Proyecto Fin de Máster

MODELADO PARA SIMULACIÓN DE REDES DE SENSORES INALÁMBRICAS PREDESPLIEGUE BASADO EN *VISUALSENSE*

Víctor Julián Roselló Gómez-Lobo

Máster en Electrónica Industrial

Universidad Politécnica de Madrid

Centro de Electrónica Industrial

Escuela Técnica Superior de Ingenieros Industriales

Departamento de Automática, Ingeniería Electrónica e Informática Industrial

Septiembre, 2009

Universidad Politécnica de Madrid

Centro de Electrónica Industrial

Escuela Técnica Superior de Ingenieros Industriales

Departamento de Automática, Ingeniería Electrónica e Informática Industrial

Máster en Electrónica Industrial

MODELADO PARA SIMULACIÓN DE REDES DE SENSORES INALÁMBRICAS PREDESPLIEGUE BASADO EN *VISUALSENSE*

Autor: Víctor Julián Roselló Gómez-Lobo

Director: Teresa Riesgo Alcaide

Codirector: Jorge Portilla Berrueco

Septiembre, 2009

Proyecto Fin de Máster

Índice

Introducción.

Las redes de sensores inalámbricas (en inglés *Wireless Sensor Networks*, WSNs), son unas de las tecnologías de control y monitorización con más crecimiento de los últimos años. Según los editores del *MIT Technology Review* están destinadas a ser "*una de las 10 tecnologías que cambiaran el mundo*" [1].

El porqué del fuerte desarrollo tecnológico en este campo, es son duda la flexibilidad de estos sistemas y el bajo impacto causado el medio que les permiten adaptarse a casi cualquier tipo de entorno y aplicación. Algunos ejemplos destacables de aplicaciones para WSNs son:

- *Aplicaciones militares [2]:* monitorización de fuerzas y equipos, vigilancia del campo de batalla, reconocimiento del terreno, detección de ataques biológicos, químicos o nucleares, etc.
- *Aplicaciones medioambientales [3]:* seguimiento de animales, monitorización de las condiciones ambientales en cultivos, riego, agricultura de precisión, detección de incendios forestales, detección de inundaciones, estudios de contaminación, prevención de desastres, monitorización de áreas afectadas por desastres, etc.
- *Aplicaciones médicas [4]:* telemonitorización de datos fisiológicos en pacientes, diagnóstico, administración de medicamentos, seguimiento de médicos y pacientes en hospitales, etc.
- *Aplicaciones en el hogar/edificios [5]:* uso económico de calefacciones, aires acondicionados y control de iluminación en edificios inteligentes, ayuda a la evacuación de personas en caso de incendios, reconocimiento del estado de estructuras para controlar su deterioro debido a, por ejemplo, vibraciones, así como control de electrodomésticos, control y optimización del consumo energético en el hogar, etc.
- *Aplicaciones industriales [6]:* seguimiento de vehículos, control de flota, control de inventarios, control de calidad de emisión de residuos al medioambiente.

Para comprender por qué este tipo de sistemas permiten aplicaciones tan heterogéneas es conveniente presentar las principales características de los mismos en general y de la plataforma para WSNs desarrollada en el Centro de Electrónica Industrial de la Universidad Politécnica Madrid, denominada *Cookies*.

I. Redes de sensores inalámbricas.

Las redes de sensores inalámbricas son sistemas formados por nodos dedicados a tomar medidas y/o actuar sobre entornos heterogéneos. Además, tienen la capacidad de cambiar su topología de forma dinámica lo que les permite adaptarse a posibles cambios en el entorno llegando incluso a tener nodos móviles que se puedan desplazar por el área de despliegue de la red.

Se pueden diferenciar los componentes de un nodo en 4 principales bloques funcionales, que se corresponde a la división clásica en sistemas digitales:

- *Sensado:* Se encarga de la adquisición de medidas del entorno. Está formado por los sensores encargados de medir los fenómenos o magnitudes físicas de interes así como los circuitos de acondicionamiento de señal que fueran necesarios.
- *Comunicaciones*: Para envío de los datos alamcenados, enrutamiento de mensajes para otros nodos y recibir mensajes de la red.
- *Alimentación:* Se encarga de porporcionar energía al nodo, ya sea con baterías desechables, mediante generacion de energías renovables, por ejemplo, paneles solares y bien conectadonlos a la red electrica.
- *Procesamiento:* Es la unidad central del nodo, encargada de controlar el envío de mensajes y procesamieto de las medidas de los sensores y la información recibida desde la red.

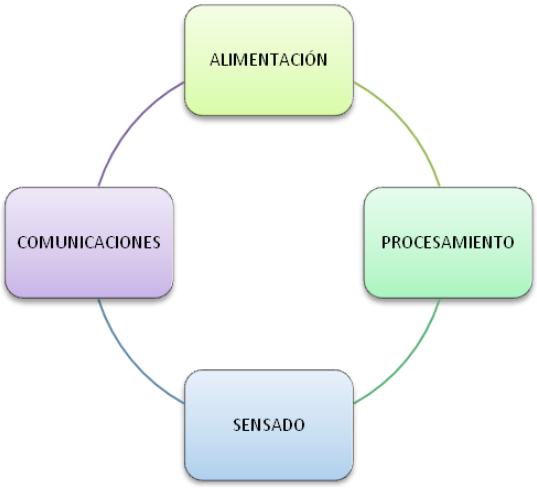


Figura 1: Bloques Funcionales de un nodo para WSNs.

Según la funcionlidad que tiene un nodo se pueden distinguir 3 tipos de nodos.

