el caso de que éstas apliquen).

- *Automatización de hogares*. Se utiliza una red de sensores para monitorear el ambiente del hogar. Dicha red se conecta a un dispositivo que actúa como pasarela, en donde converge toda la información proveniente de la red de sensores y otros dispositivos con capacidades multimedia (audio y video). El dispositivo pasarela tiene además la capacidad de controlar electrodomésticos, el sistema de iluminación y de gas, mediante el uso de actuadores.

El rango de aplicación es muy variado, sin embargo, las grandes limitantes en energía y poder de procesamiento restringen la utilidad de la red. Es necesario el desarrollo de métodos de comunicación de datos eficientes en el uso de energía, así como también una mejora en las tecnologías de alimentación y almacenamiento de energía. Además, para lograr una penetración de las redes inalámbricas de sensores en actividades de la vida diaria (es decir, fuera de la academia o la milicia) es necesario definir una arquitectura general que guie el desarrollo de aplicaciones con este tipo de tecnología (Culler *et al.*, 2005).

Una arquitectura es un juego de principios que guían la funcionalidad que se debe implementar, aunada a interfaces, componentes funcionales, protocolos y *hardware* que

sigue dichas guías. La evolución de Internet mostró cómo una arquitectura diseñada correctamente puede moldear la evolución de un sistema complejo sobre cambios significativos en la tecnología, la escala y el uso (Culler *et al.*, 2005). El área de investigación y la aplicación de las redes inalámbricas de sensores es, como se mencionó anteriormente, de aplicación especializada. Su evolución hasta ahora es similar a la situación de Internet a sus inicios (Staff, 2001).

Siguiendo con la analogía de Internet, el poder de éste se aprecia no al observar la implementación de cada uno de sus componentes específicos, sino al reconocer la habilidad para lograr un extremo crecimiento en escala y diversidad conforme la tecnología fue avanzando y se hizo utilizable. Los mismos principios: crecimiento en escala y diversidad de componentes, son conceptos primordiales en el diseño de las redes de sensores y debe ser uno de los objetivos que guíen el desarrollo de una arquitectura para éstas (Culler *et al.*, 2005).

II.6. Trabajo futuro

El área de las redes de sensores cuenta con mucha investigación y desarrollo. En la actualidad ya se utilizan redes inalámbricas de sensores para múltiples aplicaciones, como se describe en la Sección II.5. Aún así, existen muchas tecnologías que podrían mejorarse y otras con las que aún no se cuentan. A continuación se mencionan algunas líneas de investigación abiertas (Perrig *et al.*, 2004, Younis y Akkaya, 2008):

- *Diseño de nuevo* hardware. Es necesaria la investigación y el desarrollo de *hardware* más pequeño, con mayor capacidad computacional y de fuente de energía, además de una disminución en el costo de producción.
- *Seguridad.* El uso general de las redes inalámbricas de sensores requiere de estrategias de seguridad para la comunicación. Evitar posibles ataques a la red, similar a lo

que sucede actualmente en Internet.

- *Calidad en el servicio*. Definir mecanismos para ofrecer soporte a la calidad en el servicio, para habilitar el uso de las redes de sensores en situaciones de tiempo real.

- *Emplazamiento*: El proceso de programación y emplazamiento de los nodos es tedioso y propenso a errores. La reprogramación por el aire (*over-the-air programming*) elimina la necesidad de recolectar los nodos ya emplazados para corregir errores en su programación. Sin embargo, aún no se diseñan métodos para eliminar el emplazamiento manual de los nodos.

- *Relocalización y coordinación de nodos*: Algunas aplicaciones de las redes WSN involucran nodos móviles (por ejemplo detección y desactivación de minas en el campo de batalla o la exploración submarina). Ésta es un área que se ha explorado muy poco. Es necesario investigar sobre técnicas de coordinación entre nodos emplazados, y cómo afecta la movilidad de dichos nodos en su capacidad de coordinación.

II.7. Conclusión

Las redes inalámbricas de sensores ofrecen posibilidades muy atractivas para hacer realidad la visión del cómputo ubicuo. El uso actual que se le da a las redes de sensores es muy diverso, y comienza a dar señales de uso más allá de la academia, la investigación y el ambiente militar. Se visualiza un futuro en el cual los sistemas de cómputo ubicuo sean alimentados por datos obtenidos con redes de sensores de miles o millones de nodos, emplazadas a lo largo y ancho del planeta.

Una red inalámbrica de sensores se compone básicamente de varios nodos (con la posibilidad de ser una red heterogénea), además de una o más estaciones base o nodo pasarela. Los nodos de la red poseen capacidades de sensado que dependen del *hardware* utilizado. Algunas arquitecturas de *hardware* como los nodos «Mica2» permiten intercambiar distintas tarjetas de adquisición de datos, otras como la arquitectura «Tmote» integran los sensores en el mismo *hardware* del nodo.

Actualmente existe interés en mejorar la experiencia del usuario al momento de emplazar la red inalámbrica de sensores, esto para promover su uso en aplicaciones comerciales. El enfoque que se ha tomado es integrar el *hardware* necesario con un *middleware* que permita poner en funcionamiento la red con poco esfuerzo, de ahí la importancia de los sistemas *middleware* integrados a las aplicaciones de redes inalámbricas de sensores. Lo anterior permite emplazar una red sin contar con conocimientos extensos en su arquitectura o su funcionamiento, sin embargo restringe la posibilidad de aplicaciones para las cuales se puede utilizar la red ya que típicamente el *middleware* se optimiza para un solo tipo de aplicaciones. En el Capítulo III se trata más a fondo el tema de los distintos *middleware* para redes inalámbricas de sensores.

Capítulo III

Desarrollo para Redes Inalámbricas de Sensores y Actuadores

Las Redes Inalámbricas de Sensores y Actuadores presentan una convergencia entre la tecnología que integra a las redes WSN y la teoría de control (Melodia, 2007). Como se menciona en el Capítulo II, la WSAN se compone de nodos sensores y nodos actuadores (también aparecen en la literatura como nodos «actores»). Los actuadores le permiten a la WSAN modificar el medio en el que se encuentra, ampliando el espectro de posibles aplicaciones.

El concepto de redes WSAN no se puede ver tan solo como una extensión al de las redes WSN. La existencia de actuadores en conjunto a sensores introduce una nueva problemática de comunicación y coordinación. Como resultado de esto, las soluciones generadas por la intensa investigación en años anteriores alrededor de las redes WSN pueden no aplicarse directamente o del todo a las redes WSAN (Xia *et al.*, 2007).

Debido a esto, es necesario desarrollar nuevos algoritmos de comunicación y diseminación de datos, específicos para las redes WSAN. Estos nuevos algoritmos deben tomar en cuenta las principales características de una red WSAN, así como también aquellas limitantes que son relajadas en comparación a las presentes en las redes WSN. En el presente capítulo se exponen las características que posee una red WSAN, la problemática que genera la introducción de actuadores a la red, también se presenta una vista general acerca de la programación para redes WSAN y los sistemas de *software* disponibles a la fecha para asistir en el desarrollo para redes WSAN.

III.1. Características de una WSAN

Físicamente, una WSAN no es muy distinta de una WSN (salvo la integración de nodos actuadores): ambas son densamente pobladas, distribuídas en un área amplia. También, en ambas redes existe un nodo «pasarela» que sirve como punto de entrada a la red (como se describe en la Sección II.1). La red se compone de cientos o miles de nodos sensores de bajo costo, además de nodos actuadores en menor medida.

Típicamente tanto los nodos sensores como los actuadores que componen la red son estáticos, aunque algunos actuadores pueden ser móviles. Es importante notar que aún cuando todos los nodos son estáticos, la topología de la red cambia. Esto se debe a las estrategias de ahorro de energía: los nodos funcionan en períodos cortos de actividad denominados «ciclos de trabajo» (*duty-cyle*) (Sanchez *et al.*, 2007). Sin embargo, es posible que los nodos actuadores no funcionen en ciclos de trabajo, ya que normalmente son nodos con menos limitantes que los nodos sensores y cuentan con mayor capacidad de almacenamiento de energía, o se encuentran conectados a la energía eléctrica. Al espacio que cubre el emplazamiento de sensores y actuadores se le llama «campo de sensado / acción» (*sensor/actor field*). Las características físicas de las redes WSAN se muestran en la Figura 4 [1].

III.1.1. Nodos Actores

Un nodo actor o actuador en una WSAN es aquel que le permite a la red interactuar con el ambiente que le rodea, modificándolo por medio de la acción de dispositivos como motores, relevadores o interruptores. Estos nodos, a diferencia de los nodos sensores, cuentan

[1] Imagen redibujada del artículo «Wireless sensor and actor networks: research challenges» por Ian F. Akyildiz e Ismail H. Kasimoglu

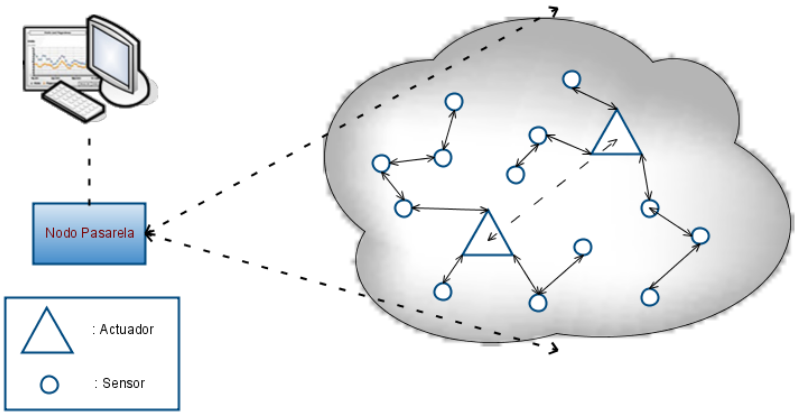


Figura 4: **Características físicas de una red WSAN**

con un mecanismo capaz de encender, apagar, ajustar o mover un dispositivo. Ejemplos de actuadores son:

- Motores eléctricos.
- Pistones hidráulicos.
- Válvulas.
- Relevadores.

Sin embargo, en una WSAN pueden existir dos tipos de «actuadores»: aquellos aumentados con capacidades de comunicación inalámbrica, pero que cuentan básicamente con las mismas restricciones que los nodos sensores, así como también nodos actuadores con altas capacidades de procesamiento, alto poder de transmisión y una fuente de energía mayor o que se encuentran conectados a la energía eléctrica (Melodia *et al.*, 2006).

La necesidad de diferenciar entre estos dos tipos de actuadores viene dada por la dinámica de comunicación que ofrece una WSAN equipada con el segundo tipo de actuadores. A este tipo se le denomina «actores»: nodos equipados con mejores capacidades de procesamiento, mayor poder de transmisión y almacenamiento de energía, capaces de tomar decisiones y ejecutar acciones sobre su ambiente (Akyildiz y Kasimoglu, 2004). Un actor

también puede ser la combinación de varios sensores y actuadores que forman una entidad más compleja, por ejemplo un brazo robótico o vehículos autónomos.

Tanto los nodos actores como los actuadores le proveen a la red la capacidad de *actuar* sobre su ambiente, es decir, las capacidades que introducen a la red son las mismas. Existe, sin embargo, una problemática introducida solamente por los nodos actores: la necesidad de *coordinación*. Es importante resaltar el hecho de que para la red, un nodo actor debe representar una sola entidad, aún cuando se componga de varios sensores y/o actuadores.

III.2. Comunicación en una WSAN

Con el fin de determinar el origen de un evento de interés, es necesario que exista comunicación entre sensores y actuadores. Al proceso de establecer una ruta entre sensores y actuadores se le llama «coordinación sensor - actor». Una vez que el evento ha sido detectado, los actores deben colaborar para tomar una decisión sobre cómo realizar la acción, a lo cual se le denomina «coordinación actor - actor» (Melodia *et al.*, 2007).

La comunicación de datos dentro (entre nodos) y fuera (hacia la estación base) de la red es de gran importancia. Para ser útil, la red debe reportar los datos sensados de manera confiable. Para garantizar la fiabilidad de los datos, éstos deben llegar a su destino a tiempo, de manera que al momento de actuar sobre el evento detectado basados en los datos obtenidos, dichos datos aún sean válidos. Las redes WSAN se pueden ver como un sistema de control distribuído, diseñadas para reaccionar a tiempo y con una acción adecuada a los datos obtenidos por los sensores (Melodia *et al.*, 2007).

Esta necesidad de comunicación y coordinación entre nodos sensores y actuadores, crea dos paradigmas de comunicación de datos o *arquitecturas* en las redes WSAN (Akyildiz y Kasimoglu, 2004):

- *Automática:* no existe una entidad de control explícita. Los nodos sensores envían datos a los actores (coordinación sensor - actor) y éstos realizan la acción correspondiente (coordinación actor - actor), de acuerdo a algún algoritmo de control. Los actores toman el rol de «controladores». Esta arquitectura puede verse en la Figura 5(a).

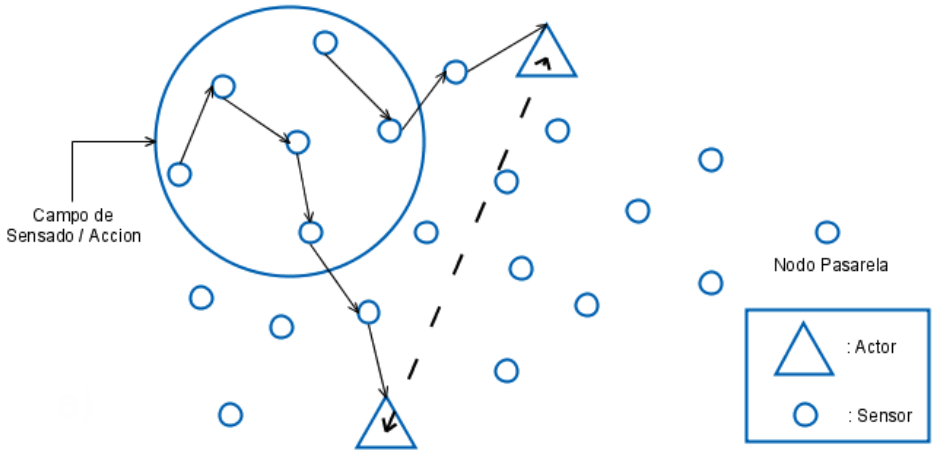
- *Semiautomática:* existe una entidad de control explícita. Los nodos sensores recolectan datos y los envían a dicha entidad de control (típicamente el nodo pasarela), la cual ejecuta un algoritmo de control para determinar las acciones necesarias. Finalmente envía las acciones resultantes a los actores, que se encargan de realizar la acción. Esta arquitectura se ilustra en la Figura 5(b).

Independientemente de la arquitectura de la red, la comunicación de los datos sensados y de las acciones a realizar es uno de los problemas investigados con mayor intensidad. El ahorro de energía, que es la principal preocupación en una red WSN, depende en su mayoría de un algoritmo de comunicación eficiente. La arquitectura semiautomática de las redes WSAN es más parecida a la arquitectura de las redes WSN, es por eso que algunos algoritmos de comunicación se pueden aplicar directamente o con pequeñas modificaciones a las redes WSAN. Sin embargo, la arquitectura automática posee algunas características que la hacen muy atractiva (Akyildiz y Kasimoglu, 2004):

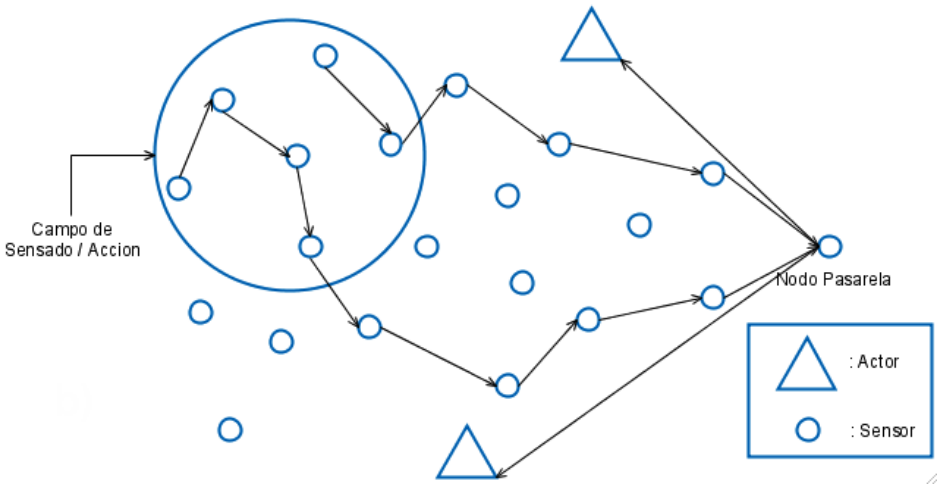
- *Baja latencia:* La información recabada en los sensores se envía directamente a los actores, eliminando la necesidad de enviarlos al nodo pasarela. Debido a que es más probable que los nodos sensores y actuadores se encuentren más cerca unos de otros que con respecto al nodo pasarela, la latencia disminuye.

- *Mayor tiempo de vida de la red:* En la arquitectura semiautomática, los nodos cercanos al nodo pasarela tienden a quedarse sin energía más rápido, debido a que sin importar donde se generó el evento los datos sensados siempre pasan por estos nodos, como se muestra en la Figura 5(b). En la arquitectura automática, los nodos cercanos a los actores se encuentran sujetos a esta misma situación. Sin embargo, es más probable que para cada evento distinto, se elija un juego distinto de actores, por

lo que el grupo de sensores cercanos a los actores también es distinto, distribuyendo la carga.



(a) Arquitectura automática



(b) Arquitectura semiautomática

Figura 5: **Distintas arquitecturas de las redes WSAN**

La diseminación de datos tanto en las redes WSN como en las redes WSAN con cualquier arquitectura se etiqueta como «centrada a datos». La teoría sobre las redes centradas a datos aplica en gran medida a las redes WSN y WSAN, debido a su enfoque en los datos y no en quien los recibe, así como también en la diseminación de datos en vez de una «conversación» (la cual es un caso especial de diseminación, cuando ésta es uno a uno) y

la necesidad de algoritmos localizados. También, el bajo costo de los nodos, aunado a la escasa fuente de alimentación en cada uno de ellos, propicia el uso de nodos redundantes. Como consecuencia de esto, la importancia de un nodo en particular reduce considerablemente, comparándola con la de una red tradicional. La importancia, como ya se dijo, se centra en los datos que este nodo puede observar.

Típicamente en los sistemas de control se utiliza la retroalimentación como medida entre lo que se espera y lo que realmente sucedió. La diferencia entre estos dos parámetros se utiliza para ajustar el comportamiento total del sistema. Gracias a la utilización de la retroalimentación, las redes WSAN son *menos sensibles* a variaciones repentinas en los datos sensados. Esto significa que los requerimientos de comunicación pueden llegar a ser menos demandantes que los requeridos por las redes WSN. Diversas investigaciones han establecido límites bajo los cuales las aplicaciones de control distribuído pueden operar con un rendimiento aceptable (Montestruque y Lemmon, 2008).

III.2.1. Estructura jerárquica

Un problema presente en las redes inalámbricas sin infraestructura (redes ad hoc, WSN y WSAN) es el de su capacidad de envío de datos (*data throughput*). En una red de n nodos homogéneos, en donde los nodos cooperan para reenviar los datos, la capacidad de envío de datos de cada nodo disminuye asintóticamente al incrementar el número de nodos. Esto se debe a que cada nodo cede una parte de su capacidad de envío y lo dedica a reenviar paquetes a nodos vecinos (Gupta y Kumar, 1999). En una WSAN pequeña (el máximo número de saltos es entre 5 y 6), una jerarquía plana es suficiente. En una jerarquía plana, todos los nodos pertenecen a la misma red y todos cooperan en el reenvío de datos. Para redes con dimensiones mayores, una alternativa es la estructura jerárquica.

En la estructura jerárquica la red se divide en subredes más pequeñas. Cada subred

cuenta con un nodo pasarela que además de agregar los datos provenientes de la red WSAN, comunica a las distintas subredes por medio de una red de área amplia (WAN por sus siglas en Inglés). Cada subred cumple con los límites establecidos en (Gupta y Kumar, 1999), obteniendo un nivel aceptable en la calidad del servicio (QoS por las siglas en Inglés de *Quality of Service*). La conexión entre nodos pasarela a través de la red WAN puede ser por infraestructura (cuando ya se cuenta con cableado de red en el lugar de interés) o inalámbrica. A la combinación de redes inalámbricas de sensores con nodos pasarela conectados por infraestructura se le llama «redes híbridas».

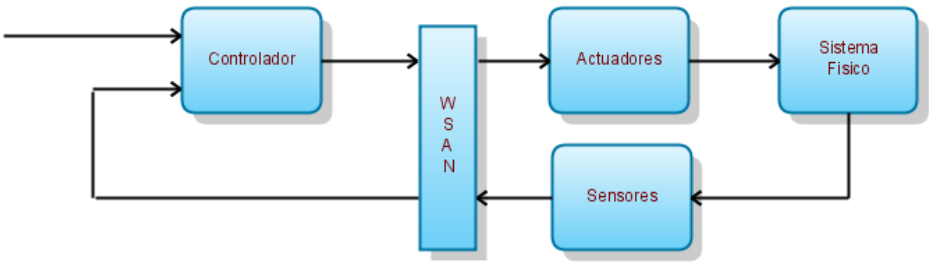
III.3. Automatización y control con redes WSAN

Las redes WSAN agregan una nueva dimensión a las aplicaciones potenciales de la tecnología de las redes inalámbricas de sensores: la capacidad de actuar. Debido a que los cambios provocados por la acción de un actor en el ambiente los pueden medir los nodos sensores, se genera una interacción de retroalimentación entre la red WSAN y el mundo exterior. Por esto último, la principal utilidad que se le da a las redes WSAN es la de crear sistemas distribuídos de automatización y control (Montestruque y Lemmon, 2008, Deshpande *et al.*, 2005, Xia *et al.*, 2007).

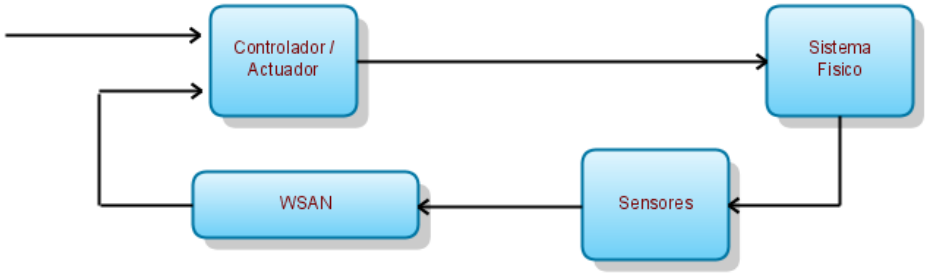
En la arquitectura semiautomática, existen una o más entidades de control. Usualmente estas entidades de control explícitas son los nodos pasarela, pero también pueden ser otros nodos equipados con suficiente energía, poder de procesamiento y comunicación. Las entidades de control ejecutan algoritmos de control, de los cuales se obtienen las acciones que se envían a los actores (Xia *et al.*, 2007). El flujo de trabajo es como se ilustra en la Figura 6(a): inicialmente los sensores obtienen datos de su medio ambiente. Estos datos se transmiten por la red inalámbrica formada por los nodos (unisalto o multisalto) a la entidad de control. Una vez en la entidad de control, se aplica un algoritmo de control y se generan una serie de acciones a ejecutar. Estas acciones se transmiten a los actores por

medio de la red inalámbrica. Finalmente los actores ejecutan las acciones de control que se les envían.

En la arquitectura automática, no existe una entidad de control explícita. Los datos obtenidos por los nodos sensores se transmiten directamente a los actores. La red inalámbrica se utiliza solamente para transmitir los datos obtenidos por los sensores, hacia los actores. Gracias a que el actor es también el controlador, las acciones de control las crea y ejecuta el mismo dispositivo (Xia *et al.*, 2007). El flujo de trabajo es como se ilustra en la Figura 6(b): los sensores obtienen datos del medio ambiente; estos datos se transmiten a través de la red inalámbrica a los actores que a su vez fungen como entidades de control. Finalmente los actores ejecutan las acciones de control, calculadas por ellos mismos.



(a) Control semiautomático



(b) Control automático

Figura 6: **Esquemas de aplicación de control con WSAN**

Las posibles aplicaciones de una WSAN para propósitos de control son procesos que se encuentren distribuídos físicamente en áreas extensas, y que se requieran controlar. Ejemplos de estos procesos son (Montestruque y Lemmon, 2008):

- Carreteras automatizadas.
- Grupos de vehículos autónomos.
- Control de tráfico.
- Control y tratamiento de aguas residuales.
- Procesos químicos automáticos.
- Sistemas de calefacción, ventilación y aire acondicionado (HVAC por las siglas en Inglés de *Heat, Ventilation and Air Conditioning*).
- Control de la red eléctrica.

En la mayoría de los casos las redes WSAN son heterogéneas, es decir, los nodos que la forman son dispositivos distintos entre si. Esto debido a las distintas acciones de control que debe soportar la red para llevar a cabo las tareas que le son programadas.

III.4. Programación

El desarrollo de aplicaciones para redes WSAN requiere de programación a distintos niveles de abstracción, entre dispositivos heterogéneos. La heterogeneidad del *hardware* disponible en una WSAN representa una dificultad para la programación de aplicaciones, ya que cada dispositivo puede tener su propia interfaz de programación de aplicaciones (API por las siglas en Inglés de *Application Programming Interface*), protocolo o paradigma de programación.

Existen varias herramientas (sistemas operativos, lenguajes de programación, ambientes de desarrollo, entre otras) enfocadas al desarrollo de aplicaciones para los nodos que componen las redes WSN y WSAN. Algunas de estas herramientas son:

- *nesC*: Es un lenguaje de programación, el cual es un dialecto derivado del lenguaje C, creado con el propósito de operar dentro de las fuertes restricciones del ambiente embebido de las redes de sensores. El lenguaje nesC tiene una sintaxis similar a C, pero incluye un modelo de concurrencia y mecanismos para ligar pequeños componentes de *software* y crear programas más complejos. Además, nesC introduce un concepto de «cableado» de componentes con el objetivo de crear código reutilizable.

- *TinyOS*[2]: Es un juego de herramientas y bibliotecas programadas en nesC que permiten el desarrollo de aplicaciones para redes de sensores. TinyOS no es un sistema operativo de propósito general para dispositivos embebidos, en vez de esto, se diseñan aplicaciones a partir de un conjunto de herramientas reutilizables, las cuales se ligan de manera estática con código específico para el dispositivo sensor.

- *Contiki*: Es un sistema que consiste de un núcleo (*kernel*), librerías, un cargador de programa y un juego de aplicaciones y servicios. Este sistema operativo para dispositivos de sensado muy limitados, como los presentes en las redes WSAN, es capaz de cargar y reemplazar programas en tiempo de ejecución, enviando la imagen binaria del programa por medio de la red inalámbrica (Dunkels *et al.*, 2004).

- *Sun SPOT*[3]: Es una plataforma de desarrollo que incluye nodos sensores que se pueden programar en el lenguaje Java[4]. Los nodos ejecutan una versión modificada de la máquina virtual de Java. El *hardware* se diseñó con la facilidad de programación en mente, a diferencia de otras plataformas en donde la eficiencia en el uso de energía es una de las principales consideraciones a tomar en cuenta. El uso de un lenguaje de alto nivel para la programación de nodos, en conjunto a la integración directa del lenguaje con un entorno gráfico de desarrollo, hacen de esta plataforma una de las más atractivas para la creación de prototipos rápidos.

[2] http://www.tinyos.net
[3] http://www.sunspotworld.com
[4] http://java.sun.com

La creación de herramientas para la programación de nodos actores es más complicada, debido a la naturaleza del *hardware* que los componen. Cada acción a realizar en una WSAN requiere de un tipo de actuador específico, con *hardware* especializado para la tarea a realizar. Debido a la falta de estandarización en el manejo de actuadores, es necesario crear el *software* que maneje a este *hardware* y permita la comunicación con sistemas existentes. Por lo mismo se han creado sistemas *middleware* que tratan de homogeneizar las interfaces de programación y proporcionar una plataforma de *software* adecuada para la creación de aplicaciones con WSANs.

Un sistema *middleware* es aquel que sirve como «pegamento» entre dos aplicaciones de *software*. Estos sistemas proveen una API con el propósito de hacer transparente la comunicación entre aplicaciones distintas, generalmente entre una API de bajo nivel y otra de más alto nivel de abstracción. De esta manera los programadores pueden acceder a la funcionalidad proporcionada por un sistema de bajo nivel, sin la necesidad de comprender detalladamente el funcionamiento de este sistema (Avilés, 2006).

Un ejemplo de sistema *middleware* para el manejo de una red WSAN es CSOnet, creado por una alianza entre la industria privada (EmNet LLC), pública (la ciudad de South Bend, Indiana, Estados Unidos) y académica (la Universidad de Notre Dame y la Universidad de Purdue) (Montestruque y Lemmon, 2008). La arquitectura de la red WSAN utilizada en este proyecto es algo distinta a una WSAN tradicional. Esta red se compone de 4 tipos de nodos:

- *Nodo de instrumentación (INode):* Son los nodos que cuentan con sensores, responsables de obtener datos del medio ambiente y enviarlos por medio de su transmisor de radio.
- *Nodo de propagación (RNode):* Estos nodos se dedican solamente a propagar mensajes provenientes de los INodes hacia los nodos pasarela. Estos nodos mejoran la conectividad de la red inalámbrica.

- *Nodo pasarela (GNode):* El nodo pasarela provee conectividad entre la red WSAN y una red distinta, típicamente una red de área local (LAN por las siglas en Inglés de *Local Area Network*) o de área amplia (como Internet).
- *Nodo actuador (ANode):* Estos nodos están conectados a actuadores, permitiendo su utilización dentro de la red WSAN.

Todos los nodos que componen a la red CSOnet comparten el mismo *hardware* para envío y recepción de datos, procesamiento y almacenamiento de energía. La diferencia radica en las tareas únicas que cada nodo debe llevar a cabo. Para el caso de los nodos de instrumentación, estos integran una unidad de sensado, mientras que los nodos de propagación no la necesitan y por lo tanto no la incluyen. Los nodos actuadores se componen de un nodo de propagación conectado a una computadora embebida en una tarjeta (SBC por sus siglas en Inglés *Single Board Computer*). La SBC es una computadora con el sistema operativo GNU/Linux, y permite controlar el actuador que tiene conectado mientras que el nodo de propagación permite la comunicación con el resto de la red.

Para la programación de aplicaciones, los autores construyeron un *middleware* basado en servicios que permitió crear una abstracción de la red adecuada para su uso. Este *middleware* proporciona servicios de red, enrutamiento, sincronización y de administración de la energía. El *middleware* se encarga de los detalles de bajo nivel, de tal manera que los programadores de aplicaciones no requieren conocer estos detalles para crear las aplicaciones.

III.5. Problemática

Los primeros trabajos de investigación sobre redes inalámbricas de sensores proponían redes similares a las ya conocidas redes ad hoc, pero con las diferencias clave presentadas en la Sección II.3. Debido a estas diferencias, la investigación en el área se dedicó a diseñar

nuevos esquemas de comunicación, algoritmos y lenguajes de programación para trabajar con este nuevo tipo de redes. Con el tiempo, el *hardware* disponible fue evolucionando, produciendo nodos más pequeños y con mayores capacidades que sus antecesores. Una vez que se contó con una infraestructura de *hardware* (nodos sensores, nodos pasarela, comunicación inalámbrica) y *software* (algoritmos de enrutamiento, sistemas operativos para los nodos, lenguajes de programación) que se consideró adecuada, se comenzaron a diseñar sistemas *middleware* con el objetivo de facilitar el desarrollo de aplicaciones para WSN y llevar esta tecnología al público en general.

En (Römer y Mattern, 2004) los autores describen que "programar sensores involucra el uso de lenguajes de programación de bajo nivel para comunicar las instrucciones al *hardware*. Por lo tanto existe una fuerte necesidad de abstracciones de programación que simplifiquen el asignar tareas a los sensores y de *middleware* que soporte tales abstracciones". Estos lenguajes de «bajo nivel» a los que se refieren son, por ejemplo, nesC o el lenguaje ensamblador del procesador de los nodos.

Para desarrollar aplicaciones con toda la funcionalidad esperada de una WSAN tomando en cuenta las restricciones inherentes, aún es necesario ser experto en el área, desarrollando con herramientas especializadas para redes WSAN (por ejemplo con TinyOS y nesC). Dicho de otra manera, existe una brecha entre la funcionalidad *potencial* del *hardware* y la funcionalidad *lograda* por el *middleware* existente.

Actualmente gran parte del *hardware* disponible para nodos sensores soporta el sistema TinyOS. Esto es una ventaja para los desarrolladores, ya que cuentan con una plataforma de *software* estándar que pueden modificar de acuerdo a sus necesidades. Por el lado de los actuadores, no existe al conocimiento del autor una estandarización en el manejo de los mismos. Esto representa un reto para los programadores, ya que la integración de actuadores a la red requiere de la programación de manejadores de bajo nivel para estos

dispositivos.

Además de no existir una forma eficiente y aceptada para controlar los actuadores en una red WSAN, se dificulta la integración de nuevo *hardware* a las soluciones actuales. Debido al estado actual de las herramientas de desarrollo, es muy difícil comenzar a desarrollar una aplicación sin una selección previa de una plataforma de *hardware* y *software* específica (dependencia de la plataforma).

III.6. Conclusión

Las redes WSAN son más que una simple extensión de las redes WSN, y por lo tanto se deben estudiar tomando en cuenta sus capacidades, limitantes y usos únicos. Por su capacidad de actuar en el ambiente que les rodea son especialmente útiles para aplicaciones de control distribuido. Debido a que los cambios que provoca un actuador los pueden medir los sensores, se genera un ciclo de retroalimentación que lo aprovechan las aplicaciones de control.

Existen básicamente dos tipos de redes WSAN: automáticas y semiautomáticas. En las primeras no existe una entidad de control explícita, siendo los nodos actores los que ejecutan el control y toman las decisiones sobre las acciones a ejecutar. Las semiautomáticas cuentan con una unidad de control centralizada, típicamente una computadora conectada al nodo pasarela. La ventaja de las automáticas es que se ahorra mucho ancho de banda y energía al reducir el número de transmisiones necesarias, sin embargo las semiautomáticas, al ser más parecidas a las redes WSN, son más fáciles de programar y las soluciones actuales se pueden utilizar o adaptar con mayor facilidad.

Las herramientas disponibles para desarrollar aplicaciones para WSAN cubren algunas de las necesidades de los desarrolladores. Sin embargo, para explotar todo el potencial

de las redes WSAN aún es necesaria la programación de bajo nivel. No existen sistemas que proporcionen una abstracción adecuada, permitiendo explotar toda la funcionalidad de las redes WSAN. Esto se hace evidente al observar que nuevos proyectos comerciales y de investigación con redes WSAN deciden desarrollar sus propios sistemas *middleware* en vez de utilizar uno existente, programando los sensores y los actuadores a bajo nivel. En el Capítulo IV se presenta la arquitectura de un *middleware* para la creación de aplicaciones que utilizan fuentes de datos heterogéneas de manera rápida y flexible, con énfasis en aplicaciones de control distribuído con redes WSAN.

Capítulo IV

Arquitectura Propuesta

La arquitectura de un sistema de software refleja las decisiones tomadas acerca de su diseño. Estas características de diseño son las más difíciles de cambiar una vez puestas en práctica (es decir, una vez programado el sistema) y a su vez son las más críticas para el éxito o fracaso de un sistema de software. La arquitectura habla sobre los componentes principales que forman el sistema, y no sobre cada una de las partes (Albin, 2003).

A la fecha gran parte de la investigación sobre redes WSAN se ha centrado en las capas de bajo nivel. Los temas con investigación más intensa han sido algoritmos de enrutamiento, agregación de datos y manejo eficiente de energía dentro de la red de sensores. Problemas como el emplazamiento, el desarrollo de aplicaciones, la estandarización y la reutilización (Tavakoli *et al.*, 2007) son aspectos que no habían recibido tanta atención (Aberer *et al.*, 2006, Younis y Akkaya, 2008). En este trabajo el objetivo principal es el desarrollo de una arquitectura que facilite la programación de aplicaciones para redes WSAN, permitiendo el soporte dinámico a *hardware* heterogéneo.

En este capítulo se presentan los requerimientos de una arquitectura para la creación de aplicaciones que utilicen una WSAN, detallando los componentes que la forman. En el Capítulo V se presenta la implementación de un prototipo, creado para probar la factibilidad de la creación de un middleware con la arquitectura propuesta, así como la utilidad del mismo.

IV.1. Requerimientos de la arquitectura

En el Capítulo III se presentaron las características de las aplicaciones con redes WSAN. Los requerimientos de la arquitectura son una consecuencia directa de estas características. La naturaleza heterogénea del *hardware* que se debe soportar en una red WSAN crea una dificultad en la integración de este *hardware* a los sistemas actuales. También, el desarrollo utilizando herramientas de bajo nivel crea una dependencia con la plataforma en la que se programaron los sistemas.

La principal aplicación que se le da a las redes WSAN es la de crear sistemas de automatización y control distribuído. Por tal motivo, la arquitectura debe permitir la obtención de datos del medio ambiente, la ejecución de algoritmos de control sobre los datos obtenidos y finalmente la ejecución de las acciones de control correspondientes. De acuerdo con lo que argumentan los autores en (Prinsloo *et al.*, 2006), en un ambiente limitado los nodos generalmente son del mismo tipo y modelo, conteniendo los mismos sensores y actuadores. Sin embargo, en aplicaciones prácticas este escenario es muy improbable, haciendo el uso y mantenimiento de estos tipos de nodos distintos una limitante mayor. Para superar estas limitantes, la arquitectura debe permitir al desarrollador la programación de sistemas de manera transparente, es decir, proporcionar una API que permita el manejo de los distintos dispositivos, sin importar el tipo y modelo del *hardware*.

Las redes WSAN son de naturaleza dinámica, su topología puede cambiar aún cuando los nodos sean estáticos debido a estrategias de ahorro de energía (en cualquier momento un nodo puede ser apagado para conservar energía). Por esta naturaleza dinámica, es importante que la arquitectura permita el agregar nuevos tipos de nodos o modificar los existentes, mientras que la red se encuentra en funcionamiento. Actualmente los sistemas con redes WSAN son muy difíciles de extender para que cubran otro tipo de necesidades no contempladas inicialmente. En ausencia de un sistema de administración de nodos adecuado, la capa de acceso a los nodos y a su información generalmente se programa desde

cero, resultando en un sistema que no es adecuado para su reutilización. Esta problemática la debe resolver la arquitectura propuesta.

Está claro que una arquitectura adecuada para la creación de aplicaciones con redes WSAN debe:

- Permitir la integración de *hardware* heterogéneo, mientras que la red se encuentra en funcionamiento.
- Eliminar la necesidad de codificación de las capas de bajo nivel.
- Permitir y facilitar la obtención de datos y la ejecución de acciones de control.
- Proporcionar una API consistente que permita el manejo de todos los dispositivos que integran la red.
- Eliminar la dependencia de la plataforma de implementación.

Además, la arquitectura debe soportar y promover el crecimiento de la red a gran escala. Actualmente las redes WSAN son pequeñas, aisladas y de propósito específico, en parte debido a la dificultad de integrar los sistemas creados con redes WSAN a otros sistemas ya establecidos, como por ejemplo aplicaciones Web. Una arquitectura para redes WSAN debe permitir la publicación de los resultados obtenidos, y el procesamiento de estos resultados de manera sencilla, promoviendo el intercambio de datos entre sistemas (Dickerson *et al.*, 2008).

Por último, la arquitectura debe procurar ser «a prueba del futuro» (*futureproof*). Lo anterior quiere decir que debido a la intensa investigación y lo reciente del área de redes de sensores las soluciones propuestas tienden a evolucionar, haciéndolas incompatibles unas con otras o desechando las que se prueba que no solucionan el problema. Un ejemplo típico de esto son los *middleware* que abstraen la red como base de datos (TinyDB[1]) (Madden

[1] http://telegraph.cs.berkeley.edu/tinydb/

et al., 2005) y las «fuentes de flujo de datos» (*stream feeds*) (Dickerson *et al.*, 2008). TinyDB permite ejecutar sentencias al estilo del lenguaje SQL en vez de tener que escribir código de bajo nivel en lenguaje nesC, es decir, provee una vista de base de datos a la red de sensores. A diferencia de esto, los *stream feeds* permiten que los flujos de datos provenientes de la red de sensores sean «ciudadanos de primera clase» de la Web, proporcionándoles un localizador uniforme de recursos (URL por las siglas en Inglés de *Uniform Resource Locator*) con la posibilidad de crear hipervínculos entre estos flujos de datos, guardarlos en los favoritos, y cualquier otra acción que se pueda hacer con los datos de la Web actual. La arquitectura propuesta debe permitir la implementación de estos métodos de obtención de datos, implementarlos lado a lado, ejecutar acciones de control sin importar cómo se obtuvieron estos datos y mezclar implementaciones, de tal manera que el desarrollador pueda evaluar el impacto de un método u otro en su aplicación específica y, posiblemente, intercambiar un método por otro.

IV.2. Trabajo previo

Algunos trabajos anteriores se han enfocado en resolver algunos de los problemas expuestos en la Sección III.5. Estos trabajos se centran en los requerimientos de las aplicaciones de control distribuido, estableciendo requerimientos y definiendo la estructura de las aplicaciones y del *hardware* que compone a la red.

IV.2.1. Deshpande *et al.* (2005)

En este trabajo, los autores dividen su sistema en 4 componentes:

- Inferencia: responsable de tomar las decisiones sobre qué sensores observar, de tal manera que se pueda inferir qué lecturas son las más valiosas y qué actuadores son los más adecuados para las necesidades del usuario.

- Control: encapsula la relación entre las lecturas de los sensores, las preferencias del usuario y el estado de los actuadores.
- Monitoreo: responsable de determinar el estado de cada uno de los nodos que forman la red.
- Abstracción: es una capa con la responsabilidad de recolectar las preferencias del usuario y los valores sensados, así como de diseminar el estado de los actuadores mientras que administra los ciclos de actividad de cada nodo.

De acuerdo con las leyes de control programadas en el sistema, se infiere cuáles son los sensores que se deben consultar y cuáles son los actuadores que se deben accionar para lograr mantener las preferencias establecidas por el usuario. El sistema de control especifica algunas reglas basadas en la teoría de control para llevar a cabo las acciones correctivas que hagan cumplir estas preferencias (Deshpande *et al.*, 2005).