- *Router:* también denomindos FFDs (*Full Funcitonal Devices*) Son nodos con capacidad de rutar mensajes por la red, por ello la radio de este tipo de nodos debe estar en funcionamiento durante todo el tiempo, o al menos una gran parte de él. Esto último sólo es posible para redes pequeñas en las que el tráfico generado esta perfectamente acotado en el tiempo. Puede, además, tener capacidad de sensado o no.
- *Coordinador:* Es el nodo que generalmente hace las funciones de pasarela, ya sea con otra red, con una unidad central de procesado y en general con otro medio. Es el que crea la red y controla el acceso de nuevos nodos a ella.
- *Sensor:* también denomindos RFDs (*Reduced Functional Devices*). Este tipo de nodos sólo tienen capacidad de sensado y envío de mensajes a un *Router*, pero no pueden formar parte de caminos de datos para otros nodos de la red (aunque sí pueden recibir mensjes de los que sean destinatarios). La mayor parte del tiempo estan en modo durmiente, para reducir así su consumo, despertándose periódicamente para realizar las táreas que le correspondan y comunicarse con su nodo padre (el *Router* del que dependen para poder comunicarse con el resto de nodos de la red).

Una de las características más importantes de las WSNs es la capacidad de los nodos para procesar y gestionar el tráfico en la red y tomar decisiones acerca del procesamiento de los datos que reciben. Por ejemplo, en un momento dado si un nodo esta sobrecargado de tráfico, podrían aparecer rutas nuevas en la red para entregar la información al destino que no inlcluyeran a este nodo, o bien un nodo puede tomar decisiones en funcion de los datos recogidos y si estos fueran de poca relevancia, no generar tráfico innecesario con el consiguiente ahorro de baterías y evitando posibles problemas de saturación en la red.

Para que esta capacidad de cambiar o reparar rutas entre nodos sea eficiente es necesario elegir una topología de red adecuada. Existen varios tipos de topologías utilizadas en WSN, todas ellas conocidas (maya, arbol, *cluster*, etc). En nuestro caso nos centraremos solamente en la redes tipo maya (ver Figura 2), ya que esta topología es la utlizada en *ZigBee*PRO, protocolo utilizado en la capa de comunicaciones plataforma *Cookies*, que sera presentada posteriormente.

En este tipo de topología sólo existe relación jerárquica entre el nodo sensor y el router del que depende, pudiendo cambiar en todo momento los nodos intermedios por los que pasan los datos para llegar de un punto a otro.

Esto les confiere una gran capacidad de autonomía ya que, en teoría, si una red esta bien planificada, no tendrían por que existir elementos críticos en la red ya que exitirían nodos en el área de funcionamiento del nodo que ha fallado que podrían suplirlo.

Además, al tratarse de redes inalámbricas, el coste de despliegue es mucho menor que el de una red convencional, a parte de ser mucho mas flexible y menos intrusivas que estas.

Todas estas características unidas a la reducción del consumo de los dispositivos, la creación de protocolos de comunicaciones adaptados a estos sistemas (*ZigBee* [7], Lowpower WiFi [8], etc.), así como las nuevas tecnologías de sensores y actuadores (MEMS), han hecho que las WSNs se conviertan en una importante alternativa para la adquisición y control en multitud de entornos y aplicaciones.

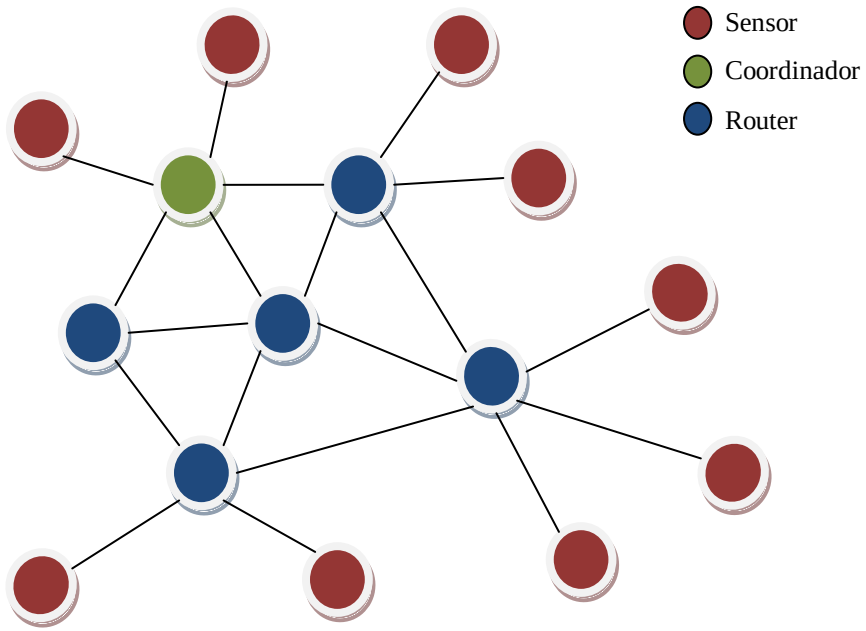


Figura 2: Topología de WSN en malla.

II. COOKIES: Plataforma modular para redes de sensores.

En el Centro de Electrónica Industrial de la Universidad Politécnica de Madrid se ha desarrollado una plataforma *hardware* para redes de sensores inalámbricas. Esta plataforma añade a todas las características presentadas anteriormente la modularidad [9].

Esta modularidad se traduce en un diseño por capas todas ellas con una interfaz común que permiten que la plataforma se pueda adaptar a cualquier tipo de aplicación optimizando al máximo su rendimiento. También le confiere la capacidad de evolucionar con la tecnología, permitiendo así que se adapte rápidamente a, nuevos protocolos de comunicaciones, sensores, microcontroladores de mejores prestaciones, etc.