IV.2.2. Xia *et al.* (2007)

Los autores proponen un diseño de aplicaciones que no modifica las capas bajas de la red, no utiliza información estadística sobre la distribución de la pérdida de paquetes y tampoco sobre algún modelo de control específico de la aplicación. La idea es obtener un diseño de aplicación que se pueda aplicar a distintos escenarios, y no a aplicaciones específicas. Para lograr el control, al detectarse una pérdida de datos el actuador predice la salida esperada utilizando salidas anteriores y un algoritmo de control.

La principal contribución de este trabajo es la definición de dos arquitecturas de las redes WSAN: automáticas y semiautomáticas (Xia *et al.*, 2007).

IV.2.3. Sgroi *et al.* (2005)

En este trabajo se identifican dos necesidades críticas de las redes WSAN: la integración incremental de nodos heterogéneos y el desarrollo de aplicaciones de una manera que sea en gran medida o completamente independiente de la plataforma de implementación.

Para lograrlo, definen una serie de servicios que consideran básicos sobre los cuales se pueden construir otros servicios más complejos o particulares para una red WSAN específica. Estos servicios básicos son:

- Servicio de peticiones: utilizado para obtener información de otros componentes.
- Servicio de comando: utilizado para definir el estado de otros componentes.
- Servicio de sincronización: utilizado para definir una base de tiempo en común.
- Servicio de localización: utilizado para que los componentes puedan descubrir su propia localización.
- Servicio de repositorio de conceptos: utilizado para mantener una base de datos de las capacidades de la red emplazada.

Los autores introducen conceptos importantes para lograr la independencia de la plataforma de implementación: el «sensor virtual» como unidad funcional y la división del software para WSAN en sistemas dedicados que interactuan entre ellos (Sgroi *et al.*, 2005).

IV.2.4. Aberer *et al.* (2006)

En este trabajo se define la arquitectura e implementación de un *middleware* desarrollado con el objetivo de resolver los problemas existentes de emplazamiento, desarrollo de aplicaciones y estandarización. Implementa algunas de las ideas plasmadas en (Sgroi *et al.*, 2005), específicamente el concepto de «sensor virtual» y la división del sistema completo

en servicios.

El *middleware* (llamado GSN por las siglas en Inglés *Global Sensor Network*) utiliza un lenguaje basado en XML para especificar los sensores virtuales, los cuales pueden ser un nodo físico o una combinación de nodos. La implementación es independiente de la plataforma y tiene la capacidad de agregar nuevo hardware en tiempo de ejecución (Aberer *et al.*, 2006).

IV.2.5. Avilés-López *et al.* (2007)

Los autores presentan una arquitectura orientada a servicios Web, creando un *middleware* para aplicaciones de monitoreo. El *middleware* proporciona una serie de servicios Web por medio de los cuales el cliente puede consultar los datos obtenidos de la red, programar eventos y alertas. La arquitectura divide el sistema completo en 4 partes (como se muestra en la Figura 7): el componente de Monitoreo que se encuentra en los nodos, el componente Pasarela encargado de la comunicación entre los nodos y la estación base, el componente de Registro que consiste en una base de datos para almacenar los datos recolectados y el componente Servidor que provee los servicios Web al mundo exterior.

Gracias al uso de servicios Web, el desarrollo de aplicaciones que utilicen la red WSN no se encuentra restringido a un lenguaje o arquitectura específica (Avilés-López y García-Macías, 2007).

IV.3. Diseño de la arquitectura

Con los requerimientos planteados en la Sección IV.1 y el estudio del trabajo previo, se diseñó una arquitectura que apoye en el cumplimiento de los objetivos de este trabajo. A continuación se detallan los 3 componentes principales de esta arquitectura: el modelo de

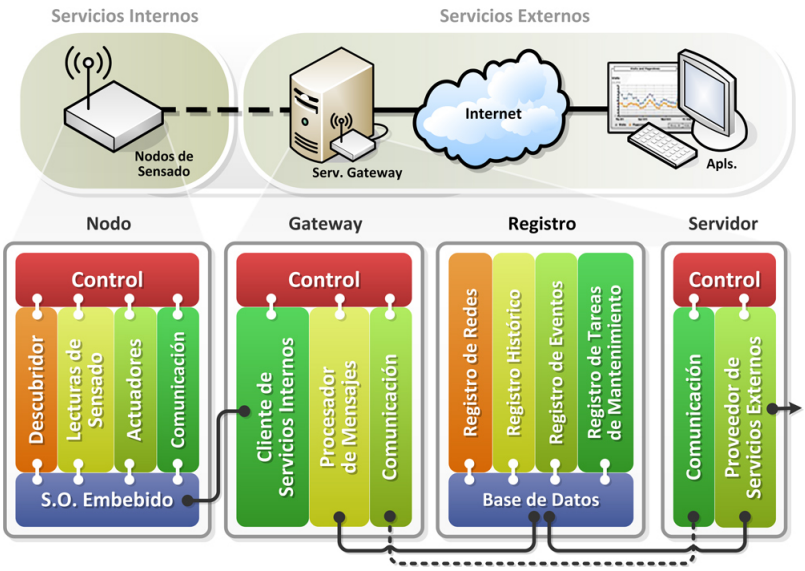


Figura 7: **Componentes de la arquitectura del *middleware* «TinySOA»**

comunicación, los servicios ofrecidos por el sistema y las cápsulas de soporte.

IV.3.1. Modelo de comunicación

El modelo de comunicación se refiere a la manera en la que la arquitectura organiza las entradas y salidas de datos, y cómo permite que estos datos se puedan consultar, procesar y almacenar. También, el modelo de comunicación tiene que ver con la manera en la que el desarrollador puede «conectar» su propia aplicación para que ésta utilice los servicios ofrecidos por la red, y de qué manera se lleve a cabo la comunicación entre los datos recolectados, la aplicación del desarrollador y las acciones de control a ejecutar.

En este contexto, la aplicación del desarrollador tiene el rol de «controlador». El controlador es el encargado de generar las acciones de control correspondientes. Las acciones de control las genera un algoritmo de control, de los cuales existen algunas propuestas siendo el proporcional integral derivativo (PID por las siglas en Inglés de *Proportional, Integral,*

Derivative) uno de los más populares. Sin embargo, el controlador puede ser cualquier programa, no necesariamente una implementación de un algoritmo de control.

El controlador es notificado de todos los eventos generados por la red. Estos eventos pueden ser datos provenientes de los sensores, o simplemente una confirmación de ejecución de una acción, por nombrar un ejemplo. Una vez recibidos los datos, el controlador determina las acciones de control y utilizando el servicio de acceso a nodos, manda ejecutar las acciones de control, cerrando el lazo de control. Lo anterior se ilustra en la Figura 8.

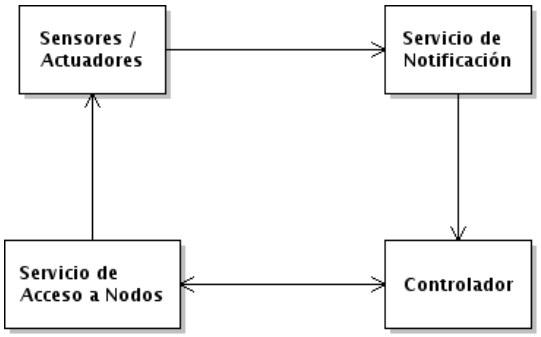


Figura 8: **Modelo de comunicación entre nodos y el controlador**

La Figura 9 muestra el diagrama UML de clases para el patrón de comportamiento llamado «Observador» (en Inglés *Observer* o *Publish / Subscribe*). Este patrón permite la comunicación entre objetos distintos. Gracias a lo anterior, es posible agregar objetos a un programa mientras éste se encuentra en funcionamiento (en tiempo de ejecución) lo que permite construir *software* altamente flexible, extensible y reutilizable. El patrón observador define a un sujeto (aquel que publica datos) y potencialmente muchos observadores (los que se suscriben a los datos). Los observadores se registran ante el sujeto, y este notifica a los observadores cuando ocurre el evento de interés (Gamma *et al.*, 1994).

El patrón observador se utiliza como la base del modelo de comunicación de la arquitectura que aquí se propone. Cada fuente de datos (sensor, actor o cualquier nodo capaz de generar datos o eventos) es un sujeto. Estos sujetos pueden tener registrados cualquier

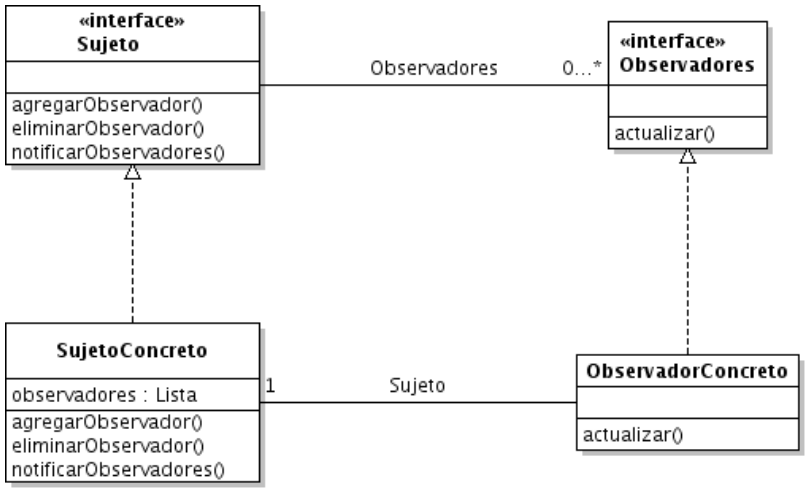


Figura 9: **Diagrama de clases para el patrón *Observer*, también conocido como *Publish / Subscribe***

número de observadores (una relación de uno a muchos). Los observadores son actualizados automáticamente con los datos generados por los sujetos.

En el modelo de comunicación, los observadores reciben los mensajes generados por el sujeto de su interés. Esto es análogo a un sistema de subscripción, como por ejemplo el periódico (y de ahí el nombre *Publish / Subscribe*) en donde los subscriptores reciben las noticias que el editor publica en el periódico, en este caso observadores y sujetos respectivamente. Los observadores tienen como propósito hacer disponible la información obtenida del sistema al mundo exterior. Un observador se registra ante el servicio de notificación para indicarle de cuál sujeto desea obtener mensajes, después el servicio le notifica automáticamente cuando se genere un mensaje proveniente de su sujeto de interés. Una vez que recibe un mensaje, el observador procesa el mensaje de acuerdo a su función, haciendo disponible la información obtenida a otros sistemas por distintos medios. Este proceso se puede visualizar en la Figura 10.

Algunas funciones típicas de observadores pueden ser:

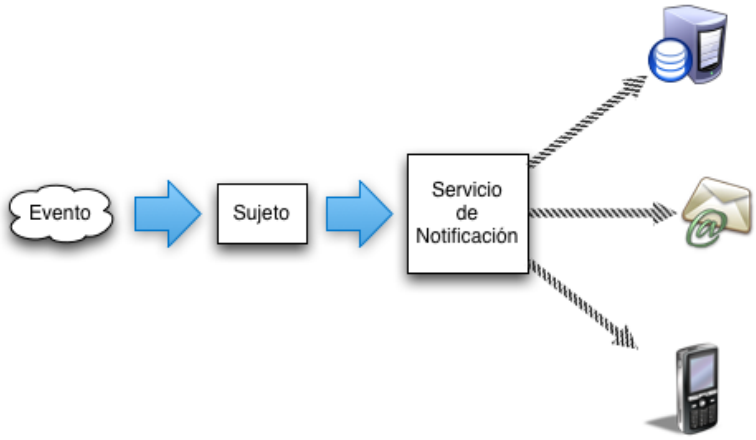


Figura 10: **Flujo de la información generada en la red. Al recibirla, los observadores la hacen disponible a otros sistemas, promoviendo la interoperabilidad y el intercambio de información.**

- Base de datos: los mensajes se guardan en una base de datos para su consulta posterior con consultas SQL.

- Fuente de flujo de datos: la información obtenida se integra a la Web utilizando URLs para su identificación, filtrado y agregación. La implementación completa de un sistema como éste puede o no residir en el observador. En caso de que no se encuentre en el observador, este sólo haría las veces de mediador (*proxy*) entre el sujeto y el sistema completo de fuente de flujo de datos.

- Correo electrónico: se envía un correo electrónico a algunas direcciones especificadas cuando la red detecta un evento, por ejemplo el accionar un relevador o la lectura de un nivel de temperatura crítico.

- Publicación de contenidos: los mensajes se publican a un medio configurado, como puede ser un archivo de texto, una página HTML o un documento XML.

- Envío de mensajes cortos por la red celular. El servicio de mensajes cortos (SMS por las siglas en Inglés de *Short Message Service*) es uno de los servicios más populares de la telefonía celular. Los mensajes se envían a un número de celular configurado por medio de este servicio.

Este modelo de comunicación promueve por diseño el intercambio de datos entre sistemas, la publicación de los resultados obtenidos y el procesamiento de resultados de manera sencilla.

IV.3.2. Servicios

En algunos trabajos de investigación ya se ha propuesto la idea de dividir la arquitectura en servicios (Sgroi *et al.*, 2005, Deshpande *et al.*, 2005). También se han creado sistemas *middleware* implementando una arquitectura de servicios como la propuesta en estos trabajos (Aberer *et al.*, 2006). La orientación a servicios es un paradigma de diseño, el cual especifica que la lógica de automatización se crea en servicios, obteniendo una clara separación de responsabilidades (Allen *et al.*, 2004). Un servicio es un mecanismo para habilitar el acceso a una o más características, donde el acceso se proporciona utilizando una interfaz prescrita y se ejecuta de manera consistente a las políticas y restricciones especificadas por la descripción del servicio (OASIS, 2008). Es importante destacar las diferencias entre arquitectura orientada a servicios y «servicios Web», este último es una implementación del paradigma de diseño orientado a servicios, utilizando protocolos y sistemas Web.

La arquitectura propuesta se divide en servicios, proporcionando un punto único de acceso a la funcionalidad del sistema. Esto es una característica deseable: el mantenimiento del sistema se hace más sencillo al tener entidades bien separadas responsables de cierta funcionalidad, así como también los usuarios del sistema ven simplificada la tarea de aprender a crear sus aplicaciones gracias a la clara división de responsabilidades y la API (por las siglas en Inglés de *Application Programming Interface*) resultante de esto último.

Los servicios ofrecidos por la arquitectura son:

- Servicio de acceso al *hardware*: mantiene un registro del *hardware* disponible y de las acciones que este *hardware* ofrece a las aplicaciones. Existe una distinción entre los nodos individuales de la red y el *hardware* disponible; mientras que una típica

red WSN tiene entre cientos o miles de nodos, el *hardware* disponible es solamente de un tipo, dispositivos de sensado. El servicio de acceso al *hardware* es el responsable de cargar de manera dinámica (es decir, mientras la red se encuentra en funcionamiento) el soporte para los distintos tipos de dispositivos que puedan existir en la red WSAN, y provee los medios para utilizar estos dispositivos.

- Servicio de notificación: encargado del envío de los mensajes provenientes de los sujetos, a los observadores registrados. El servicio de notificación mantiene el estado del registro de los observadores y los notifica, enviándoles los datos generados, cuando un sujeto de interés los genera. Utilizando este servicio, los observadores pueden administrar su subscripción a los distintos sujetos. También, el servicio de notificación envía todos los mensajes generados al controlador.

- Servicio de procesamiento de peticiones: recibe peticiones externas al sistema, las procesa y las ejecuta. Una petición externa se entiende como una instrucción de realizar alguna de las acciones puestas a disposición por el servicio de acceso a *hardware*, generada desde fuera del sistema (es decir, no por el controlador). El servicio de procesamiento de peticiones recibe la petición, verifica su validez de acuerdo a varios criterios (seguridad, sintaxis, si se puede completar en ese momento, entre otros) y finalmente la ejecuta, apoyado en los otros servicios.

- Servicio de mantenimiento de red: se encarga de persistir el estado del modelo de comunicación (subscripciones, módulos de *hardware* en uso) entre arranques del sistema, así como también de proporcionar información estadística sobre el uso de la red.

- Servicio de acceso al sistema de archivos: proporciona información sobre eventos del sistema de archivos local. El servicio de acceso al sistema de archivos provee los medios para notificar a los otros servicios sobre la presencia de nuevas cápsulas (ver Sección IV.3.3).

Los servicios que interactúan con el *hardware* de la red, lo hacen por medio de módulos autocontenidos de soporte llamados «cápsulas».

IV.3.3. Cápsulas de soporte: modularización de la funcionalidad

En la Sección IV.1 se estable que uno de los requerimientos de la arquitectura es el de ser extensible y «a prueba del futuro», permitiendo el intercambio de implementaciones y el aumento de las capacidades de la red en tiempo de ejecución. Lo que se desea lograr es tener una arquitectura que permita utilizar, en lo posible, el sistema en maneras no previstas, creando una plataforma sobre la cual otros desarrolladores puedan probar sus propias implementaciones, facilitando el trabajo de integración con tecnologías existentes que necesiten utilizar.

Para lograr una arquitectura con las características anteriores, el autor propone el concepto de «cápsulas» basado en la modularidad y extensibilidad que presentan otros programas (como el navegador «Firefox» o el editor de texto «JEdit») por medio de *plugins*, así como también en el concepto del «sensor virtual» propuesto originalmente en (Sgroi *et al.*, 2005). Una cápsula es un módulo de soporte que puede ser agregado al sistema mientras este se encuentra ejecutándose. Las cápsulas contienen la implementación del soporte que proporcionan, archivos de configuración y cualquier otro recurso necesario para su funcionamiento. Existen 2 tipos de cápsulas:

- *Sujeto*: también llamada *publisher*, proveen de nuevas acciones al sistema y son fuente de datos. Al agregar una cápsula sujeto, esta incrementa las capacidades del sistema.
- *Observador*: se registran para escuchar mensajes de un sujeto específico, notificando al exterior los resultados de las acciones ejecutadas.

Una acción es un método ofrecido por una cápsula sujeto, que se puede ejecutar ya sea por iniciativa del controlador o por una petición externa. Una cápsula que controla a

una red WSN podría tener acciones como obtener el valor de una variable monitoreada u obtener los niveles de batería restante, mientras que una cápsula de control para un actuador podría ser tan simple que solamente proporcione las acciones activar actuador y desactivar actuador. La relación entre acciones, cápsulas sujeto y observadores se ilustra en la Figura 11. La Figura 12 muestra el diagrama general de la arquitectura propuesta; el servicio de acceso al *hardware* se comunica con las cápsulas sujeto y provee una interfaz común para hacer uso de los dispositivos que forman a la red WSAN por medio de un controlador. Las cápsulas sujeto utilizan el servicio de notificaciones para publicar los datos generados por la red, mismos que se reciben y procesan por las cápsulas observador registradas a la cápsula sujeto que dio origen a los datos.

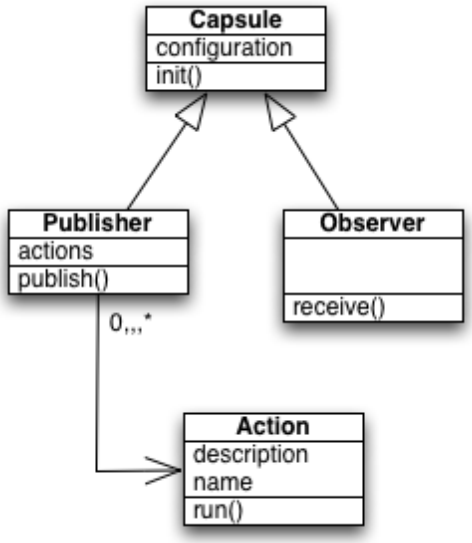


Figura 11: **Diagrama de clases para las cápsulas de soporte. Los sujetos y observadores heredan de la clase padre** «***Capsule***»

IV.4. Programación de aplicaciones

Uno de los objetivos de la arquitectura es el de permitir la creación de aplicaciones que utilicen una red WSAN. Gracias a que el desarrollador no se tiene que preocupar por

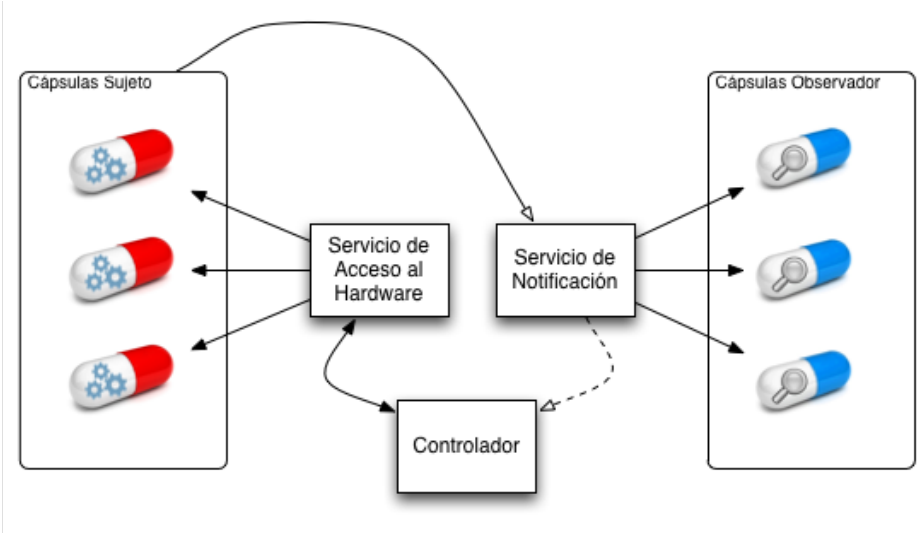


Figura 12: **Diagrama general de la arquitectura propuesta. Muestra la interacción entre los servicios, el controlador y las cápsulas. La línea punteada muestra interacción opcional.**

programar el soporte para el *hardware* específico de su red, esta tarea se simplifica grandemente.

El desarrollador se enfoca a la tarea de programar el controlador. El controlador es lo que hace a cada sistema único y que varía entre implementaciones. Es en el controlador donde se generan las acciones de control que se llevan a cabo, así como también es el que ejecuta dichas acciones de control apoyado en el servicio de acceso a *hardware* y el de notificación.

La arquitectura no impone restricciones sobre el controlador, permitiendo gran libertad al desarrollador sobre su estructura. Cuando un sujeto publica datos, estos llegan al controlador automáticamente (además de enviarse a los observadores correspondientes) y es él quien decide las acciones de control a llevar a cabo. El controlador también es el responsable de inicializar el sistema.

Una aplicación completa se compone del código del controlador y programas *script*

(programas más pequeños interpretados) que ejecutan los sujetos y los observadores. Los sujetos se pueden programar con programas *script* para llevar a cabo acciones repetitivas, por ejemplo, consultar un sensor cada minuto, eliminando la interacción directa del controlador en esta acción. Los observadores pueden aplicar filtros al contenido que reciben utilizando estos mismos programas *script*, por ejemplo solo enviar por correo aquellos valores por encima de los 90°C.

IV.5. Conclusión

En este capítulo se propone una arquitectura para un sistema *middleware*, dirigido a aplicaciones de control distribuído utilizando redes WSAN. La arquitectura se enfoca en la integración de *hardware* heterogéneo, la independencia de la plataforma, el intercambio de información con otros sistemas y la facilidad de programación de aplicaciones. Estos son los principales requerimientos de los sistemas *middleware* para redes WSAN.

La arquitectura se divide en servicios, similar a lo presentado en los trabajos de (Sgroi *et al.*, 2005, Deshpande *et al.*, 2005, Aberer *et al.*, 2006, Avilés-López y García-Macías, 2007). Es posible extender el *hardware* soportado por el sistema en tiempo de ejecución gracias al uso de cápsulas de soporte. Las cápsulas pueden ser de dos tipos: sujeto y observador, proporcionando nuevas acciones al sistema o notificando al exterior los resultados de las acciones ejecutadas, respectivamente. El modelo de comunicación y las cápsulas forman una arquitectura flexible, extensible y reutilizable.

Para programar aplicaciones con la arquitectura propuesta es necesario crear un controlador. El controlador se registra ante el sistema para recibir los mensajes generados, aplica un algoritmo de control y ejecuta acciones de control resultantes de los datos obtenidos de la red apoyado de los servicios. También es posible ordenar acciones que se ejecutarán una sola vez, gracias al servicio de procesamiento de peticiones. Lo anterior es útil para

generar acciones de control de manera sencilla.

Por último, en el Capítulo V se detalla la implementación de un *middleware* utilizando la arquitectura descrita en el presente capítulo.

Capítulo V

Implementación de un Prototipo

Con el objetivo de probar la arquitectura propuesta en el Capítulo IV, se llevó a cabo la implementación de un sistema *middleware* prototipo basado en dicha arquitectura. El desarrollo del *middleware*, denominado «ActiveCloud», permitió identificar errores no previstos en el diseño de la arquitectura, así como también comprobó la viabilidad de construir un sistema con la arquitectura propuesta. En el presente capítulo se describe la implementación de este *middleware*.

V.1. Plataforma de *hardware*

El objetivo principal del presente trabajo es el soporte dinámico de *hardware* heterogéneo. Inicialmente se contó con una plataforma de *hardware* a la cual se le dio soporte por medio del desarrollo de cápsulas sujeto. Esta plataforma incluye nodos MICAz y eKo de Crossbow[1], así como también phidgets[2].

Figura 13: **Nodo MICAz**

V.1.1. Nodos MICAz y TinySOA

El soporte para los motes MICAz (ver Figura 13) se desarrolló al crear una cápsula sujeto para el *middleware* TinySOA[3]. MICAz es un mote para redes WSN que incluye un transmisor y receptor de radio frecuencia compatible con el estándar IEEE 802.15.4/ZigBee. Estos nodos son compatibles con TinyOS 1.1.7 ó superior y se pueden configurar con distintas placas de sensado. TinySOA incluye el soporte para la placa de sensado MTS310CA (ver figura 14) (Avilés-López y García-Macías, 2007).

La comunicación no se hace directamente con el *hardware*, en vez de esto, se utilizan los servicios Web proporcionados por TinySOA para hacer las consultas a los nodos. TinySOA incluye un componente pasarela, el cual se encarga de la comunicación por medio del protocolo TCP/IP, gracias al programa *serialforwarder* que se incluye como parte de la distribución de TinyOS.

[1]Crossbow Technology, Inc. http://www.xbow.com/
[2]Phidgets, Inc. http://www.phidgets.com/
[3]http://www.tinysoa.net/

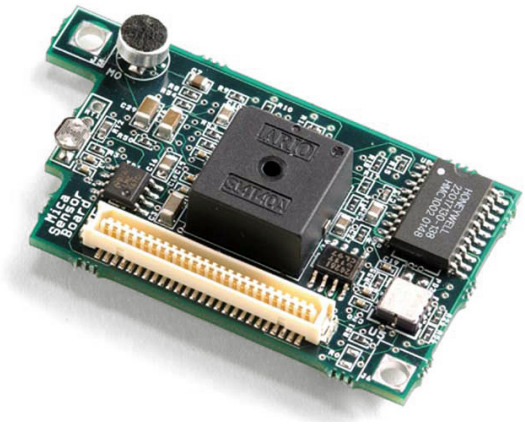


Figura 14: **Placa de sensado MTS310CA, la cual es soportada por el *middleware* TinySOA**

V.1.2. eKo PRO Series

El kit de desarrollo eKo PRO Series incluye los nodos eKo eN2100. Estos nodos utilizan un protocolo ad-hoc multisalto llamado XMesh desarrollado por Crossbow. Las características físicas de la red WSN que se forma con los nodos eKo son las mismas que una red WSN tradicional. Sin embargo, los nodos eKo eN2100 se diseñan para utilizarse a la interperie (ver Figura 15). Estos nodos incluyen la familia de procesadores y transmisores de radio IRIS, los cuales se alimentan con baterías recargables y páneles solares integrados en el mismo nodo. El rango del radio en exteriores es entre 150 y 450 metros (500 y 1500 pies) dependiendo de las condiciones del área de emplazamiento (presencia de obstáculos, línea de vista entre los nodos, entre otros factores). Pueden operar a temperaturas entre -40^{o}C y 60^{o}C.

Estos nodos contienen 4 puertos en la parte inferior, dedicados a conectar 4 sensores compatibles. Los sensores se conectan al nodo y al reiniciar, éste identifica los nuevos sensores instalados. El kit PRO Series viene con sensores de humedad y temperatura para tierra y aire. Para la comunicación utiliza una estación base, responsable de comunicar a sistemas exteriores con la red WSN formada por los nodos. La estación base ejecuta el sistema operativo Debian GNU/Linux[4] y ofrece una interfaz via Web para la consulta de los datos.

[4]Debian GNU/Linux. http://www.debian.org/

Figura 15: **Nodo eKo eN2100. Estos nodos cuentan con una cubierta que les permite trabajar a la interperie, además de baterías recargables y páneles solares como fuente de energía.**

La comunicación con estos nodos se hace a través de la estación base, utilizando el protocolo *Secure Shell* (SSH), el cual provee una conexión segura a través de redes inseguras (como Internet) gracias al uso de encriptación (Crossbow, 2008).

V.1.3. Phidgets

Los phidgets (ver Figura 16) son dispositivos de sensado y control que funcionan por medio del bus serie universal (USB por las siglas en Inglés de *Universal Serial Bus*). La comunicación USB con la computadora se hace por medio de su API, misma que se puede utilizar desde una variedad de lenguajes de programación entre los que se incluyen .NET, LabView, Java, Delphi, C, C++ y Python, así como también una interfaz de servicios Web que permite controlar los phidgets via Web. El propósito de dar soporte a estos dispositivos fue contar con actuadores de prueba de bajo costo.

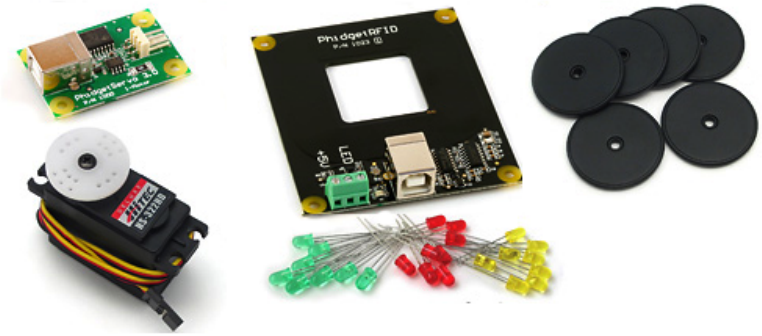


Figura 16: **Varios phidgets entre los que se encuentran lectores RFID y controladora de servomotor**

El dispositivo de actuado que se incluye es un servomotor con una exactitud de pasos de 0.1 grados. Este servomotor se controla por medio del puerto USB, mismo que le provee la energía necesaria. Existen también otros dispositivos phidget como:

- Dispositivos de salida (LEDs y pantallas LCD).
- Servomotores, motores a pasos y motores de corriente directa (DC).
- Lectoras de identificación por radio frecuencia (RFID por las siglas en Inglés de *Radio Frequency Identification*).
- Sensores de varios tipos: temperatura, fuerza, acelerómetro, potencial hidrógeno (pH), entre otros.

La razón por la que se eligió trabajar con phidgets es su facilidad de uso y de integración a proyectos existentes, además de su compatibilidad con los principales sistemas operativos (GNU/Linux, Mac OS X y Windows) y con una gran variedad de lenguajes de programación. Su bajo costo y capacidad de ser controlados desde cualquier computadora con puertos USB disponibles los hacen muy accesibles como dispositivos de prueba.

V.2. Implementación

En esta sección se presentan los detalles de la implementación del *middleware* prototipo «ActiveCloud». Este *middleware* se basa en la arquitectura propuesta en el Capítulo IV. La implementación se hizo en el lenguaje Java debido al conocimiento del lenguaje por parte del autor, así como también a la disponibilidad de código fuente en dicho lenguaje del proyecto TinySOA, mismo que fue utilizado al inicio de la implementación para motivos de pruebas. El *middleware* se encarga del manejo del *hardware* de la red, así como de la comunicación de los datos generados, la programación de aplicaciones y la ejecución de acciones de control en el sistema físico. Lo anterior se muestra en la Figura 17.

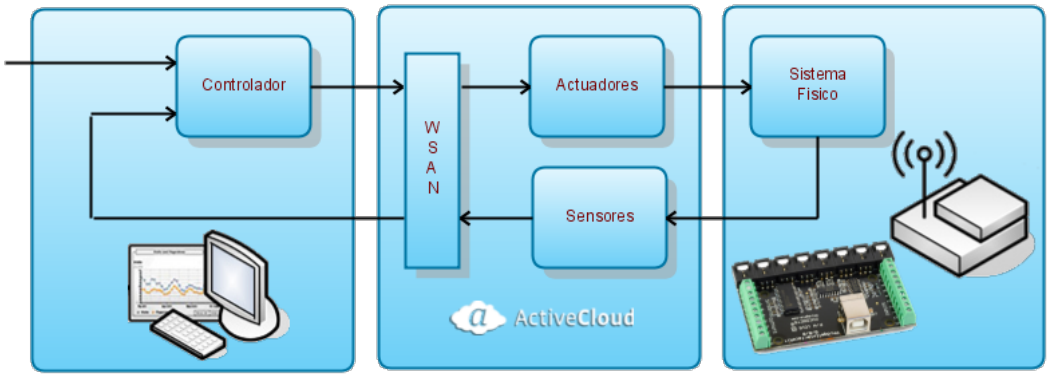


Figura 17: **Los componentes que encapsula ActiveCloud son el acceso al *hardware* de la red (mostrado en el bloque del centro) y la ejecución de acciones de control mediante el sistema físico. El controlador lo proporciona el desarrollador**

Tanto el controlador como los servicios ofrecidos por el *middleware* residen en la misma computadora (ver el diagrama de emplazamiento mostrado en la Figura 18). Típicamente esta computadora tiene instalados todos los requisitos de *hardware* y *software* para poder manejar los dispositivos que necesite. Por ejemplo, si se desea controlar un dispositivo utilizando el puerto serie, es necesario que la computadora cuente con manejadores adecuados preinstalados[5]. El *middleware* se comunica indirectamente con el *hardware*

[5]Para el caso de Java éste puede ser RXTX. Disponible en http://users.frii.com/jarvi/rxtx/

que soporta mediante el uso de diferentes cápsulas sujeto, las cuales a su vez se comunican directamente con el *hardware* por distintos medios dependiendo del dispositivo en cuestión.

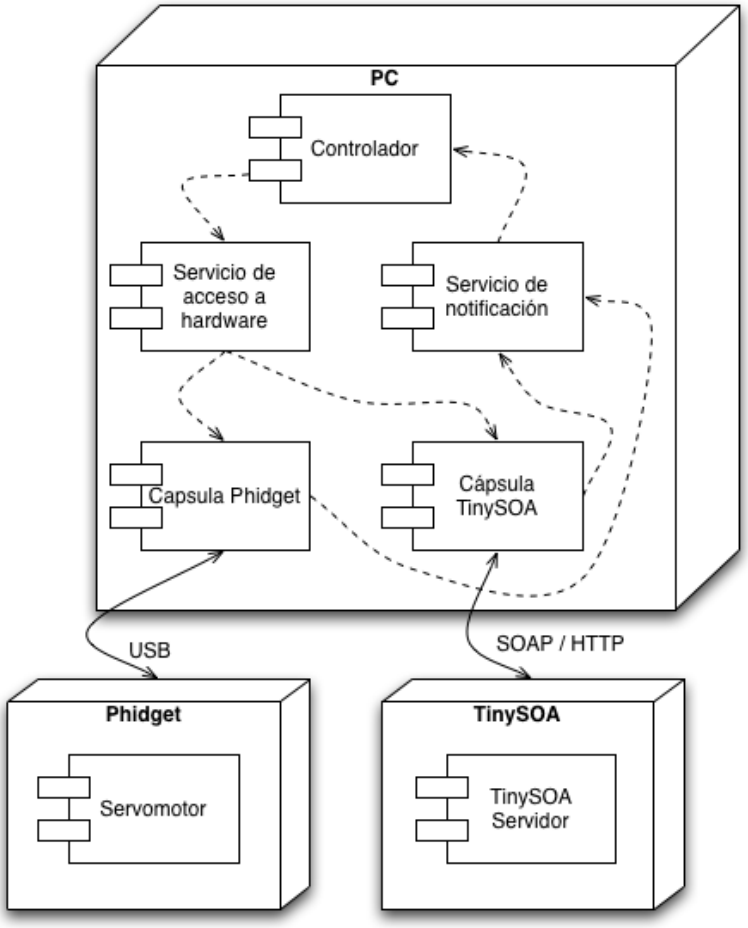


Figura 18: **Diagrama de emplazamiento de ActiveCloud**

El *hardware* con el que se comunica ActiveCloud puede residir en cualquier lugar, siempre y cuando la cápsula sujeto cuente con los medios necesarios para enviarle instrucciones de control. La limitante la proporciona la cápsula misma, y es responsabilidad del autor de la cápsula sujeto el ofrecer distintos medios de comunicación con el *hardware* si así se requiere. Una de las ventajas ofrecidas por la arquitectura es la separación de responsabilidades: el desarrollador de aplicaciones se preocupa solamente por programar el

controlador, y deja en manos de otro desarrollador (probablemente el fabricante del *hardware*) los detalles de comunicación.

V.2.1. Cápsulas

Las cápsulas sujeto y observadores contienen la implementación de la comunicación con el *hardware*, un archivo de configuración, una imagen que la identifica en el caso que se muestre en una interfaz gráfica y cualquier librería de soporte utilizada. El formato de archivo utilizado para las cápsulas es el formato de archivos de Java (JAR por las siglas en Inglés de *Java Archive*) el cual se basa en el formato ZIP. El formato JAR se utiliza para agregar varios archivos en uno solo y es básicamente un archivo ZIP con un directorio META-INF el cual es opcional (SUN, 1999). Se eligió este formato por la necesidad de empaquetar los varios archivos que forman una cápsula y a que Java proporciona soporte para este formato por defecto.

Una cápsula hereda de la clase `Publisher` u `Observer` para formar una cápsula sujeto u observador respectivamente, como se define en la Sección IV.3.3. A su vez, las clases `Publisher` y `Observer` implementan la interfaz `Capsule`. El sistema intentará agregar la cápsula a su repositorio de soporte llamando al método `init()` indistintamente de si se trata de una cápsula sujeto u observador.

Antes de cargar la cápsula, el sistema verifica que se trate de una cápsula válida. Se define como una cápsula válida aquel archivo que:

- Sea un archivo JAR que el sistema puede leer.
- Contenga un archivo llamado “config.xml” en la raíz del archivo JAR, que cumpla con los requerimientos del esquema del archivo de configuración de cápsulas (ver la Sección V.2.1).

- El cargador de clases estándar de Java lo pueda cargar.
- Contenga una clase que herede de `Publisher` u `Observer`, la cual se especifica en el archivo "config.xml" como la clase de inicio.
- Contenga todas aquellas bibliotecas de soporte que necesite para realizar sus funciones y que no formen parte del sistema.
- Contenga un archivo de imagen llamado "icon.png" con el formato de gráfico portátil de red (PNG por las siglas en Inglés de *Portable Network Graphics*). Aunque el tamaño de la imagen no es crítico, típicamente se utiliza una imagen con dimensiones de 120 pixeles de ancho por 120 pixeles de alto.

La estructura de directorios de una cápsula puede observarse en la Figura 19. En esta figura se muestra cómo la cápsula contiene los archivos requeridos (icon.png y config.xml) en la raíz (el directorio "src" es la raíz del archivo JAR). La clase que implementa la funcionalidad de la cápsula es PhidgetRFID.java.

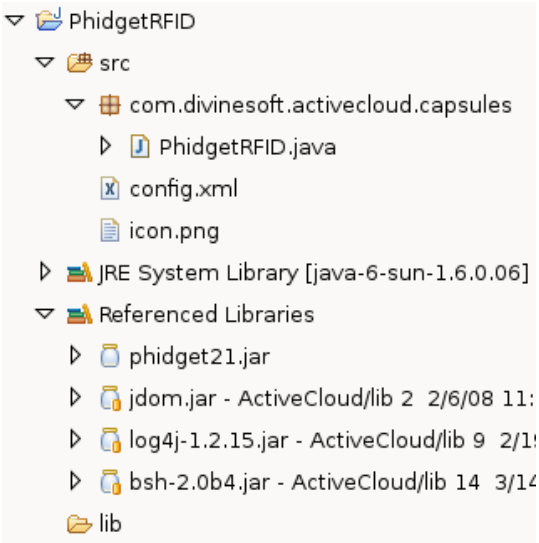


Figura 19: **Estructura de directorio para la cápsula sujeto "PhidgetRFID", la cual agrega soporte para la lectura de dispositivos phidget RFID.**

Si la cápsula cumple con los términos descritos, el sistema determina si se trata de una

cápsula sujeto u observador y procede a ejecutar su método `init()`. Este método es reservado para propósitos de inicialización, como podría ser el caso de establecer conexión con un servidor, abrir un puerto, comprobar que se encuentren instalados los manejadores necesarios para el *hardware* o asignar valores de configuración por defecto. Por último, el sistema coloca la cápsula en un directorio predeterminado y la hace disponible al controlador y a los servicios. El proceso de inicialización de cápsulas se ilustra en el diagrama de secuencia presentado en la Figura 20.

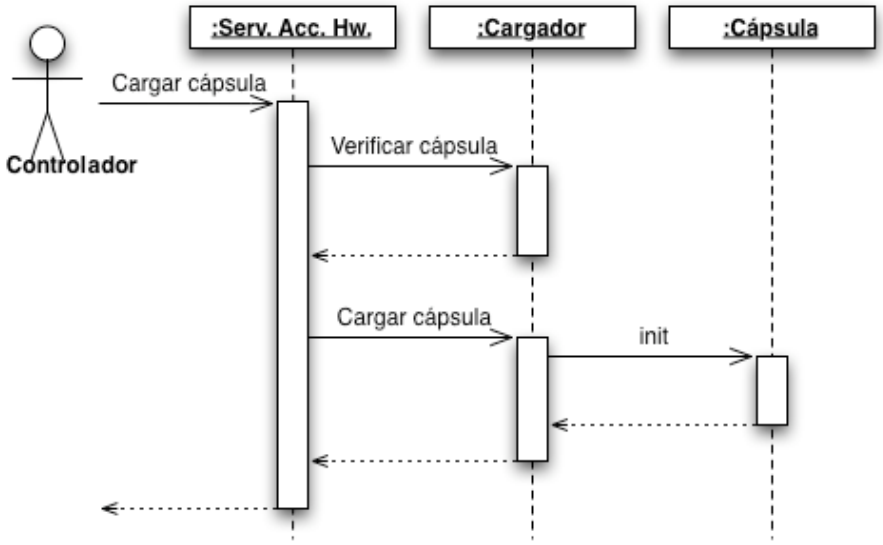


Figura 20: **Diagrama UML de secuencia que representa al proceso de inicialización de una cápsula.**

Esquema del archivo de configuración de cápsulas

El esquema del archivo de configuración de cápsulas es un formato sencillo basado en el lenguaje de marcado extendido (XML por las siglas en Inglés de *Extended Markup Language*) que dicta la forma que debe llevar el archivo de configuración de cápsulas. Como todo documento XML, éste debe iniciar con el prólogo (W3C, 2006). El contenido del archivo de configuración para la cápsula PhidgetRFID se muestra en el Listado V.1:

Listado de código fuente V.1: **Archivo de configuración**

```
1 <?xml version="1.0"?>
2 <capsule>
3         <subject class="com.divinesoft.activecloud.capsules.
              PhidgetRFID" />
4 </capsule>
```

En este caso, el nodo raíz `capsule` contiene un nodo hijo `subject` con el atributo `class`, cuyo valor es el nombre completamente calificado (es decir, incluye el nombre del paquete) de la clase que implementa la cápsula. El nodo raíz siempre debe ser del tipo `capsule` y éste a su vez solo puede tener un solo nodo.