Estas capas están separadas según los bloques funcionales de la Figura 1. Es decir, exenten 4 tipos de capas diferentes:

- **Procesamiento:** La característica de la plataforma para procesamiento principal es que ofrece la posibilidad de utilizar una FPGA (*Field Programmable Gate Array)* para procesamiento de señales digitales. Además cuenta con un microcontrolador encargado de controlar las comunicaciones y el procesamiento de señales analógicas.
- **Comunicaciones**: Actualmente existen dos tipos de capas de comunicaciones para *Bluetooth* y *ZigBee*.
- **Alimentación**: Ofrece diversos modos de alimentación de los nodos. Como por ejemplo, baterías, USB, panel solar o red eléctrica.
- **Sensado:** Aceleración, temperatura, humedad, luz, IR, deformación. Tanto sensores con interfaz analógico como digital.

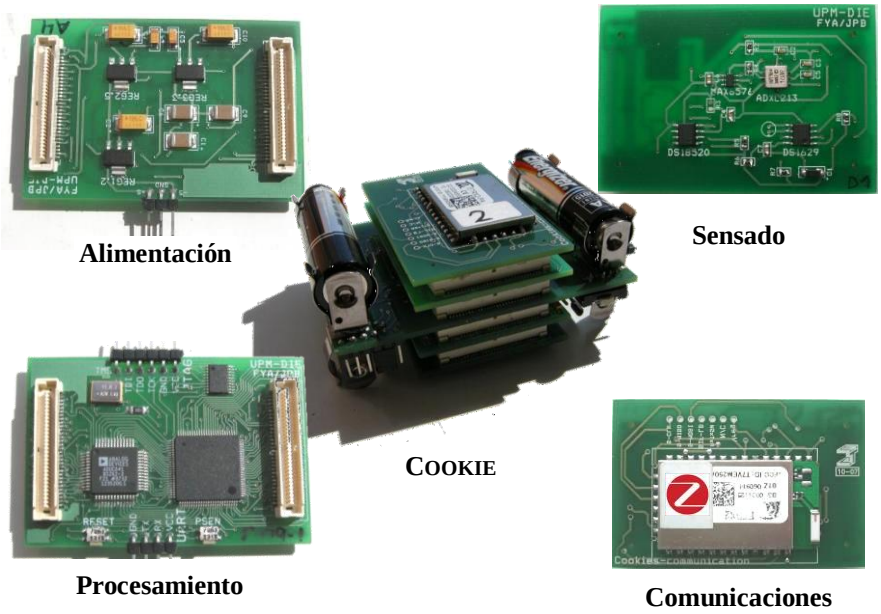


Figura 3: Plataforma de WSN y detalle de las capas.

III. Justificación del proyecto.

Si recapitulamos las características expuestas vemos que las WSNs conforman un sistema basado en nodos, desde unas pocas unidades a incluso cientos o miles y que esta cantidad puede incrementarse o decrementarse dinámicamente sin que el funcionamiento del sistema se vea afectado. Una WSN puede cubrir grandes áreas, en entornos heterogéneos todo ello de una forma desentendida y sostenida en el tiempo (idealmente durante meses o años).

Además si vemos alguno de los ejemplos de aplicación, por ejemplo [3], donde se distribuirán un centenar de nodos por una ciudad para medir las emisiones de gases contaminantes se puede llegar a la conclusión que un mal despliegue de la red puede resultar dramático para el rendimiento y la viabilidad de la red.

Todo eso hace que sea imprescindible poder conocer el comportamiento de la red antes de desplegarla en el entorno real, ya que una vez que la red ha sido instalada puede resultar complicado y costoso acceder a los nodos para realizar cambios por fallos que no fueron previstos, como por ejemplo, nodos con tráfico excesivo, partes de la red aislados por obstáculos, o simplemente que los días de tormenta los nodos no se comunicasen por que se reducía demasiado el alcance de la radio.

Por ello se hace imprescindible poder realizar simulaciones predespliegue de las WSNs, y será preciso contar con una herramienta capaz de modelar y simular el tanto el comportamiento del nodo como el comportamiento de la red, todo ello de forma concurrente.

Como ya se ha planteado anteriormente las WSN presentan unas características muy especiales por lo que se hace necesario elegir una herramienta adecuada para este tipo de plataforma que vaya más allá de un simulador de redes convencional, ya que el modelado de WSN requiere no sólo el modelado de canales de comunicación, y protocolos de red, si no también, canales de sensado, consumo de energía, mecanismos de posicionamiento.

En resumen es necesario poder modelar tanto a nivel del nodo, es decir el *hardware* que lo compone y las funciones que esta realiza y como a nivel de sistema, es decir modelado de las comunicaciones entre nodos, creación de rutas de datos y efectos del medio en el despliegue y las comunicaciones.

IV. Organización del documento.

El documento está organizado siguiendo las etapas en las que se realizó el trabajo presentado, empezando con la definición del objetivo del proyecto.

Primero se realizó una búsqueda de la plataforma más apropiada para la realización de este proyecto partiendo de unos requisitos mínimos deseados, terminando con la elección de la plataforma para modelado y simulación de redes de sensores *VisualSense*.

Posteriormente se presentara la metodología de diseño elegida y se justificará comparándola con otras soluciones en el estado de la técnica.

A continuación se describirá las interfaces de diseño y los modelos creados para cada una de las capas de la plataforma (Fuente de Alimentación, Sensado, Procesamiento y Comunicaciones.

El modelo de la capa de comunicaciones será tratado de una forma más profunda por tratarse del que más complejidad conlleva prestando especial atención al protocolo de rutado utilizado y el control de acceso al medio, que serán descritos en el Capítulo 6:.

Finalmente se presentarán los resultados obtenidos con las conclusiones y las líneas futuras abiertas tras la finalización de este trabajo.

Objetivos.

En este proyecto fin de máster se desarrolla un modelo de simulación de la plataforma *Cookies* y se define una interfaz de diseño que permita reflejar la principal característica diferencial de esta plataforma, la modularidad.

Para ello se propone una estructura basada en 4 submodelos independientes, uno por cada una de las capas de la plataforma, definidos con máquinas de estados o FSM (*Finite State Machine)*.