La restricción es que el nodo raíz puede tener solamente un nodo hijo. El tipo de este nodo no es relevante ya que el sistema determina el tipo de cápsula examinando si es instancia de `Subject` u `Observer`. Por convención, el tipo del nodo hijo se asigna `subject` para las cápsulas sujeto y `observer` para las cápsulas observador. El nodo hijo debe contener el atributo `class` con el nombre de la clase que implementa la cápsula.

Opcionalmente el desarrollador de la cápsula puede incluir valores por defecto como parte del archivo de configuración. Estos valores por defecto se pasan a la cápsula por medio del método `init()`, siendo responsabilidad del desarrollador de la cápsula inicializar cualquier variable con los valores obtenidos del archivo de configuración. En el Listado V.2 puede observarse un archivo de configuración con valores por defecto:

Listado de código fuente V.2: **Archivo de configuración con valores por defecto**

```
1 <?xml version="1.0"?>
2 <capsule>
3         <observer class="com.divinesoft.activecloud.capsules.
              RSSCapsule"  />
```

```
4           <config>
5                   <key name="feedtype" value="atom_1.0" />
6                   <key name="filename" value="rsscapsule.xml" />
7                   <key name="encoding" value="utf-8" />
8                   <key name="feedtitle" value="RFIDoor" />
9                   <key name="maxentries" value="15" />
10                  <key name="timestampformat" value="HH:mm:ss z" />
11          </config>
12  </capsule>
```

El formato de los valores por defecto es un nodo `config` con uno o varios nodos hijo `key`. Cada nodo hijo `key` tiene dos atributos: `name` especifica el nombre de la variable y `value` su valor. En el Apéndice D se especifica la sintaxis que debe llevar el archivo de configuración y en el Apéndice A se detalla el proceso de creación de cápsulas sujeto y observador.

Acciones

Las cápsulas sujeto contienen una lista de acciones (ver Sección IV.3.3). Una acción es funcionalidad que ofrece el *hardware* al que controla la cápsula sujeto y se pueden invocar por medio del servicio de acceso a *hardware* (ver Sección V.2.2). No existe una limitante (más allá de la memoria disponible a la computadora) en el número de acciones que una cápsula sujeto pueda publicar. También es posible crear acciones «virtuales», es decir, acciones que no correspondan directamente con una funcionalidad específica del *hardware* sino que se apoyan de la computadora para ser ejecutadas. Un ejemplo es el obtener máximos y mínimos de las lecturas históricas de un sensor, aún cuando el sensor no soporta esta operación directamente.

Una acción extiende a la clase abstracta `Action`, la cual define el método abstracto

run(). Este método contiene la implementación de la acción, y se ejecuta cuando la acción se invoca. Al método `run()` se le pasa un elemento de configuración con formato XML en donde se listan parámetros específicos a la ejecución de dicha acción. Un ejemplo es definir la fecha de inicio y final para la acción de obtener las lecturas históricas de un sensor. Cada acción tiene un nombre y una descripción definidos por el autor de la cápsula sujeto. El nombre se utiliza para ejecutar la acción. En la Figura 21 se muestra la representación en diagrama UML de clases para las acciones.

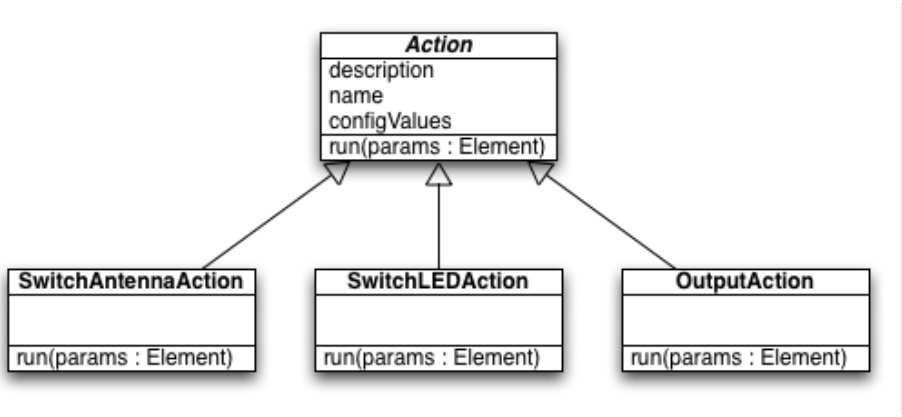


Figura 21: **Diagrama UML de clases para la clase abstracta `Action` y la implementación de las acciones que la cápsula «PhidgetRFID» proporciona**

V.2.2. Servicio de acceso a *hardware*

Como se estableció anteriormente, existen dos tipos de cápsulas: sujeto y observador. El servicio de acceso a *hardware* es un punto de acceso único hacia toda la funcionalidad que el *middleware* proporciona alrededor del concepto de cápsulas. La funcionalidad principal de este servicio es el de agregar o «cargar» una cápsula al sistema mientras éste se encuentra en ejecución, así como proporcionar los medios para obtener una instancia de cualquier cápsula disponible al sistema. El servicio evita que una cápsula se agregue más de una vez al sistema, verificando el nombre completamente calificado de la clase principal que compone a la cápsula. Cada cápsula observador se puede asignar a tantas cápsulas

sujeto como sea necesario, guardando una configuración distinta para cada vínculo entre sujeto y observador.

El *middleware* puede recibir notificación de que existe una cápsula nueva que se necesita ser cargar de distintas maneras (ver la Sección V.2.5). Cualquiera que sea la forma, la cápsula se agrega al sistema llamando a la función `loadCapsule()`. Esta función recibe el archivo JAR que compone a la cápsula e inicia el proceso de validación descrito en la sección V.2.1. Una vez validada y en caso de no haber sido ya agregada, la cápsula se carga al sistema y las acciones que publica se ponen a la disposición del controlador.

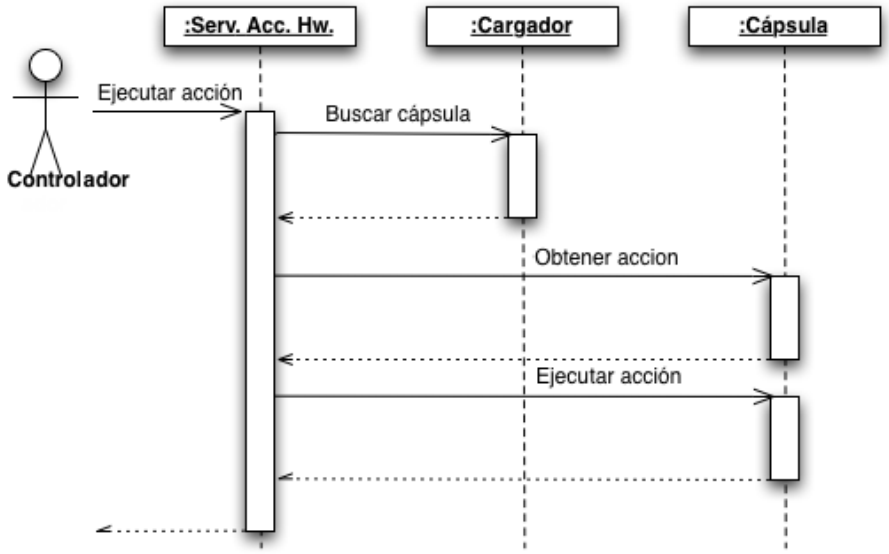


Figura 22: **Diagrama UML de secuencia para el proceso de ejecución de una acción**

Para ejecutar una acción se hace por medio del método `invokeAction()` que recibe como parámetros el nombre de la acción a ejecutar y los parámetros que se desea enviar a la acción, de ser necesarios. El sistema localiza la cápsula sujeto que provee la acción solicitada y crea un nuevo hilo de ejecución dentro del cual será ejecutada la acción, enviándole los parámetros recibidos inicialmente. Este proceso se muestra en la Figura 22.

El paso de parámetros a las acciones es por medio de un nodo XML que varía en formato entre cada una de las acciones. Por convención, el formato implementado es muy similar al mostrado en el Listado V.2, con la diferencia de tratarse solamente del nodo `config` y no del documento XML completo. En el Listado V.3 se muestra el nodo de configuración para la ejecución de la acción `RFID.SwitchAntenna` de la cápsula sujeto «PhidgetRFID»:

Listado de código fuente V.3: **Parámetros para la acción RFID.SwitchAntenna**

```
1 <config>
2         <key name="antennastate" value="true" />
3 </config>
```

También es posible especificar los parámetros en un formato más sencillo, esto con el objetivo de facilitar la escritura de peticiones externas. El formato es simplemente *variable1=valor1, variable2=valor2,..., variableN=valorN*. Esta representación más corta de los parámetros de ejecución de una acción se convierte internamente al mismo formato mostrado en el Listado V.3 y no lo interpreta directamente la acción. Por último, es posible invocar una acción sin especificar parámetros cuando esta no los requiera. La ejecución de una acción utilizando el formato alternativo se muestra en el listado V.4.

Listado de código fuente V.4: **Ejecución de acción con formato alternativo**

```
1 NodeAccessService.get().invokeAction("Out.Switch","state=True,index
    =5");
```

V.2.3. Servicio de notificación

El objetivo del servicio de notificación es el de administrar el ambiente «sujeto - observador» de las cápsulas. Este servicio lo utilizan las cápsulas sujeto para publicar los datos generados a partir de la ejecución de una acción. También lo utiliza el controlador para registrar y eliminar subscripciones de una cápsula observador a una cápsula sujeto.

Al suscribir una cápsula observador a una cápsula sujeto se forma un «vínculo de entrega» (`DeliveryLink`). Un vínculo de entrega liga a una cápsula sujeto, una cápsula observador y una configuración de ejecución la cual dicta los parámetros con los cuales contará la cápsula observador al momento que se invoque su método `receive()`. Una vez que se crea un vínculo de entrega, al momento de que una cápsula sujeto publique datos, el servicio de notificación invocará el método `receive()` de las cápsulas observador con las cuales la cápsula sujeto cuente con vínculo de entrega.

Para publicar datos, una cápsula sujeto hace uso del método `publish()` del servicio de notificación. La cápsula sujeto envía un nodo XML como parámetro que contiene, entre otras cosas, la clase de la cápsula sujeto que envía dichos datos y los datos en si. En el Listado V.5 se muestra el nodo XML que envía la cápsula sujeto «PhidgetRFID» al ejecutar su acción `RFID.SwitchAntenna`:

Listado de código fuente V.5: **XML enviado al ejecutar la acción SwitchAntenna**

```
1 <payload class="com.divinesoft.activecloud.capsules.PhidgetRFID">
2         <entry name="RFID.SwitchAntenna" value="1" />
3 </payload>
```

Cada cápsula sujeto es libre de enviar cuantos nodos `entry` como sean necesarios, sin embargo el formato para el nodo `payload` debe seguir la guía mostrada en el Listado V.5. Para el caso del controlador, el servicio de notificación le hace llegar todos los datos generados por las cápsulas sujeto por medio de eventos estándar de Java.

V.2.4. Servicio de procesamiento de peticiones

El servicio de procesamiento de peticiones se implementa como un lenguaje interpretado (de *script*) dentro del *middleware*. Para ejecutar una petición se utiliza el método `eval` enviando como parámetro la petición. El servicio cuenta con un intérprete del lenguaje

script que se encarga de completar la petición en caso de ser posible.

Con el propósito de facilitar la creación de peticiones, se seleccionó *BeanShell*[6] como intérprete. *BeanShell* permite crear programas *script* en el lenguaje Java con la posibilidad de eliminar los tipos de datos, efectivamente creando un lenguaje *script* con la misma sintaxis y las mismas herramientas disponibles al lenguaje Java, dentro de la máquina virtual en la que se está ejecutando el resto del *middleware*. Es posible trabajar con objetos reales Java que se ejecuten al mismo tiempo que el *script* se evalúa, también es posible intercambiar datos hacia dentro o fuera del *script* permitiendo una integración simple e imperceptible entre el programa Java y el *script*.

La sintaxis de una petición es la misma que la de cualquier programa Java. Esto representa una ventaja para el desarrollador ya que no es necesario que aprenda otro lenguaje: tanto el controlador, como las cápsulas y las peticiones utilizan el mismo lenguaje (Java). El servicio de procesamiento de peticiones también se utiliza dentro de las cápsulas, permitiendo automatizar la ejecución de acciones para el caso de las cápsulas sujeto, o filtrar automáticamente los datos publicados para las cápsulas observador.

V.2.5. Servicio de acceso al sistema de archivos

El objetivo de este servicio es del de facilitar la interacción entre el *middleware* y el sistema de archivos local. El servicio de acceso al sistema de archivos provee una API independiente de la plataforma que permite recibir eventos del sistema de archivos local cuando se detecta la presencia de una nueva cápsula, con lo cual el controlador puede utilizar el servicio de acceso a *hardware* para agregar la nueva cápsula al *middleware*.

Este servicio delega el monitoreo del sistema de archivos a distintas «tareas». Cada

[6] http://www.beanshell.org/

una de estas tareas se encarga de generar los eventos pertinentes cuando ocurre una operación sobre alguno de los directorios monitoreados por el servicio de acceso al sistema de archivos. En la Figura 23 se muestra la interfaz `FileSystemTask` que define el método `run()`, que la implementan tareas concretas para comenzar el monitoreo de los directorios definidos.

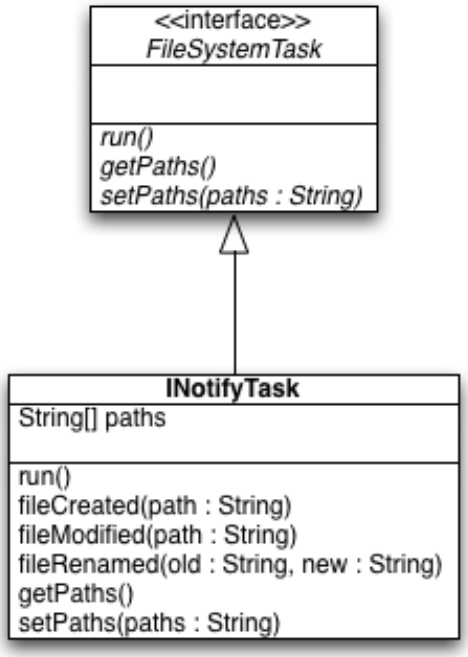


Figura 23: **Diagrama UML de clases para las tareas del sistema de archivos**

Para iniciar el monitoreo se ejecuta el método `start()` del servicio, al cual se le pasa como parámetro un arreglo con las rutas de los directorios que se deben monitorear. El servicio le pasa estas rutas a la tarea configurada y ejecuta su método `run()`. Debido a que el acceso al sistema de archivos depende directamente del sistema operativo, la arquitectura del servicio de acceso al sistema de archivos y las tareas de monitoreo se pensó para permitir la implementación de distintas tareas, de tal manera que exista una tarea de monitoreo por tipo de sistema de archivos a soportar.

El soporte para el sistema de archivos de Windows y GNU/Linux se proporciona a través de `INotifyTask`. Esta tarea de monitoreo utiliza la librería *JNotify*[7] que le permite a las aplicaciones Java recibir eventos del sistema de archivos. Bajo Windows, *JNotify* utiliza la propia API de Windows, bajo Linux hace uso de inotify[8] el cual es un subsistema del núcleo Linux para proporcionar los eventos del sistema de archivos. Se pueden soportar otros sistemas operativos mediante la creación de nuevas tareas de monitoreo.

V.3. ActiveCloud GUI

Con el objetivo de comprobar la funcionalidad del sistema se creó un programa que sirve como interfaz gráfica a los servicios que proporciona el *middleware* ActiveCloud. Este programa denominado ActiveCloud GUI permite al usuario instalar nuevas cápsulas, invocar acciones, automatizar cápsulas sujeto, administrar observadores y crear controladores mediante el uso del lenguaje *script*.

Al iniciar, ActiveCloud GUI muestra una pantalla dividida en tres secciones verticales como se observa en la Figura 24. La columna izquierda muestra las cápsulas sujetos instaladas y la columna derecha muestra las cápsulas observador, en la Figura 24 se muestra la cápsula sujeto PhidgetRFID y la cápsula observador Chart instaladas. La columna del centro cambia de acuerdo a la sección en la que se encuentra el usuario en ese momento. En la pantalla inicial la columna central muestra la consola, desde donde se pueden evaluar peticiones externas y crear controladores por medio del lenguaje *script*. Desde la consola es posible utilizar todos los servicios proporcionados por el *middleware*.

Uno de los objetivos de la arquitectura es el de permitir la instalación de nuevas cápsulas en tiempo de ejecución. Para lograrlo se apoya tanto del servicio de acceso al sistema

[7] http://sourceforge.net/projects/jnotify/
[8] http://www.kernel.org/pub/linux/kernel/people/rml/inotify/README

Figura 24: **Pantalla de inicio de ActiveCloud GUI mostrando dos cápsulas instaladas**

de archivos (monitoreando el directorio de instalación de cápsulas) y del servicio de acceso a *hardware* (instalando las cápsulas detectadas por el sistema de archivos). ActiveCloud GUI permite la instalación de nuevas cápsulas de manera sencilla: se arrastra y suelta con el ratón de la computadora el archivo JAR sobre la pantalla de instalación y la cápsula se instala e inicializa. La pantalla de instalación se muestra en la Figura 25.

Para configurar los observadores, ActiveCloud GUI proporciona la vista *Observers*, en donde se muestra una tabla con los observadores definidos actualmente. Es posible agregar y eliminar observadores utilizando dicha tabla o directamente desde la consola, utilizando el servicio de notificación. Al presionar sobre el icono de una cápsula sujeto se carga la vista de *script*, donde es posible automatizar las tareas de una cápsula sujeto como se muestra en la Figura 26.

Es posible utilizar el *middleware* ActiveCloud sin el uso de la interfaz gráfica. Sin embargo, ActiveCloud GUI permite familiarizarse con muchos de los conceptos clave del *middleware*, principalmente la utilización de los servicios, la configuración de cápsulas sujeto,

Figura 25: **Pantalla de instalación de cápsulas de ActiveCloud GUI. Para instalar una cápsula nueva esta se arrastra y suelta sobre la pantalla de instalación**

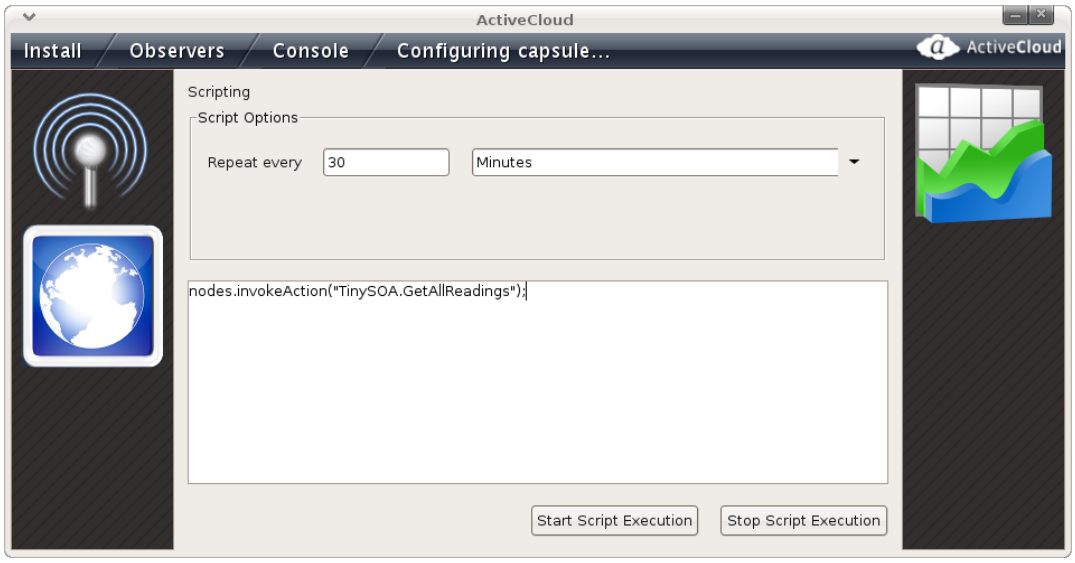


Figura 26: **Pantalla de *script* para las cápsulas sujeto**

observador y la creación de *scripts*. También, ActiveCloud GUI es útil en tiempo de desarrollo. Se puede probar cada una de las cápsulas independientemente o ligarlas por medio de la interfaz gráfica y programas *script*. Una vez que se probaron cada uno de los componentes de manera individual, se puede crear el controlador y eliminar el uso de la interfaz gráfica.

V.4. Conclusión

En el presente capítulo se presentaron los detalles de la implementación del *middleware* ActiveCloud, el cual sigue la arquitectura presentada en el Capítulo IV. ActiveCloud implementa los servicios descritos en la arquitectura, con excepción del servicio de mantenimiento de red. También implementa el concepto de cápsulas, permitiendo extender su funcionalidad en tiempo de ejecución.

Al momento de la implementación de ActiveCloud se crearon cápsulas sujeto para TinySOA (que proporciona soporte para los motes MICAz), el kit eKo y algunos phidgets. Entre las cápsulas observador desarrolladas se encuentran Chart que permite representar de manera gráfica los datos obtenidos, TinySOAEvent que genera los mismos eventos que TinySOA y permite la compatibilidad entre programas creados con TinySOA y los controladores de ActiveCloud, RSSCapsule que genera un canal RSS con los datos obtenidos y XMLPrinter que simplemente imprime los datos obtenidos a la salida estándar. Por último, se creó la interfaz gráfica ActiveCloud GUI que permite el uso de los servicios de ActiveCloud de manera gráfica.

Capítulo VI

Evaluación

En este capítulo se presenta la evaluación realizada a la arquitectura por medio de la implementación del *middleware* prototipo. La evaluación se hizo a manera de experimento con un grupo de usuarios potenciales a quienes se les dio la tarea de usar el *middleware* ActiveCloud para realizar ciertas tareas, con el objetivo de estimar el grado en el que la arquitectura cumple con los objetivos propuestos. También se evaluaron los requerimientos presentados en el Capítulo IV para verificar su cumplimiento.

VI.1. Diseño del experimento

La evaluación del prototipo se realizó para determinar si la arquitectura propuesta y el *middleware* ActiveCloud cumplen con los objetivos del presente trabajo, los cuales se derivan de los requerimientos de los sistemas *middleware* para redes WSAN. El experimento de evaluación y los resultados obtenidos se presentan a continuación. Se inicia describiendo el escenario utilizado para el experimento, las variables medidas y la mecánica de las sesiones de evaluación. En la Sección VI.3 se discuten los resultados obtenidos de la evaluación, los cuales se utilizan para dar respuesta a las preguntas de investigación planteadas en el Capítulo I. En la evaluación se contó con la participación de 16 personas, entre estudiantes de la maestría en ciencias de la computación y profesionistas del área de cómputo. La población incluyó personas con distintos niveles de conocimiento en el área de redes WSN y WSAN, así como también distintos niveles de experiencia en programación orientada a objetos.

VI.1.1. Variables del experimento

Como variable independiente se definió el lenguaje de programación (Java, por las razones expuestas en la Sección V.2) y el entorno de desarrollo. Todos los usuarios potenciales utilizaron el mismo ambiente integrado de desarrollo (IDE por las siglas en Inglés de *Integrated Development Environment*), el mismo lenguaje y la misma versión de ActiveCloud. Como variables dependientes se seleccionaron el número de tareas completadas satisfactoriamente y el tiempo que se llevó terminar las tareas.

VI.1.2. Tareas a realizar

Las tareas a realizar cubren una parte significativa de la funcionalidad provista por el *middleware*. Estas tareas ilustran el proceso de desarrollo de una aplicación completa, sin embargo, su alcance es menor debido a las restricciones de tiempo bajo las que se realizó la evaluación. La evaluación se dividió en dos partes. En la primera parte se le da una introducción al grupo de usuarios potenciales al mismo tiempo que utilizan ActiveCloud GUI (el componente gráfico del *middleware*) con el fin de familiarizarse con el *middleware*. Las tareas a realizar fueron:

1. Cargar una cápsula sujeto y una cápsula observador.
2. Ejecutar una acción de control.
3. Registrar una cápsula observador a una cápsula sujeto.

La segunda parte consistió en la creación de un controlador para realizar una aplicación de control utilizando ActiveCloud. En esta segunda parte se le entregó a los usuarios potenciales un proyecto configurado en el IDE Eclipse[1], el cual contenía el «esqueleto» de un controlador. El objetivo del controlador fue encender un LED cuando el nivel de luminosidad del ambiente se encontraba por debajo de un intervalo determinado.

[1] http://www.eclipse.org/

VI.1.3. Condiciones del experimento

El experimento se realizó en un laboratorio de cómputo. Se utilizaron cuatro computadoras conectadas en una red local ejecutando Windows XP como sistema operativo, a las cuales se les instaló el proyecto del experimento, el middleware ActiveCloud, el lenguaje Java versión 1.6, la implementación de mDNS de Apple para Windows (llamada *Bonjour*[2]) y el ambiente de desarrollo Eclipse versión 3.4. Como se muestra en la Figura 27, el servidor TinySOA se instaló en otra computadora localizada fuera del laboratorio ejecutando GNU/Linux como sistema operativo.

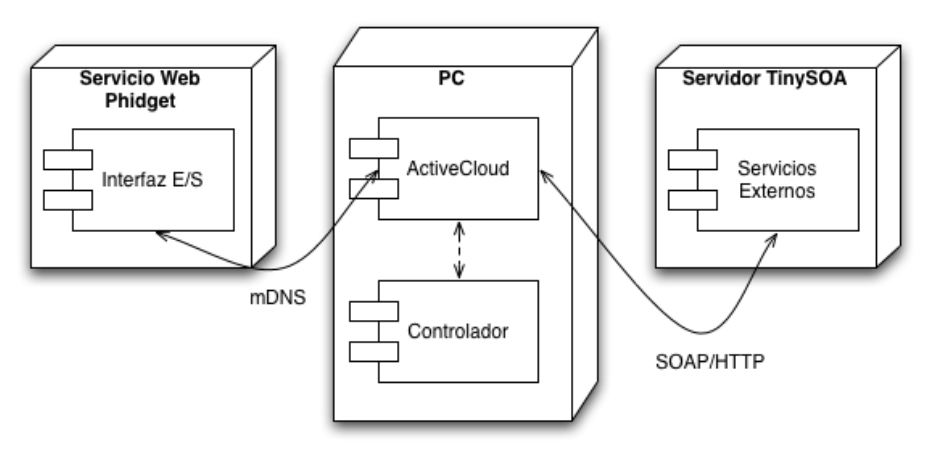


Figura 27: **Diagrama de emplazamiento del experimento.**

Para efectos de la evaluación, el actuador se implementó con una tarjeta *phidget* de entrada y salida. Esta tarjeta cuenta con ocho entradas digitales, ocho salidas digitales y ocho entradas analógicas. Se colocó un LED (diodo emisor de luz, por las siglas en Inglés de *Light Emitting Diode*) en cuatro de las salidas digitales, mismo que sirvió para indicar de manera visual cuándo había sido accionado un actuador. La tarjeta se conectó al puerto USB de una computadora portátil ejecutando el sistema operativo OSX versión 10.5. Esta computadora portátil ejecutaba además un servicio Web por medio del cual se comunicaba la cápsula dedicada al soporte de la tarjeta de E/S. La comunicación entre la cápsula

[2]http://www.apple.com/support/downloads/bonjourforwindows.html

sujeto y el servicio Web del *phidget* fue por medio del protocolo *Multicast DNS*[3].

VI.1.4. Procedimiento

La evaluación se efectuó en dos sesiones de cuatro usuarios y en una sesión de ocho usuarios, debido a las restricciones del *hardware* disponible. Al inicio de la evaluación se realizó una presentación básica sobre las redes inalámbricas de sensores y actuadores, resaltando las características únicas de las redes WSAN y la problemática que presentan. Posteriormente se explicó el *middleware* ActiveCloud, sus componentes principales y cómo se relacionan para formar programas de control. También se proporcionaron ejemplos de uso mediante el componente ActiveCloud GUI, al tiempo que los usuarios potenciales ejecutaban los ejemplos en el sistema. Una vez realizada la introducción y los ejemplos de uso, se procedió a realizar la aplicación de prueba descrita en la Sección VI.1.2. Al finalizar se aplicó un cuestionario de salida (ver Apéndice B) con el fin de medir la percepción de los desarrolladores hacia la arquitectura presentada y el *middleware* prototipo. El tiempo total de cada una de las sesiones de evaluación fue de aproximadamente 1 hora.

VI.2. Descripción de la población

En la evaluación participaron 16 personas, entre los cuales se encontraron estudiantes de maestría en ciencias de la computación, estudiantes de doctorado en ciencias de la computación y profesionistas del área de cómputo. El grupo de usuarios potenciales incluyó a personas con experiencia en el desarrollo de aplicaciones orientadas a objetos y patrones de diseño, así como también a personas nóveles al tema.

[3] http://www.multicastdns.org/

El lenguaje de programación más popular entre los usuarios potenciales fue Java, como se puede observar en la Figura 28. También, el 50 % (8 personas) de los usuarios potenciales se encuentran familiarizados con los conceptos de la programación orientada a objetos, como se muestra en la Figura 29. El 88 % (14 personas) de los usuarios potenciales declaró desconocer el patrón de diseño «Observador» (también conocido como *Publish - Subscribe*).

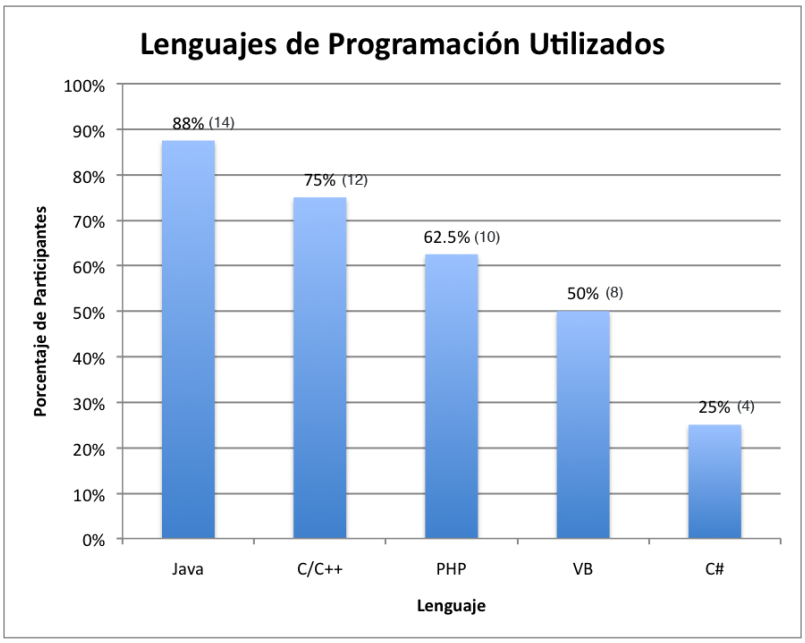


Figura 28: **Porcentaje de participantes con experiencia en distintos lenguajes de programación.**

La programación de aplicaciones de control con la arquitectura propuesta requiere de la correcta identificación de los componentes que manejarán las cápsulas sujeto y observador, así como la relación que existe entre ellas. Por tal motivo, el cuestionario de salida incluyó una sección con el propósito de evaluar el nivel de comprensión del modelo de comunicación (ver Apéndice C).

El grupo de usuarios potenciales se dividió en usuarios nóveles y expertos. Se considera un usuario experto aquel con experiencia en desarrollo de aplicaciones utilizando el

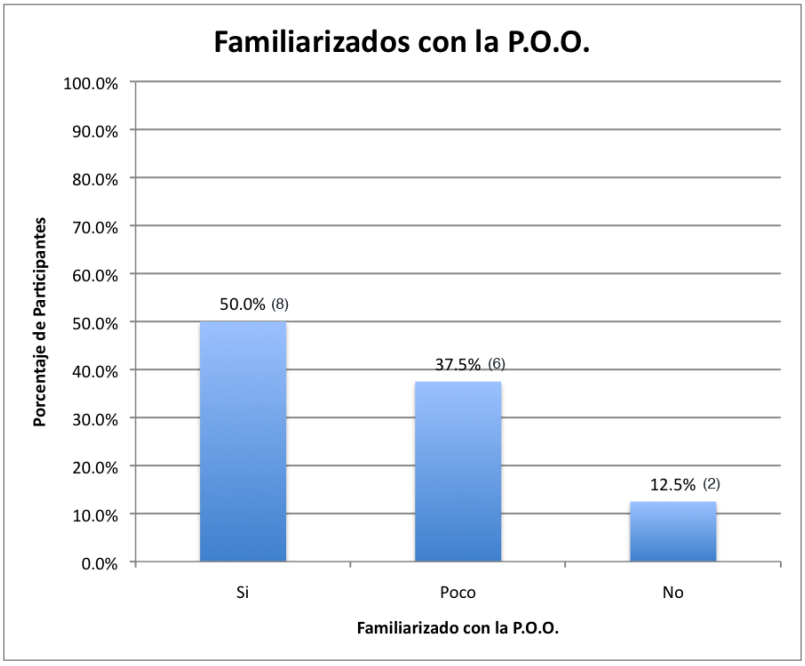


Figura 29: **Porcentaje de participantes familiarizados con la programación orientada a objetos.**

paradigma orientado a objetos, además de contar con conocimiento previo en el área de redes inalámbricas de sensores. Como se puede observar en las Figuras 29 y 30, el grupo de usuarios expertos es menor que el total de usuarios con experiencia en programación orientada a objetos.

VI.3. Resultados

El cuestionario de salida (ver Apéndice B) se compone de 13 oraciones en escala Likert (Likert, 1932). En la escala Likert, el rango de valores posibles va desde «completamente en desacuerdo» (con valor numérico 1) hasta «completamente de acuerdo» (con valor numérico de 7). Las oraciones formuladas en escala Likert miden el nivel en el que se está en acuerdo o en desacuerdo con cierta oración. Las oraciones en escala Likert se utilizan tradicionalmentes en pruebas de actitud y en el modelo de aceptación de tecnología (TAM por las siglas en Inglés de *Technology Acceptance Model*). La validez y fiabilidad del uso de las oraciones en escala Likert y del modelo TAM se han demostrado en diversos trabajos

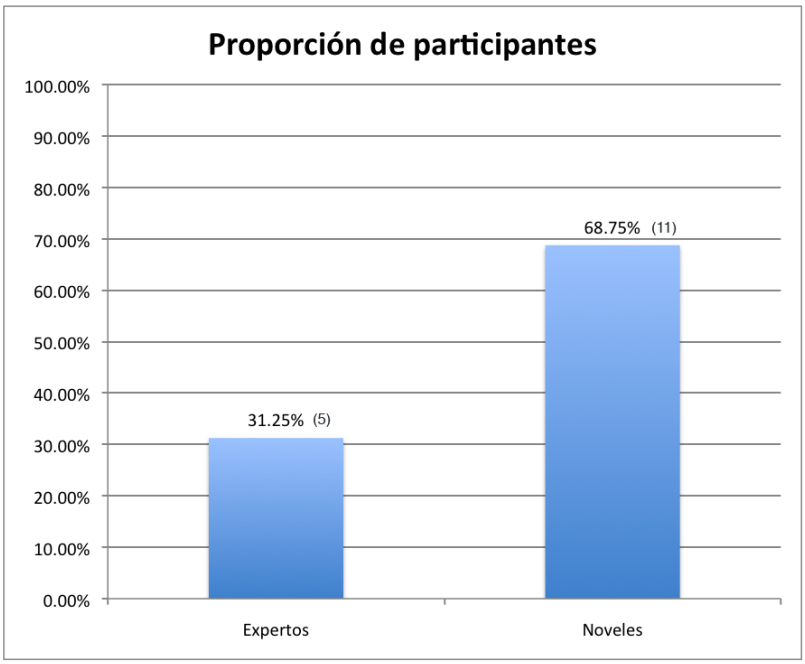


Figura 30: **Porcentaje de participantes expertos y nóveles.**

(Davis, 1989; 1985).

Las oraciones fueron agrupadas de acuerdo al concepto que intentan medir. Se formaron tres grupos de oraciones, que corresponden a las preguntas de investigación planteadas en el Capítulo I: facilidad (grado de esfuerzo), utilidad y nivel de abstracción. Se obtuvo la media aritmética y la desviación estándar tanto de las oraciones individuales como de cada uno de los grupos formados. Esto se muestra en las Tablas I-III.

Facilidad (grado de esfuerzo)	Media	Desviación Estándar
Encuentro fácil el uso de ActiveCloud	6.25	0.707
Considero que no tendría mayor problema en la adopción de ActiveCloud	5.875	0.834
ActiveCloud me facilita la obtención de datos de múltiples fuentes	6.25	0.462
Promedio	6.125	0.667

Tabla I: **Resultados para el grupo de preguntas correspondiente a la hipótesis 1**

Utilidad	Media	Desviación Estándar
Considero posible la integración de los datos generados por las cápsulas sujeto con otros sistemas	6.5	0.886
El API de ActiveCloud es suficiente para el desarrollo de aplicaciones de control con WSAN	6.125	0.353
Considero que es conveniente utilizar ActiveCloud para el desarrollo de aplicaciones de control con WSAN	6.25	1.164
Considero que ActiveCloud puede ayudarme en el desarrollo de aplicaciones de control con WSAN	6.625	0.517
En general, encuentro útil el middleware ActiveCloud	6.375	0.744
Promedio	6.375	0.732

Tabla II: **Resultados para el grupo de preguntas correspondiente a la hipótesis 2**

Nivel de abstracción	Media	Desviación Estándar
ActiveCloud me oculta los detalles de bajo nivel para el desarrollo de aplicaciones con WSAN	5.5	1.772
Considero que la API de ActiveCloud es coherente y permite el manejo de todos los dispositivos que integran la red WSAN	6.125	0.640
ActiveCloud me permite la utilización de hardware heterogéneo	6.125	0.991
Considero que ActiveCloud puede soportar nuevo hardware que aún no se encuentra disponible (mediante la creación de nuevas cápsulas sujeto)	6	0.925
El ambiente publish/subscribe es adecuado para el desarrollo de aplicaciones de control con WSAN	6.5	0.534
Promedio	6.05	0.972

Tabla III: **Resultados para el grupo de preguntas correspondiente a la hipótesis 3**

Los resultados obtenidos para los tres grupos muestran valores en el rango de 6 y 6.5 en una escala del 1 al 7, lo que corresponde a «ciertamente de acuerdo». Los usuarios potenciales lograron identificar las cápsulas necesarias para el desarrollo del programa de evaluación; inclusive usuarios nóveles sin experiencia en programación orientada a objetos lograron identificar correctamente las cápsulas necesarias y definir la interacción entre ellas. En la Figura 31 se muestra cómo se distribuye el número de errores cometidos en la evaluación al modelo de comunicación (ver Apéndice C) entre los usuarios nóveles y expertos. Como error se consideró la identificación incorrecta de una cápsula sujeto u observador o la definición incorrecta de la interacción entre ellas. La figura muestra más errores cometidos por usuarios nóveles. Sin embargo, la relación entre el número de errores cometidos y el número de usuarios, para usuarios nóveles y expertos es muy similar

(0.4 errores por persona para usuarios expertos y 0.6 errores por persona para usuarios nóveles). Esto indica que la experiencia previa en el desarrollo para redes WSN o WSAN, así como la experiencia previa en el patrón «Sujeto - Observador» no son un factor crítico para la creación de aplicaciones con la arquitectura propuesta.

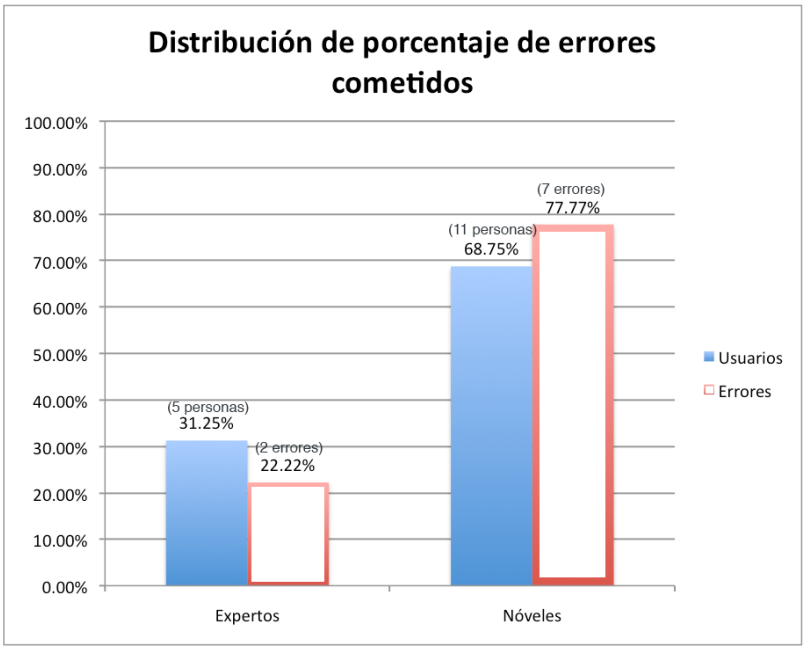


Figura 31: **Distribución de porcentaje de errores cometidos entre los participantes.**

Las preguntas de investigación "¿Cuál es el grado de utilidad de un sistema que permita la programación de aplicaciones para redes WSAN con soporte dinámico a *hardware* heterogéneo?" y "¿Cuál es el grado de esfuerzo que se requiere para crear aplicaciones con la arquitectura propuesta?" planteadas en el Capítulo I se refieren a los conceptos de facilidad y utilidad percibida. Estos dos conceptos se miden con el modelo TAM. Los resultados obtenidos (ver Tablas I y III) muestran que, según la percepción de los usuarios potenciales, al utilizar el *middleware* incrementan su desempeño en el desarrollo de aplicaciones para redes WSAN (utilidad percibida), sin requerir mayor esfuerzo (facilidad percibida).