Para cada una de las capas se crean varios modelos para probar que se cumple con la condición de que las todas las funcionalidades del nodo sean independientes entre sí, manteniendo así la modularidad característica de la plataforma Cookies.

El primer paso es analizar las plataformas disponibles para modelado y simulación de WSNs, fijando previamente las características y requisitos mínimos exigidos.

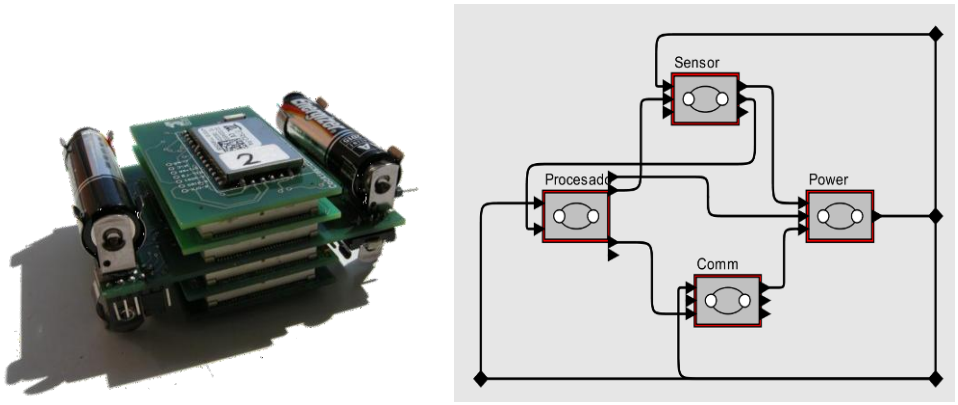


Figura 4: Plataforma hardware y Modelo propuesto.

Capítulo 1: Elección del Simulador.

Actualmente existen multitud de simuladores validos para WSNs, muchos de ellos con grandes restricciones como pueden ser, una plataforma *hardware* concreta, software bajo licencia, o para un tipo de sistema operativo concreto.

En este capítulo se fijan los requisitos mínimos deseables para la herramienta idónea para, tras presentar los simuladores existentes en el estado de la técnica, realizar la elección del entorno en el que se desarrolla el trabajo de modelado y simulación de la plataforma *Cookies*.

I. Requisitos Básicos.

Los requisitos básicos para la herramienta elegida serán clasificados en dos conjuntos diferentes, el primero para características de carácter general de la herramienta y el segundo para características especificas con respecto a las WSNs.

Los principales requisitos para la elección de la herramienta de simulación en el marco del presente proyecto fin de máster han sido:

1. Debe ser una herramienta basada en código abierto, que permita hacer modificaciones en el núcleo del programa sin costes de licencia en caso de que sea necesario.
2. Fácilmente integrable con otra herramienta, para ello es necesario que los datos de entrada y salida estén en un formato estándar
3. Conocido y utilizado en la actualidad por la comunidad científica, y con una actividad reciente en actualizaciones o nuevas versiones
4. Flexibilidad en cuanto a la forma en la que se describen los modelos.
5. Disponibilidad de modelos o que estos sean de fácil desarrollo de los siguientes elementos:
    a. Modelos del canal de comunicaciones
    b. Modelos del canal de sensado
    c. Modelos del *hardware*
    d. Modelos de consumo
    e. Modelos de obstáculos en el medio.
6. Capacidad para manejar objetos (nodos) móviles

Los requisitos funcionales del simulador:

1. Debe ser capaz de mostrar datos de funcionamiento a nivel de cada nodo y a nivel de sistema completo.
2. El simulador deberá contar con modelos de protocolos de comunicaciones, o proporcionar los mecanismos para implementarlos y tener la capacidad para poder aplicar diferentes protocolos de comunicación.
3. La herramienta de simulación debe poder gestionar modelos basados en máquinas de estados finitos y el modelado de eventos discretos, a fin de poder modelar los nodos con un enfoque funcional, evitando así utilizar el código de una aplicación para un *hardware* o un sistema operativo específico.
4. El medio ambiente ha de poder ser modelado de forma sencilla. Debe ser capaz de manejar modelos basados en la atenuación de la señal.

Teniendo en cuenta todos estos requisitos básicos, se ha realizado una búsqueda de las herramientas presentes en el estado de la técnica. A continuación se presentan las principales herramientas encontradas y finalmente, se describen las principales características de la herramienta elegida como plataforma de modelado y simulación

II. Plataformas de simulación analizadas.

Existen multitud de plataformas para simulación de redes en general y de WSNs en particular.

A continuación se hace una breve presentación del las herramientas consideradas más relevantes y que cubren todo el espectro existente en simuladores para redes de sensores inalámbricas. Desde simuladores para plataformas específicas, hasta simuladores de redes tradicionales.

II.1 *NS-2 y Mannasím*

Ns-2 [10]:

El *Network Simulator* (Ns) es un simulador basado en eventos discretos para redes. Está preparado para trabajar con simulaciones de TCP y UDP, varios protocolos para la MAC, soporta diferentes protocolos de enrutamiento de mensajes y multidifusión (*multicast*) tanto para redes tradicionales como para redes inalámbricas.

Mannasím [11]:

Es la extensión para WSN de *Ns-2.* Introduce nuevos módulos para diseño, desarrollo y análisis para aplicaciones basadas en WSN.

El objetivo de *Mannasím* es proporcionar un entorno de simulación en el que se puedan modelar con precisión una amplia gama de nodos y aplicaciones y que sirva como banco de pruebas para algoritmos y protocolos.

Es el simulador de red "por excelencia". El principal inconveniente de esta plataforma es su elevada complejidad ya que requiere un conocimiento elevado de leguajes de programación.

II.2 *TOSSIM*

TOSSIM [12]:

Se trata de un simulador basado en eventos discretos para aplicaciones de WSN basadas en TinyOS. El objetivo es que los usuarios puedan probar una aplicación para en TOSSIM antes de cargar el programa en los nodos. Esto permite a los usuarios depurar, probar y analizar los algoritmos en un entorno controlado y repetible. El objetivo principal de TOSSIM es proporcionar una simulación fidedigna de las funcionalidades de TinyOS.