En la premisa "ActiveCloud me oculta los detalles de bajo nivel para el desarrollo de

aplicaciones con WSAN", parte del conjunto que trata sobre el nivel de abstracción (ver Tabla II) se obtuvo una media de 5.5 con desviación estándar de 1.77, lo que corresponde a ligeramente de acuerdo. Lo anterior refleja la percepción por parte de los usuarios potenciales de que el *middleware* no oculta los detalles de bajo nivel lo suficiente. De acuerdo a algunos comentarios realizados en el transcurso de la evaluación, así como también en el cuestionario de salida, el hecho de que algunos dispositivos (por ejemplo los nodos con TinySOA) requieran de la especificación de una dirección IP y otros dispositivos no lo requieran, obliga a los desarrolladores a pensar en el emplazamiento de la red. Aún cuando todos los dispositivos de la red se manejen de forma consistente con la misma API, la necesidad de conocer los detalles de emplazamiento dificulta el desarrollo de aplicaciones. Lo anterior es un aspecto del cual se toma nota para considerarlo como mejora o trabajo futuro.

Como comentarios generales sobre el *middleware* y la arquitectura, los usuarios potenciales manifestaron no tener mayor problema con el concepto de cápsulas y el modelo de comunicación basado en el patrón observador. Una sugerencia fue la de incluir un tutorial o ejemplos de programas cortos realizados con el *middleware* para complementar la documentación estándar.

VI.4. Prueba a los requerimientos del *middleware*

En la Sección VI.1 se menciona que los objetivos del presente trabajo se derivan de los requerimientos de los sistemas *middleware* para redes WSAN.

VI.4.1. Requerimiento de creación de aplicaciones

En la Sección IV.1 se establece que la principal aplicación que se le da a las redes WSAN es la de crear sistemas de automatización y control distribuído. El *middleware* debe permitir la creación de dichas aplicaciones, eliminando la necesidad de la programación de bajo nivel, proporcionando una API consistente para todos los dispositivos que integran la red, proporcionando soporte dinámico de *hardware* heterogéneo, así como también la posibilidad de intercambio de datos entre la red y sistemas existentes.

El diseño de la arquitectura se propuso a partir del estudio de resultados obtenidos por proyectos similares, específicamente de (Aberer *et al.*, 2006) y (Avilés-López y García-Macías, 2007). También, el *middleware* prototipo se utiliza por otros proyectos de investigación donde se requiere una capa de abstracción para el manejo de *hardware* heterogéneo dedicado a la obtención de datos del medio ambiente.

Gracias a la división del sistema completo en módulos de funcionalidad (cápsulas) y a la API ofrecida por el *middleware*, es posible crear aplicaciones de control distribuído con redes WSAN. La aplicación desarrollada para la evaluación es muestra de lo anterior: utilizando una red de sensores formada con nodos MicaZ ejecutando el sistema TinySOA, así como una tarjeta phidget de E/S y ActiveCloud, se creó un controlador que monitorea el nivel de luminosidad continuamente y enciende un LED cuando dicho nivel se encuentra en un rango predeterminado. Esta aplicación fue creada por los participantes de la evaluación sin conocer detalles de bajo nivel de la red emplazada.

VI.4.2. Requerimiento de compatibilidad entre sistemas

El modelo de comunicación permite que el total o un conjunto de los datos generados se procesen, aún cuando estos datos no generen una acción de control.

Con el propósito de evaluar la compatibilidad entre sistemas, se integró el *middleware* a una aplicación de monitoreo ambiental, la cual consultaba datos de una red WSN. Al utilizar ActiveCloud, fue necesario adaptar los datos generados de tal manera que se minimizara el impacto en la aplicación programada, evitando la reprogramación de las rutinas de análisis de datos. Para lograrlo se creó una cápsula observador que toma los datos originales y los convierte al formato utilizado en la aplicación original. El resultado es una aplicación de monitoreo capaz de utilizar distintas fuentes de datos sin necesidad de modificar las rutinas de análisis debido a que el formato de estos datos se mantiene. La utilización del *middleware* en otro proyecto de investigación comprueba la factibilidad del intercambio de datos entre sistemas y de la integración del *middleware* con sistemas existentes.

VI.5. Cumplimiento de los objetivos planteados

En el Capítulo IV se plantea como objetivo del presente trabajo "desarrollar una arquitectura que permita la programación en alto nivel de aplicaciones para redes WSAN, brindando soporte dinámico a *hardware* heterogéneo". Los resultados obtenidos por medio de la evaluación realizada permiten verificar el cumplimiento del objetivo planteado.

Primeramente, el soporte a *hardware* heterogéneo se refiere no solo a la capacidad de la arquitectura de dar soporte a distintos dispositivos, sino también de hacerlo de manera transparente al usuario, es decir, que no exista diferencia entre el manejo de un dispositivo u otro. Este concepto es similar los «flujos de entrada y salida» (*I/O streams*) en el lenguaje de programación Java, en donde mediante el uso de la misma API se pueden obtener los datos contenidos en un archivo de texto, un archivo binario, una cadena contenida dentro del mismo programa o entrada del usuario, entre otras fuentes de datos. De la misma manera, la arquitectura debe permitir el uso de distintas fuentes de datos (en este caso distintos dispositivos controlados por cápsulas sujeto). Evidentemente este objetivo se cumple al proporcionar una API consistente para el manejo de las cápsulas sujeto, que son las únicas

capaces de controlar dispositivos y de publicar datos.

VI.6. Conclusión

Los resultados obtenidos por la evaluación son un indicativo del grado en el que se cumplen los requerimientos obtenidos en la etapa de análisis, presentada en el capítulo IV. En la evaluación, los usuarios potenciales implementaron una aplicación de control con el objetivo de determinar si es posible crear aplicaciones de control con *hardware* heterogéneo utilizando el *middleware*, así como también el grado de esfuerzo requerido y la utilidad que el *middleware* proporciona.

De acuerdo a los resultados obtenidos, los usuarios potenciales encuentran útil el uso del *middleware* prototipo, además de requerir de un nivel bajo de esfuerzo (es fácil de utilizar). Sin embargo, se encontró que la arquitectura no esconde lo suficiente algunos detalles del emplazamiento de la red, lo que resulta en esfuerzo por parte del desarrollador.

Capítulo VII

Conclusiones

Las redes inalámbricas de sensores ofrecen la posibilidad de monitorear áreas extensas con dispositivos de bajo costo sin contar con infraestructura previa. El uso de actuadores en la red permite modificar el ambiente en el que se encuentran emplazados los nodos. Debido a que los actuadores modifican las variables monitoreadas y los sensores reportan estos cambios, se crea una retroalimentación que se puede utilizar para controlar procesos de manera distribuída. Sin embargo, actualmente las redes WSAN en su mayoría son pequeñas redes de prueba, con un número de nodos reducido y con *hardware* homogéneo. Esta arquitectura de prueba se utiliza debido a varios factores: costo y disponibilidad de los nodos, complejidad de emplazamiento, tiempo disponible para el desarrollo de la aplicación y soporte para los distintos dispositivos que se desea utilizar, entre otros. La investigación en el área de desarrollo de sistemas *middleware* que utilicen los protocolos resultantes de la intensa investigación de bajo nivel, permitiendo el uso de *hardware* heterogéneo y cumpliendo con los requerimientos de las redes WSAN, permitirá en un futuro contar con redes emplazadas en un área amplia, densamente pobladas con nodos sensores y actuadores cooperando para controlar uno o varios procesos distribuídos.

Claramente la necesidad de programar *hardware* heterogéneo representa una problemática para los desarrolladores de aplicaciones que no se encuentran familiarizados con el área. La realización de este trabajo es un paso más hacia la consolidación de una arquitectura agnóstica al *hardware*, permitiendo que los desarrolladores se preocupen por la lógica de la aplicación (el controlador) y no por la manera en la que interactúan los distintos dispositivos que forman la red. La finalidad de la investigación futura en el área debe ser la creación de primitivas de programación que permitan el crecimiento masivo de las

redes WSAN y ofrezcan soporte para la diversidad de componentes presente en dichas redes.

El desarrollo del *middleware* prototipo y la evaluación del mismo proporcionó información sobre la viabilidad de la arquitectura propuesta y la aceptación de dicha tecnología por parte de usuarios potenciales. Los usuarios fueron capaces de programar una sencilla aplicación de control en alto nivel. Los resultados de la evaluación muestran intención de uso, ya que los usuarios manifiestan que el uso del *middleware* requiere de bajo nivel de esfuerzo y que aumenta su productividad al utilizarlo. Sin embargo, la evaluación muestra que, según la percepción de los usuarios, el nivel de abstracción logrado por el *middleware* no los aísla de los detalles de bajo nivel lo suficiente, lo cual se considera para trabajo futuro. Además de la evaluación, se utilizó el *middleware* en una aplicación de monitoreo ambiental que usa una red WSN como fuente de datos. Al trabajar con otros desarrolladores, se obtuvo retroalimentación sobre el proceso de integración del *middleware* con un sistema existente y sobre la compatibilidad de datos entre sistemas, misma que fue integrada a la arquitectura durante el proceso de desarrollo. La integración con aplicaciones existentes y la compatibilidad de datos entre sistemas es un requerimiento primordial, debido al potencial crecimiento a gran escala de las redes WSAN y su posible interacción con sistemas como aplicaciones que utilizan a Internet.

VII.1. Aportaciones

La aportación principal de este trabajo es el desarrollo de una arquitectura para redes WSAN, que permite la programación de aplicaciones en alto nivel y la integración de *hardware* heterogéneo en tiempo de ejecución. El problema de la integración de *hardware* heterogéneo ha sido abordado principalmente por Aberer *et al.* (Aberer *et al.*, 2006), sin embargo su investigación se centra sobre los requerimientos de redes WSN, además de contar con una interfaz de servicios Web que impone el paradigma de diseño orientado

a servicios (SOA por las siglas en Inglés de *Service Oriented Architecture*). La arquitectura presentada en este trabajo se creó con los requerimientos específicos de las redes WSAN, al tiempo que es flexible con el formato de intercambio de datos utilizado, sin imponer un paradigma de diseño específico en el controlador.

Otra aportación es la implementación de un *middleware* basado en la arquitectura propuesta, llamado ActiveCloud. Este sistema se encuentra en un nivel de desarrollo en el que se puede utilizar para extender el trabajo aquí presentado. También, mediante el uso del componente gráfico ActiveCloud GUI es posible el desarrollo de nuevas cápsulas sujeto y observador, probando cada uno de los componentes de manera independiente antes de integrarlos a un sistema completo.

Por último, se propuso una abstracción de acceso al *hardware* que compone a la red WSAN. Esta abstracción se logra a través de módulos autocontenidos llamados «cápsulas» los cuales poseen una interfaz bien definida. Lo anterior permite una separación de responsabilidades clara, lo cual lleva a sistemas reutilizables. Las cápsulas creadas para el *middleware* ActiveCloud se pueden utilizar en otros sistemas con relativa facilidad.

VII.2. Trabajo futuro

Aún existe mucho trabajo por realizar, tanto en el desarrollo de arquitecturas para sistemas *middleware* enfocadas a redes WSAN como en el desarrollo de aplicaciones utilizando los sistemas existentes. Algunas de las oportunidades de mejora y trabajo futuro identificados en la realización de este trabajo son:

- Modificar la arquitectura para integrar una estrategia de comunicación sólida entre las cápsulas sujeto y observador. Investigar la factibilidad de realizar una comunicación basada en un «contrato» sin la complejidad de sistemas existentes como el

protocolo simple de acceso a objetos (SOAP por las siglas en Inglés de *Simple Object Access Protocol*). Una posibilidad puede ser el protocolo llamado *Protocol Buffers*[1] creado por Google Inc.

- Agregar la posibilidad de eliminar una cápsula del sistema en tiempo de ejecución.
- Desarrollar nuevas cápsulas sujeto para aumentar el *hardware* soportado por el *middleware*, así como también cápsulas observador para fomentar el intercambio de datos entre sistemas.
- Experimentar con el uso del *middleware* en otros proyectos, con redes WSAN de distintas características y alcances.
- Investigar sobre metodologías de emplazamiento de redes WSAN e integrarlas al sistema *middleware*. Actualmente el emplazamiento y la verificación de la comunicación entre los distintos dispositivos que integran la red es una de las tareas más tardadas y propensas a errores. Contestar a la pregunta ¿Es posible crear un sistema para el emplazamiento de una red WSAN sin necesidad de configurar manualmente la comunicación de cada uno de los componentes que la integran?
- Investigar sobre el nivel de abstracción adecuado en los sistemas *middleware* para redes WSAN. Es necesario establecer abstracciones de alto nivel que faciliten la programación de aplicaciones, sin perder el control proporcionado por la programación de bajo nivel, el cual es necesario para la optimización de la comunicación entre los nodos que forman la red y la estación base, así como también del código que se ejecuta en los nodos.

[1] http://code.google.com/apis/protocolbuffers/

Bibliografía

Aberer, Karl, Hauswirth, Manfred y Salehi, Ali. 2006. A middleware for fast and flexible sensor network deployment. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 1199–1202 p.

Akyildiz, Ian F. y Kasimoglu, Ismail H. 2004. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, (2), 351 – 367 p.

Albin, Stephen T. 2003. *The Art of Software Architecture: Design Methods and Techniques*. John Wiley and Sons.

Allen, Paul, Higgins, Sam, McRae, Paul y Schlamann, Hermann. 2004. *Service Orientation: Winning Strategies and Best Practices*. Cambridge University Press.

ArchRock. 2007. ArchRock Primer Pack. URL `http://www.archrock.com/downloads/datasheet/primerpack_datasheet.pdf`. (Consultado en Agosto de 2007).

Avilés-López, Edgardo y García-Macías, J. Antonio. 2007. Providing Service-Oriented Abstractions for the Wireless Sensor Grid. In *GPC*. 710-715 p.

Avilés, Edgardo. 2006. *Arquitectura Orientada a Servicios para Redes Inalámbricas de Sensores*. Tesis de Maestría, Centro de Investigación Científica y de Educación Superior de Ensenada.

Buonadonna, Phil, Gay, David, Hellerstein, Joseph M., Hong, Wei y Madden, Samuel. 2005. TASK: Sensor Network in a Box. *Proceedings of the Second European Workshop on Wireless Sensor Networks*, 133–144 p.

Crossbow. 2008. eKo Datasheet. URL `http://www.xbow.com/eko/Images/eKo_brochure_Crossbow.pdf`. (Consultado en Julio de 2008).

Culler, David, Dutta, Prabal, Ee, Cheng Tien, Fonseca, Rodrigo, Hui, Jonathan, Levis, Philip, Polastre, Joseph, Shenker, Scott, Stoica, Ion, Tolle, Gilman y Zhao, Jerry. 2005. Towards a sensor network architecture: lowering the waistline. In *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*. USENIX Association, 24–24 p.

Davis, Fred D. 1985. *A Technology Acceptance Model For Empirically Testing New End-User Information Systems: Theory and Results*. PhD in management, Massachusetts Institute of Technology.

Davis, Fred D. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*, (Sept.), 319–339 p.

Deshpande, Amol, Guestrin, Carlos y Madden, Samuel R. 2005. Resource-Aware Wireless Sensor-Actuator Networks. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*.

Dickerson, Robert, Lu, Jiakang, Lu, Jian y Whitehouse, Kamin. 2008. Stream Feeds - An Abstraction for the World Wide Sensor Web. In *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 360–375 p.

Dunkels, Adam, Grönvall, Björn y Voigt, Thiemo. 2004. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*.

Gamma, Erich, Helm, Richard, Johnson, Raph y Vlissides, John. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Professional.

González, Víctor M., Favela, Jesus y Rodríguez, Marcela. 2004. Towards a Methodology to Envision and Evaluate Ubiquitous Computing. In *Proceedings of workshops*. 78–85 p.

Gupta, P. y Kumar, P. 1999. Capacity of wireless networks.

Hauben, Ronda. 1998. From the ARPANET to the Internet. URL `http://www.columbia.edu/~rh120/other/tcpdigest_paper.txt`. (Consultado en Agosto de 2007).

Hill, Jason, Horton, Mike, Kling, Ralph y Krishnamurthy, Lakshman. 2004. The Platforms Enabling Wireless Sensor Networks. *Communications of the ACM*, 47(6):41–46 p.

Karl, H. y Willig, A. 2003. A Short Survey Of Wireless Sensor Networks. Reporte técnico, Technische Universitat Berlin.

Karl, Holger y Willig, Andreas. 2005. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons.

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

Madden, Samuel R., Franklin, Michael J., Hellerstein, Joseph M. y Hong, Wei. 2005. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173 p.

Melodia, Tommaso. 2007. *Communication and Coordination in Wireless Multimedia Sensor and Actor Networks*. Tesis de Doctorado, Georgia Institute of Technology.

Melodia, Tommaso, Pompili, Dario y Akyldiz, Ian F. 2006. A Communication Architecture for Mobile Wireless Sensor and Actor Networks. *Sensor and Ad Hoc Communications and Networks*, 1(28):109–118 p.

Melodia, Tommaso, Pompili, Dario, Gungor, Vehbi C. y Akyildiz, Ian F. 2007. Communication and Coordination in Wireless Sensor and Actor Networks. *IEEE Transactions on Mobile Computing*, 6(10):1116 – 1129 p.

Montestruque, Luis y Lemmon, M.D. 2008. CSOnet: A Metropolitan Scale Wireless Sensor-Actuator Network. *International Workshop on Mobile Device and Urban Sensing (MODUS)*.

OASIS. 2006. XML Schemas. URL `http://xml.coverpages.org/schemas.html`. (Consultado en Agosto de 2008).

OASIS. 2008. OASIS SOA Reference Model TC. URL http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm. (Consultado en Junio de 2008).

Park, Young-Choong, Ahn, Yang-Keun, Choi, Kwang-Soon, Jung, Kwang-Mo y Kim, Seong-Dong. 2007. The Sensor Network based Home Control System: Supporting the Next Generation Home Gateway System for a Ubiquitous Home Environments. *Journal of Ubiquitous Convergence Technology*, 1(1):1 – 5 p.

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

Prinsloo, Jaco M., Schulz, Christian L., Kourie, Derrick G., Theunissen, W. H. Morkel, Strauss, Tinus, Heever, Roelf Van Den y Grobbelaar, Sybrand. 2006. A service oriented architecture for wireless sensor and actor network applications. In *SAICSIT '06: Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*. South African Institute for Computer Scientists and Information Technologists, 145–154 p.

Römer, Kay, Kasten, Oliver y Mattern, Friedemann. 2002. Middleware challenges for wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):59–61 p.

Römer, Kay y Mattern, Friedemann. 2004. The design space of wireless sensor networks. *Wireless Comunications, IEEE*, 11(6):54–61 p.

Sanchez, Juan A., Ruiz, Pedro M. y Stojmenovic, Ivan. 2007. Energy-efficient geographic multicast routing for Sensor and Actuator Networks. *Comput. Commun.*, 30(13):2519–2531 p.

Sgroi, M., Wolisz, A., Sangiovanni-Vincentelli, A. y Rabaey, J.M. 2005. A Service-Based Universal Application Interface for Ad Hoc Wireless Sensor and Actuator Networks.

Staff, National Research Council. 2001. *Embedded Everywhere: A Research Agenda for Networked Systems of Embedded Computers*. National Academy Press.

SUN. 1999. JAR File Specification. URL http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html. (Consultado en Julio de 2008).

Tanenbaum, Andrew S., Gamage, Chandana y Crispo, Bruno. 2006. Taking Sensor Networks from the Lab to the Jungle. *Computer*, 39(8):98–100 p.

Tavakoli, Arsalan, Dutta, Prabal, Jeong, Jaein, Kim, Sukun, Ortiz, Jorge, Culler, David, Levis, Phillip y Shenker, Scott. 2007. A modular sensornet architecture: past, present, and future directions. *SIGBED Rev.*, 4(3):49–54 p.

W3C. 2006. Extensible Markup Language (XML) 1.0 (Fourth Edition). URL http://www.w3.org/TR/2006/REC-xml-20060816/#sec-prolog-dtd. (Consultado en Julio de 2008).

Xia, Feng, Tian, Yu-Chu, Li, Yanjun y Sun, Youxian. 2007. Wireless Sensor/Actuator Network Design for Mobile Control Applications. *Sensors*, (7), 2157 – 2173 p.

Younis, Mohamed y Akkaya, Kemal. 2008. Strategies and techniques for node placement in wireless sensor networks: A survey. *Ad Hoc Netw.*, 6(4):621–655 p.

Apéndice A

Creación de Cápsulas

El objetivo de las cápsulas es hacer extensible la arquitectura, permitiendo el intercambio de implementaciones y el aumento de las capacidades de la red en tiempo de ejecución. La extensibilidad de aplicaciones no es una idea nueva y ha sido implementada con éxito en distintos tipos de *software* desde reproductores multimedia hasta navegadores Web. Para el caso del *middleware* ActiveCloud, las cápsulas permiten aumentar las capacidades del *middleware*, es decir, aumentar el soporte a *hardware* (cápsulas sujeto) y los métodos de entrega de datos generados por la red (cápsulas observador).