TOSSIM no pretende simular el mundo real sino el comportamiento de TinyOS. Puede ser utilizado para comprender las causas del comportamiento observado en un despliegue real, y no debe utilizarse para la evaluación absoluta.

No debe considerarse como una herramienta de simulación completa TOSSIM ya que a pesar de hacer simulaciones muy precisas sobre ciertos aspectos hay otros que están demasiado simplificados.

Está dirigido a un sistema operativo muy concreto, por la tanto se queda fuera del conjunto de herramientas que podrían utilizarse en este proyecto, a pesar de se un herramienta muy utilizada actualmente.

II.3 *SHAWN*

SHAWN [13]:

Los objetivos principales en el diseño de *Shawn* son:

- Simular el efecto causado por un fenómeno, no el fenómeno en sí.
- Escalabilidad y soporte para redes de gran tamaño.
- Permitir libertad en la elección del modelo deseado para la aplicación.

El hecho de simular los efectos causados por un fenómeno en lugar de reproducir el fenómeno en si tiene importantes implicaciones en las simulaciones realizadas con *Shawn*.

Por una parte, son más previsibles y hay una ganancia de rendimiento, ya que ese modelo se puede implementar de manera más eficiente. Por otro lado, esto significa que *Shawn* es incapaz de proporcionar el mismo nivel de detalle que un simulador de red tradicional ofrece con respecto a la capa física o de los fenómenos a nivel de paquetes. Sin embargo, si el modelo está bien escogido, los efectos de la aplicación son prácticamente las mismas.

Hay que tener en cuenta las características de los modelos utilizados a la hora de interpretar los resultados obtenidos en una simulación. Si el modelo utilizado no es suficientemente preciso los resultados no serán comparables al del sistema real.

Otro de los objetivos principales de *Shawn* es proporcionar los mecanismos para poder simular escenarios con un gran número de nodos. Para poder aumentar la velocidad de simulación se sustituyen las estructuras de bajo nivel que no afecten al comportamiento general del sistema por estructuras simplificadas más rápidas reduciendo el coste en tiempo de CPU necesario para realizar la simulación del sistema completo.

Para realizar una simulación rápida, *Shawn* permite adaptarse a las necesidades del problema seleccionando las opciones de configuración apropiados. Esto permite a los desarrolladores poder adaptar el rendimiento de *Shawn* específicamente para cada escenario.

II.4 *GTNetS y GTSNetS*

GTNETS [14]:

El *Georgia Tech Network Simulator* (GTNetS) es un simulador bajo licencia BSD para el diseño de topologías de gran escala y con gran número de nodos. La filosofía de diseño de GTNetS es crear un entorno de simulación que se estructure de manera similar a como se estructura una red real.

GTSNETS [15]:

Es un entorno de simulación de redes de sensores que permite a los usuarios evaluar los efectos de diferentes opciones de arquitectura y estrategias para la vida útil y el rendimiento de una red de sensores. También puede ser utilizado para evaluar los protocolos de enrutamiento y algoritmos de cooperación. Incorpora modelos para las distintas unidades funcionales que componen un nodo sensor y caracteriza el consumo de energía de cada uno.

Este simulador es una extensión de GTNetS y utiliza su núcleo de simulación.

Al igual que Ns-2 requiere un alto conocimiento de lenguajes de programación.

II.5 *AVRORA*

AVRORA [16]:

Es un proyecto de investigación del *UCLA Compilers Group*, es un conjunto de herramientas de simulación y análisis de programas escritos para microcontroladores AVR y los nodos de WSN, Mica2 [referencia a Mica2].

II.6 *ATEMU*

ATEMU [17]:

Es un emulador de software para sistemas basados en procesadores AVR.

Junto con el soporte para el procesadores AVR, también incluye soporte para otros periféricos de la plataforma de redes de sensores *Mica2*, como el módulo de radio.

Atemu puede ser utilizado para realizar emulación para redes de sensores con gran número de nodos y modelar su ejecución y las interacciones entre ellos.

II.7 *SENSE*

SENSE [18]:

Las principales características de *SENSE* son:

Simulación basada en componentes, en lugar de en objetos, los componentes cuentan con interfaces compatibles entre sí lo que hace que puedan sustituirse unas por otros o combinarlos para generar componentes más grandes.

Permite elegir a cierto nivel el modo de simulación en paralelo o secuencial, permitiendo que sólo ciertas partes se ejecuten en paralelo.

Es un proyecto en desarrollo y no tienen un versión completamente funcional del software.

II.8 *J-SIM*

J-Sim [19]

Se trata de un simulador orientado a objetos, basado en componentes, escrito en Java. La característica principal de *J-Sim* es que los módulos se pueden añadir y eliminar fácilmente de forma *plug-and-play*. Proporciona modelos de, nodos sensores y *sink*, canales de sensado y de comunicaciones inalámbricas, medio físico, modelos de alimentación.

En este caso, parece un proyecto abandonado ya que carece de nuevas versiones desde el año 2006.

Estas fueron herramientas descartadas por varios motivos, alguno de ellos son simuladores de redes genéricos que no ofrecen soporte para redes de sensores, otros como *Avrora* y *Atemu*, son para plataformas concretas, o sistemas operativos concretos, como *TOSSIM*, y algunos, como *J-Sim*, llevan demasiado tiempo sin actividad y parecen abandonados.

A continuación se describirá con más detalle la plataforma elegida para desarrollar este proyecto, *VisualSense*.

III. VisualSense.

VisualSense [20] es una plataforma específica para redes de sensores desarrollada dentro del proyecto *Ptolemy* de la universidad de Berkeley.

VisualSense es una plataforma de código abierto para el "*Modelado visual para sistemas basados en redes inalámbricas y de sensores*". Está implementado sobre *PtolemyII* [21], el entorno software para el modelado, simulación y diseño sistemas embebidos, concurrentes, en tiempo real, del proyecto Ptolemy.

Se trata de un proyecto sostenido a lo largo de los años (el primer *Ptolemy II*, data del año 2000 y el primer Ptolemy del año 93), *VisualSense* está disponible como parte de *PtolemyII* desde 2004.

Proporciona modelos básicos que puede ser modificados utilizando Java, se pueden combinar los modelos provistos para crear nuevos modelos complejos en la GUI (interfaz gráfica de usuario), o crear modelos nuevos partiendo del las "clases padre" proporcionadas en el entorno.

También tiene la capacidad de combinar los modelos con comportamientos definidas en distintos dominios (con tiempo continuo, de eventos discretos, de flujo de datos síncronos, máquinas de estados finitos, sistemas inalámbricos, etc.).

Debido a todos estos motivos, fue elegido como el marco ideal para modelar y simular una plataforma modular, que ofrece una gran flexibilidad para simular modelos en diferentes niveles de abstracción para la misma plataforma o para cambiar fácilmente los modelos de los diferentes componentes del *hardware* (fuente de alimentación, sensores, comunicaciones, etc.).

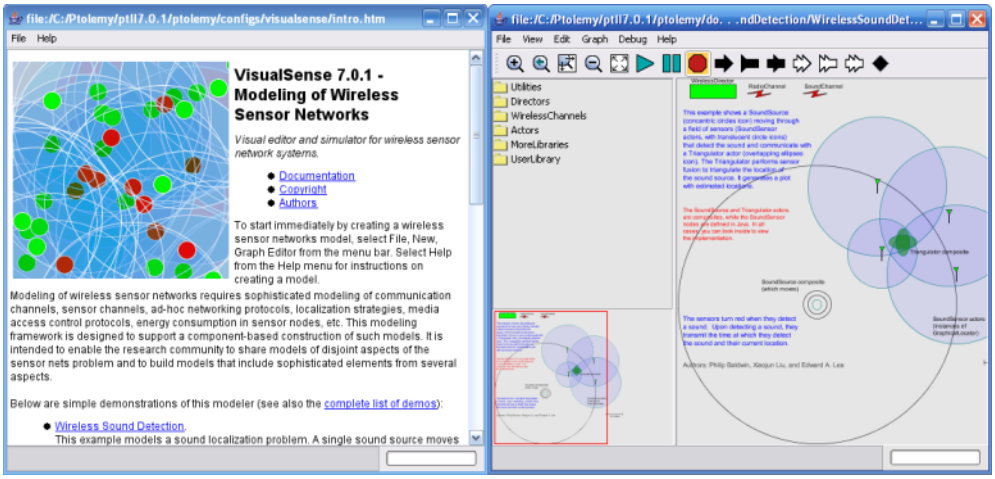


Figura 5: VisualSense. Ventana principal y entorno de simulación.

Los componentes específicos para aplicaciones basadas en redes de sensores inalámbricas proporcionados en *VisualSense* son:

III.1 Wireless Director.

El director juega un papel clave en *VisualSense*, ya que define el dominio en el que se trabaja. Fija el modelo de convergencia y los mecanismos de comunicación. En *VisualSense*, el director implementa el simulador. El *Wireless Director* trabaja como un *Diescret Event Director* de *Ptolemy II*.

Este domino establece la semántica de ejecución, la interacción entre los componentes es mediante eventos con marcas de tiempo. Los eventos se van encolando con su marca de tiempo y el director se encarga de planificar y controlar la ejecución de estos de manera ordenada.

Este domino soporta cambios dinámicos en la topología de interconexión de los componentes. Estos cambios son tratados como mutaciones de la estructura del modelo. Para conseguir que esto sea posible se utiliza *multithread,* de modo que mientras un hilo está controlando la ejecución de la simulación, otro, por ejemplo, se encarga de controlas el movimiento de los componentes. Siendo los son predecibles y consistentes.

La comunicación entre componentes no se realiza de forma explícita (conectando los componente entre sí), en su lugar la asociación entre componentes se hace utilizando canales (descritos a continuación).

El algoritmo para determinar la conectividad entre componentes esta encapsulado en el propio modelo del canal, estos modelos pueden ser implementados por el diseñador del modelo.

III.2 Channel Models.

Un modelo de canal en *VisualSense* es en sí mismo un actor. Cuando un emisor produce un evento en un puerto inalámbrico referencia al canal por su nombre, el evento se entrega al canal para las operaciones necesarias.

El canal puede alterar las propiedades de emisión del transmisor (energía, relación señal/ruido, etc.), y puede retrasar la entrega de los eventos a los posibles receptores para retrasar el modelo de propagación.

III.3 Wireless Composite.

Los propios nodos sensores pueden ser modelados en Java, utilizando diagramas de bloques convencionales, modelarlos como diagramas de flujo de datos, máquinas de estados o incluso modelarlos como sistemas continuos.

En nuestro caso se utilizará el modelado basado en máquinas de estados. Cada uno de estos estados a su vez modelara un modo de operación concreto basado en diagramas de bloques, es decir se definirá una funcionalidad diferente para cada estado del modelo, de esta manera también se pueden modelar ciertas características propias del *hardware*, como por ejemplo variar el consumo del nodo dependiendo de si la radio esta activa o en reposo.

III.4 Collision Detector.

En la semántica de eventos discretos de *VisualSense*, los acontecimientos se producen de forma instantánea en un tiempo concreto. Es decir, no tienen duración.. Para poder detectar la colisión de mensajes dentro de un canal la duración de cada mensaje debe ser añadida explícitamente como un de las propiedades del canal.

Este actor modela una interfaz típica de *PHY LAYER* para un receptor inalámbrico. Modela un receptor en el que los mensajes tienen una duración distinta de cero y estos pueden chocar entre sí provocando un fallo de recepción.