A.1. Proceso de creación de una cápsula

Las cápsulas son programas Java con cierto formato establecido por la arquitectura de ActiveCloud. Para crear una nueva cápsula se debe de:

- Determinar si se trata de una cápsula sujeto (ejecuta acciones sobre hardware y/o publica datos) u observador (recibe datos, los procesa y los hace disponibles a otro sistema).
- Crear una nueva clase Java que herede de `PublisherCapsule` o de `ObserverCapsule` para cápsulas sujeto u observador, respectivamente.
- Empaquetar la cápsula junto a su archivo de configuración, icono y bibliotecas de soporte como se detalla en la Sección V.2.1.
- Opcionalmente puede probar que la cápsula funciona correctamente utilizando ActiveCloud GUI.

A.1.1. Ejemplo de creación de cápsula sujeto

La presente sección detalla la creación de la cápsula sujeto «PhidgetOut». Esta cápsula sujeto controla un phidget conectado a una computadora por el puerto USB. El phidget en cuestión es una tarjeta de E/S con 8 salidas y 8 entradas digitales, así como 8 entradas analógicas. La comunicación con el phidget se hace por medio del protocolo *Multicast Domain Name Server* (mDNS), gracias a un servicio Web que se ejecuta en la computadora con el phidget conectado. El ejemplo sólo incluye el control de las 8 salidas digitales.

En primera instancia se tiene que determinar la manera en la que la cápsula se comunica con el *hardware*. Esto varía dependiendo de la API disponible para el *hardware* en cuestión, siendo esta complejidad la que se pretende eliminar al encapsular su comunicación para su uso con ActiveCloud. Para el caso del phidget, toda la comunicación se hace por medio de una biblioteca Java proporcionada por los fabricantes. Esta biblioteca utiliza la interfaz nativa de Java (JNI por las siglas en Inglés de *Java Native Interface*) para comunicarse con un manejador nativo al sistema operativo. Es este manejador el que se comunica con el *hardware* directamente.

Antes de crear una cápsula sujeto para dicho phidget, se necesita conocer la manera «tradicional» de programar aplicaciones que utilicen el phidget. Inicialmente se crea una implementación de la clase de la API que permite la comunicación con el phidget, en este caso la clase es `InterfaceKitPhidget()` (En el Listado A.1 se muestran los métodos de ésta clase). Esta clase envía mensajes para ciertos eventos de interés. Para ser notificado de dichos eventos, una clase se registra como «escucha» del evento, de la misma manera en que un programa Swing[1] se registra para el evento generado cuando un botón se presiona. Una vez configurados los eventos, se establece una conexión con el *hardware* y en este momento se puede utilizar cualquier método de la implementación de `InterfaceKitPhidget()`

[1]Swing es un conjunto de clases para la creación de interfaces gráficas en Java. http://java.sun.com/docs/books/tutorial/uiswing/

para controlar el *hardware*. En el Listado A.2 se muestra el método principal de un programa que implementa lo descrito anteriormente. Cuando el estado de una salida cambia, el programa imprime a la salida estándar el número de la salida afectada y el nuevo estado de la salida.

Listado de código fuente A.1: **Métodos de la clase InterfaceKitPhidget InterfaceKit**

```
1  public void addInputChangeListener(InputChangeListener l);
2  public void addOutputChangeListener(OutputChangeListener l);
3  public void addSensorChangeListener(SensorChangeListener l);
4  public int getInputCount();
5  public boolean getInputState(int index);
6  public int getOutputCount();
7  public boolean getOutputState(int index);
8  public boolean getRatiometric();
9  public int getSensorChangeTrigger(int index);
10 public int getSensorCount();
11 public int getSensorRawValue(int index);
12 public int getSensorValue(int index);
13 public void removeInputChangeListener(InputChangeListener l);
14 public void removeOutputChangeListener(OutputChangeListener l);
15 public void removeSensorChangeListener(SensorChangeListener l);
16 public void setOutputState(int index, boolean newVal);
17 public void setRatiometric(boolean state);
18 public void setSensorChangeTrigger(int index, int newVal);
```

Listado de código fuente A.2: **Programa que cambia las salidas del phidget InterfaceKit**

```
1  public static final void main(String args[]) throws Exception {
2          InterfaceKitPhidget ik;
3          ik = new InterfaceKitPhidget();
4
5          ik.addOutputChangeListener(new OutputChangeListener() {
6                  public void outputChanged(OutputChangeEvent oe) {
7                          System.out.println("Salida #"+oe.getIndex());
8                          System.out.println("Nuevo valor: "+oe.getState());
9                  }
10         });
11
12         ik.openAny();
13         ik.waitForAttachment();
14
15         boolean o[] = new boolean[ik.getOutputCount()];
16         for (int i = 0; i < ik.getOutputCount(); i++)
17                 o[i] = ik.getOutputState(i);
18         for (int i = 0; i < 100000; i++) {
19                 int n = (int)(Math.random() * ik.getOutputCount());
20                 ik.setOutputState(n, !o[n]);
21                 o[n] = !o[n];
```

```
22                  try {
23                          Thread.sleep(1);
24                  } catch (Exception e) {}
25          }
26          ik.close();
27  }
```

Inicialización

Una vez identificados los requerimientos de la API del *hardware*, es posible comenzar a programar la cápsula. Al cargar una cápsula sujeto u observador, el *middleware* ejecuta el método `init()`. Este método, como su nombre lo indica, se utiliza para inicializar las variables de la cápsula. El método `init()` recibe como parámetros una cadena formato XML que contiene la configuración inicial. Las clases padre `PublisherCapsule` y `ObserverCapsule` implementan este método de manera genérica, es decir, no es necesario que las clases hijo lo sobreescriban para la correcta inicialización de la cápsula. Sin embargo, típicamente `init()` es sobreescrito para realizar tareas de inicialización propias del *hardware* o de la API utilizada.

En el Listado A.3 se muestra el método `init()` de la clase `PhidgetOut`. En este método, además de realizar la inicialización por defecto (en la línea `super.init(e)`), se crea la instancia de la API para el control del phidget (`ifk = new InterfaceKitPhidget()`) y se configuran los eventos (*callback*) utilizados por la API para propósitos de notificación. Los eventos que soporta la API del phidget son:

- Conexión de un cliente al manejador de la tarjeta (`addAttachListener()`).
- Desconexión de un cliente al manejador de la tarjeta (`addDetachListener()`).
- Error (`addErrorListener()`).
- Cambio de estado de una entrada digital (`addInputChangeListener()`).
- Cambio de estado de una salida digital (`addOutputChangeListener()`).

- Cambio de estado de una entrada analógica (`addSensorChangeListener()`).

Listado de código fuente A.3: **Método init() sobreescrito de la cápsula PhidgetOut**

```
1  @Override
2  public void init(Element e) throws CapsuleInitException{
3          super.init(e);
4
5          try{
6                  ifk = new InterfaceKitPhidget();
7                  ifk.addAttachListener(this);
8                  ifk.addDetachListener(this);
9                  ifk.addErrorListener(this);
10                 ifk.addInputChangeListener(this);
11                 ifk.addOutputChangeListener(this);
12                 ifk.addSensorChangeListener(this);
13         }catch(PhidgetException pe){
14                 throw new CapsuleInitException(pe.getMessage(),this.getClass().getCanonicalName());
15         }
16 }
```

Comunicación con el *hardware*

La clase `PublisherCapsule` provee dos métodos abstractos: `start()` y `stop()` para iniciar y detener la comunicación con el *hardware* respectivamente. Siendo métodos abstractos, éstos los deben implementar cualquier clase que herede de `PublisherCapsule`. En el Listado A.4 se muestra la implementación de los métodos `start()` y `stop()` para la clase `PhidgetOut`. En el método `start()` se intenta establecer la comunicación con el phidget utilizando algunos parámetros preestablecidos, en caso de que fallar intenta utilizar un método genérico. En el método `stop()` simplemente se intenta cerrar la conexión. Como se puede observar, el código que contienen los dos métodos utilizan la API provista por el *hardware*.

Listado de código fuente A.4: **Métodos start() y stop()**

```
1  public void start() throws StartException {
2          try{
3                  ifk.open(PhidgetOut.serial,null);
4                  ifk.waitForAttachment();
5                  return;
6          }catch(PhidgetException pe){
7                  log.error(pe.getDescription());
8          }catch(NumberFormatException nfe){
9                  log.error(nfe.getMessage());
10         }
11
12         try{
13                 ifk.openAny();
14         }catch(PhidgetException pe){
15                 log.error(pe.getMessage());
16         }
17 }
18
19 public void stop() throws StopException {
20         try{
21                 ifk.close();
22         }catch(PhidgetException pe){
23                 throw new StopException(pe.getMessage());
24         }
25 }
```

Acciones

Las cápsulas sujeto proveen acciones para la manipulación del *hardware*. En el caso de la cápsula para el phidget, solo es necesaria una acción: el control de las 8 salidas digitales. Como se menciona en la Sección V.2.1, las acciones son métodos implementados por las cápsulas sujeto y soportados por el *hardware* al que controla la cápsula sujeto. Una acción es un objeto que hereda de la clase abstracta `Action` y por lo tanto tiene una descripción, nombre y valores de configuración. Por último, el objeto debe implementar el método abstracto `run()` que se invoca cuando se ejecuta la acción. Esto se ilustra en la Figura 32. Se utiliza el método `addAction()` para agregar nuevas acciones, como se muestra en el Listado A.5.

Listado de código fuente A.5: **Acción Out.Switch de la cápsula PhidgetOut**

```
addAction(new Action("Out.Switch","Switch a digital output"){
        public void run(Element e){
                setVariables(e);
                boolean outState = Boolean.parseBoolean(get("outstate"));
                int index = Integer.parseInt(get("index"));

                try{
                        ifk.setOutputState(index, outState);
                }catch(PhidgetException pe){
                        log.error(pe.getMessage());
                }
        }
});
```

Típicamente (más no necesariamente) las acciones se agregan de manera anónima en el método `init()`, precisamente como se muestra en el Listado A.5. La acción utiliza la instancia de `InterfaceKitPhidget` creada en el método `init()` y ejecuta su método `setOutputState()` para modificar el estado de la salida indicada por la variable `index`. Las acciones, al igual que el método `init()` de las cápsulas, reciben como parámetro un nodo XML con valores de configuración. Para el caso de la acción `Out.Switch`, se esperan 2 valores: `outState` e `index`.

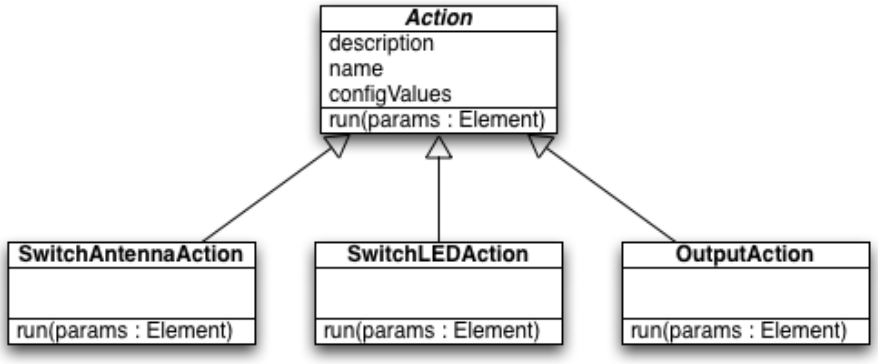


Figura 32: **Diagrama UML de clases para la clase abstracta `Action` y la implementación de las acciones que la cápsula «PhidgetRFID» proporciona**

Publicación de datos

Por último, una cápsula sujeto debe publicar los datos que genera. Como se menciona en la Sección V.2.3, el servicio de notificación se utiliza para este fin. La API del phidget ejecuta el método `outputChanged()` cuando cambia el estado de una salida, lo que representa un dato generado por la red y por lo tanto es conveniente que se publique. En el Listado A.6 se muestra la utilización del servicio de notificación para publicar los datos generados.

Listado de código fuente A.6: **Utilización del servicio de notificación para publicar datos**

```
1 public void outputChanged(OutputChangeEvent arg0) {
2         Element payload = createPayload(arg0);
3         NotificationService.get().publish(payload);
4 }
```

Se utiliza el método `publish()` del servicio de notificación para publicar datos. Este método recibe como parámetro un nodo XML que contiene los datos generados. El nodo XML generado por `createPayload()` se muestra en el Listado A.7.

Listado de código fuente A.7: **Formato de datos enviados por el servicio de notificación**

```
1 <payload class="com.ubisoa.activecloud.PhidgetOut">
2         <entry name="OutputChanged" value="True" />
3 </payload>
```

Una vez que se programa la cápsula, se empaqueta de acuerdo a lo estipulado en la Sección V.2.1 y está lista para utilizarse. Para el caso del presente ejemplo, la cápsula generada es capaz de manipular las salidas del phidget a través de la red, por medio del protocolo mDNS. Los usuarios de la cápsula no necesitan conocer los detalles del funcionamiento de la API del phidget, solo llamar inicialmente al método `start()` y comenzar a ejecutar las acciones de interés, procedimiento que es el mismo para cualquier *hardware* que cuente con una cápsula sujeto.

A.1.2. Ejemplo de creación de cápsula observador

La presente sección detalla la creación de la cápsula observador «PrintXMLCapsule». Esta cápsula recibe los datos generados por las cápsulas sujeto a las que ha sido asignada como observador e imprime estos datos a la salida estándar (muy útil para propósitos de depuración).

Inicialización

Al igual que las cápsulas sujeto, cuando se carga una cápsula observador se ejecuta su método `init()` con el propósito de inicializar las variables de la cápsula. El procedimiento es el mismo que el indicado en la Sección A.1.1. Sin embargo, debido a que la cápsula es muy sencilla, no es necesario (en este caso) sobreescribir el método `init()` ya que la implementación por defecto es suficiente.

Control de comunicación

Las cápsulas observador también deben implementar los métodos abstractos `start()` y `stop()`. Su propósito es muy similar al de las cápsulas sujeto: iniciar y detener la comunicación con algún dispositivo o sistema, en caso de ser necesario. Para el caso de la cápsula «PrintXMLCapsule» esto no es necesario por lo tanto se dejan vacíos. Algunos casos en los que si se necesita implementar estos métodos son:

- Modificar un archivo en un servidor remoto (por medio de protocolos como SSH, FTP, HTTP, etc).
- Enviar un mensaje SMS por medio de una red celular.
- Enviar un correo electrónico.
- Agregar una entrada a una base de datos.

Recepción de datos

Las cápsulas observador deben implementar el método abstracto `receive()`. Este método lo ejecuta el servicio de notificación cuando existen datos publicados por alguna cápsula sujeto a la que la cápsula observador se ha registrado. El método `receive()` recibe como parámetro los datos en formato XML enviados por la cápsula sujeto (ver Sección A.1.1). En el Listado A.8 se muestra la implementación de la cápsula «PrintXMLCapsule».

Listado de código fuente A.8: **Implementación de la cápsula PrintXMLCapsule**

```
1  public class PrintXMLCapsule extends ObserverCapsule{
2          private static Logger log = Logger.getLogger(PrintXMLCapsule.class);
3
4          public void receive(Element payload) throws ReceiveException {
5                  XMLOutputter out = new XMLOutputter(Format.getPrettyFormat());
6                  log.debug(out.outputString(payload));
7          }
8
9          public void start() throws StartException {}
10
11         public void stop() throws StopException {}
12 }
```

Al igual que las cápsulas sujeto, las cápsulas observador deben ser empaquetadas de acuerdo a lo estipulado en la Sección V.2.1.

A.2. Conclusión

Como se puede deducir de los ejemplos presentados, generalmente es más complicada la programación de cápsulas sujeto que la programación de cápsulas observador. Es necesario conocer la manera en la que se programan aplicaciones para dicho *hardware*, lo cual varía entre dispositivos. Esta problemática está presente a los desarrolladores de aplicaciones para redes WSAN, lo cual se oculta por las cápsulas.

Las cápsulas observador pueden tener varios usos: desde guardar la información para

consultarse posteriormente, hasta presentarla en distintas maneras (por ejemplo gráficas), enviarla a otros sistemas o transformarla a distintos formatos por motivos de compatibilidad. La complejidad de las cápsulas observador está dada por el trabajo a realizar.

Apéndice B

Cuestionario de Evaluación

1. El API de ActiveCloud es suficiente para el desarrollo de aplicaciones de control con WSAN

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

2. El ambiente publish/subscribe es adecuado para el desarrollo de aplicaciones de control con WSAN

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

3. ActiveCloud me oculta los detalles de bajo nivel para el desarrollo de aplicaciones con WSAN

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

4. Considero que la API de ActiveCloud es coherente y permite el manejo de todos los dispositivos que integran la red WSAN

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

5. Considero posible la integración de los datos generados por las cápsulas sujeto con otros sistemas

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

6. ActiveCloud me permite la utilización de hardware heteroéneo

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

7. ActiveCloud me facilita la obtención de datos de múltiples Fuentes

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

8. Considero que ActiveCloud puede soportar hardware que aún no se encuentra disponible (mediante la creación de nuevas cápsulas sujeto)

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

9. Considero que es conveniente utilizar ActiveCloud para el desarrollo de aplicaciones de control con WSAN

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

10. Encuentro fácil el uso de ActiveCloud

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

11. Considero que ActiveCloud puede ayudarme en el desarrollo de aplicaciones de control con WSAN

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

12. En general, encuentro útil el middleware ActiveCloud

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

13. Considero que no tendría mayor problema en la adopción de ActiveCloud

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

Perfil personal

¿Cuál es su experiencia en el desarrollo de aplicaciones?

¿Con cuales lenguajes de programación tiene experiencia?

¿Con cuales ambientes de desarrollo (IDEs) tiene experiencia?

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

¿Qué tan familiarizado se encuentra con los patrones de diseño de software utilizados en ActiveCloud?

¿Qué tan familiarizado se encuentra con el área de las redes inalámbricas de sensores y actuadores?

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

Comentarios y sugerencias en cuanto a las características de ActiveCloud:

Comentarios sobre el proceso de evaluación:

Apéndice C

Evaluación Sobre el Modelo de Comunicación

C.1. Escenario

A continuación se presenta un escenario para el cual se necesita desarrollar una aplicación de control utilizando una WSAN. Detalle las cápsulas necesarias para desarrollar dicha aplicación con el middleware ActiveCloud, así como también la interacción entre ellas. Se tiene una casa inteligente y se desea instalar un sistema de monitoreo de intrusos. La casa cuenta con sensores de detección de presencia instalados estratégicamente, además de contar con cámaras con la capacidad de reportar video en vivo desde el lugar de los hechos. Se desea una aplicación de control que cuando el sistema detecte un intruso, avise a la policía via telefónica, además de comenzar a grabar video del lugar donde se detectadó la intrusión.

C.1.1. Preguntas

- ¿Qué cápsulas sujeto son necesarias?
- ¿Qué cápsulas observador son necesarias?
- ¿Cuál es la interacción entre ellas?

Apéndice D

Especificación del Archivo de Configuración

El archivo de configuración de cápsulas se rige bajo las especificaciones de un esquema XML (*XML Schema*). Existen varios lenguajes para la especificación de esquemas XML (OASIS, 2006). Algunos de estos lenguajes son:

- Lenguaje de Marcado de Definición de Documento (DDML por las siglas en Inglés de *Document Definition Markup Language*).
- Lenguaje de Definición de Esquema de Document (DSDL por las siglas en Inglés de *Document Schema Definition Language*)
- Descripción de Estructura de Documento (DSD por las siglas en Inglés de *Document Structure Description*)
- Definición de Tipo de Documento (DTD por las siglas en Inglés de *Document Type Definition*)
- Lenguaje de Enrutamiento de Espacio de Nombres (NRL por las siglas en Inglés de *Namespace Routing Language*)
- Esquema XML de la W3C (WXS o XSD)

En el Listado D.1 se muestra el esquema para el archivo de configuración de cápsulas utilizando el formato Esquema XML de la W3C.

Listado de código fuente D.1: **Esquema del archivo de configuración**

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2
3  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
4    <xs:element name="capsule">
5      <xs:complexType>
6        <xs:sequence>
7          <xs:element ref="subject" />
8          <xs:element ref="observer" />
9          <xs:element ref="config" />
10       </xs:sequence>
11     </xs:complexType>
12   </xs:element>
13
14   <xs:element name="subject">
15     <xs:complexType>
16       <xs:attribute name="class" type="xs:NMTOKEN" use="required" />
17     </xs:complexType>
18   </xs:element>
19
20   <xs:element name="observer">
21     <xs:complexType>
22       <xs:attribute name="class" type="xs:NMTOKEN" use="required" />
23     </xs:complexType>
24   </xs:element>
25
26   <xs:element name="config">
27     <xs:complexType>
28       <xs:sequence>
29         <xs:element ref="key" maxOccurs="unbounded" />
30       </xs:sequence>
31     </xs:complexType>
32   </xs:element>
33
34   <xs:element name="key">
35     <xs:complexType>
36       <xs:attribute name="name" type="xs:NMTOKEN" use="required" />
37       <xs:attribute name="value" type="xs:string" use="required" />
38     </xs:complexType>
39   </xs:element>
40 </xs:schema>
```