Cuando llega un mensaje a este actor además de los datos deben enviarse también las propiedades correspondientes a la duración y a la potencia de la trasmisión.

El mensaje se mantiene durante un intervalo de tiempo igual a la duración. Si durante este tiempo se superpone otro mensaje y tiene la potencia suficiente, entonces el mensaje que se procesara como fallido. De lo contrario, se procesará normalmente.

Hay un umbral mínimo para detectar un mensaje y una relación señal ruido para detectar colisiones, para poder conocer lo parámetro de potencia y duración necesarios de un mensaje existe el actor *Get Properties* que separa la información transportada en el mensaje de las propiedades del mensaje

III.5 Terrain Property.

Este modela un obstáculo que atenúa las señales que lo atraviesan. La forma del obstáculo y las características de atenuación del obstáculo son definidas por el diseñador del modelo.

III.6 Link Visualizer.

La principal utilidad de este agente es visualizar que nodos se están comunicando entre sí en cada instante por un canal dado.

Dentro de los componentes genéricos, cabe resaltar *Modal Model* por su relevancia dentro del desarrollo de los modelos desarrollados en este proyecto.

III.7 Modal Model.

Este es un actor se utiliza para definir modelos basados en máquinas de estados. Dentro del actor se define una máquina de estados finita y transiciones condicionales al resto de los estados definidos. A su vez cada estado define un modelo de funcionamiento

Cada modelo definido en los estados puede estar descrito en un dominio distinto y requiere su propio *Director*.

En la Figura 6 se puede ver un ejemplo de un modelo basado en *Modal Model*.

Más adelante se hará una descripción más detallada de la metodología de diseño que implica utilizar este tipo de componentes.

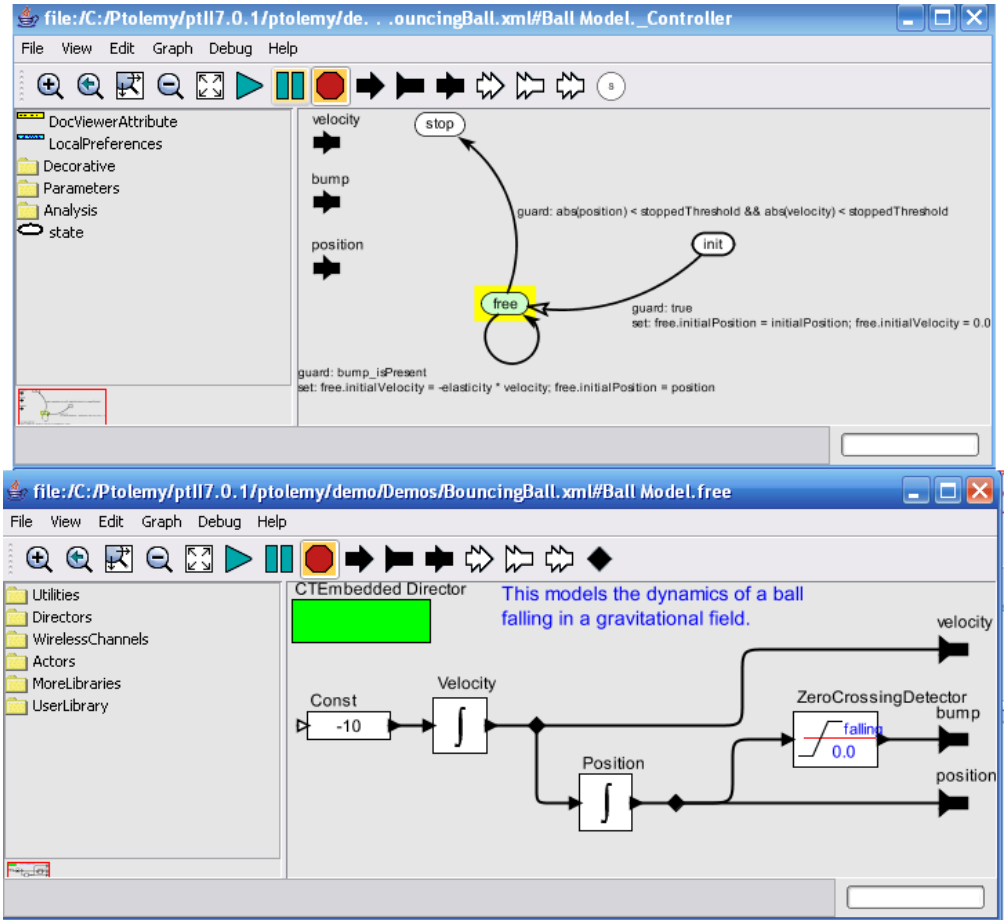


Figura 6: Ejemplo de un modelo basado en Modal Model.

Capítulo 2: Modelado de *Discrete Event Systems.* Aplicación a las Redes de Sensores Inalámbricas.

En este capítulo se describe la metodología de diseño utilizada, modelado de *Discrete Event System (DES)* basado en *Finite State Machines (*FSM), o Máquinas de Estados Finitos., y por qué se ha elegido para realizar el modelado de nuestra plataforma.

Antes de poder empezar a modelar es conveniente recordar que es un Sistema basado en Eventos Discretos o DES.

> *Un DES es un sistema con un espacio de estados discretos, y conducido por eventos, la evolución dentro del espacio de estados depende completamente de la sucesión de eventos discretos y asíncronos en el tiempo"*[22].

La utilización de máquinas de estados en la descripción de los modelos del *hardware* de la plataforma de una WSN permite un modelado rápido y preciso que se adapta a muy bien para describir la naturaleza asíncrona de los eventos que se producen en una WSN.

En el estado de la técnica existen soluciones comparables con la propuesta que se hace en este trabajo de máster.

En [23] utilizan *Discrete Event System Specification* (DEVS), esta forma de caracterizar los parámetros de un sistema de eventos discretos. Utilizan modelos basados en estados utilizando la semántica y las reglas del DEVS para describir sus modelos. Separan las funcionalidades en modelos diferentes, definen la relación jerárquica entre ellos y el formato de los mensajes entre los diferentes bloques. Finalmente utilizan un simulador capaz de manejar modelos descritos según las reglas del DEVS para comprobar la validez de solución.

Es decir, utilizan una técnica de modelado válida para cualquier tipo de sistema basado en eventos discretos, con una semántica y unas reglas precisas que les permitan usar un simulador basado en las reglas de DEVS en concreto.

En [24] utilizan FSMs principalmente para modelar el consumo del los diferentes elementos que componen un nodo. En este caso también separan las funcionalidades por componentes con el objetivo de contar con un conjunto de modelos que permitan generar diferentes configuraciones de un nodo fácilmente. Se centran más en presentar la metodología y marco de diseño utilizando los nodos de una red de WSN como un caso de uso, no como un objetivo en sí mismo.

En [25] se presenta un enfoque para el modelado formal de redes de sensores, con el objetivo de proporcionar una semántica clara que permita crear móldelos

independientes del motor de ejecución que finalmente se vaya a utilizar. Tanto el *hardware* que implementa el nodo, las capas de protocolo de comunicaciones, el código de la aplicación y la física del medio ambiente visto por los sensores, son descritos por un formalismo común. Por otra parte se pretende mostrar un modelo completo construido de forma modular, con diferentes niveles de detalle.

Finalmente plantea un esquema basado en máquinas de estados. Asociado a cada estado hay datos cuantitativos con respecto a su consumo energético y temporal. No pone restricciones a la forma en la que se describe la funcionalidad del *hardware* del nodo, aunque no presenta ningún modelo del *hardware* y se centra principalmente el control de acceso al medio (MAC), la radio y el modelo del medio físico.

I. Metodología de diseño utilizada.

Analizando la definición de DES, se puede ver que un nodo de un WSN se adapta perfectamente a ella, ya que puede definirse un espacio discreto de estados para modelar su funcionamiento. Un ejemplo simplificado para el modelo de un nodo de WSN basado en máquina de estados puede ser: {*SENSADO, EN_REPOSO, ENVIANDO_DATOS, RECIBIENDO_DATOS, SIN_ALIMENTACIÓN*}.

Estos estados, o algunos de ellos, por ejemplo, {RECIBIENDO_DATOS} y {SIN_ALIMENTACIÓN} dependen claramente de eventos asíncronos, ya que son dependientes de la topología, pueden variar dinámicamente y nos son predecibles ya que no dependen del propio nodo.

Para describir estos sistemas se han utilizado máquinas de estados finitos (FSM). Los elementos básicos utilizados en una FSM para describir un sistema son: entradas, salidas, funciones de transición entre estados y funciones para generación de los valores de salida.

En la Figura 7 se ve la estructura de un modelo genérico para una FSM:

- X, valores a la entrada
- Y, los valores de la salida.
- S, el conjunto de estados del modelo
- T_{int} la función de transición interna, es decir la función de transición debida a eventos internos
- $T_{ext,}$ la función de transición externa, es decir la función de transición debida a eventos externos
- Δ, es la función de generación de valores externos.

Además, es necesario definir el estado inicial del modelo y el/los estados finales, aunque los últimos no son obligatorios.

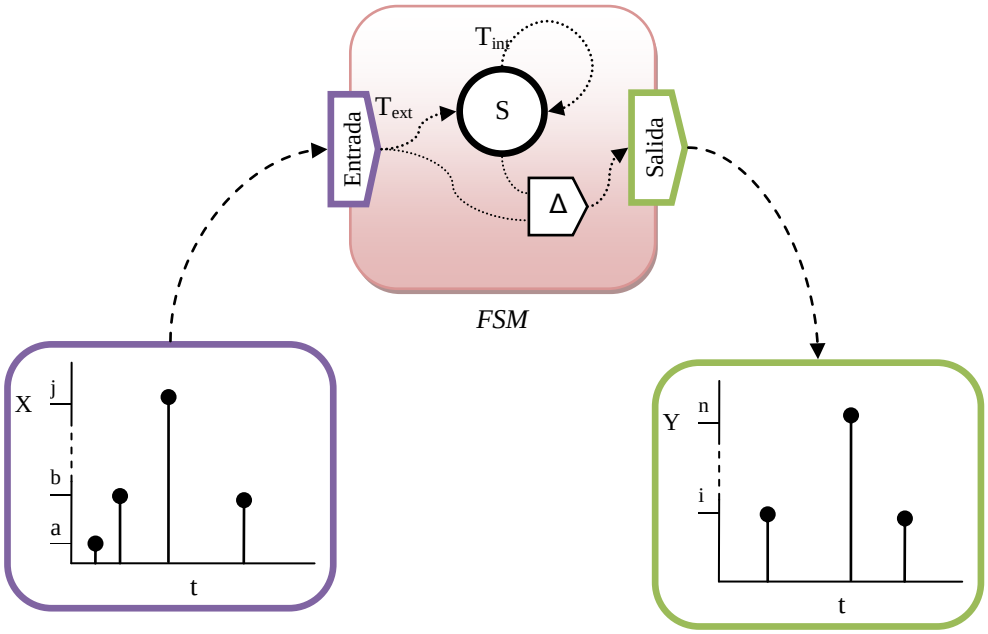


Figura 7: Esquema básico de un FSM.

Si utilizamos como ejemplo el modelo básico del nodo presentado en la Figura 8. Cada uno de los elementos correspondientes con los componentes básicos de una FSM serían:

- X→ Existen dos entradas, *CommIn* y *Sense*, para representar el canal de comunicaciones y sensado respectivamente.
- Y→ Sólo hay una salida, *CommOut*, que representa el canal de transmisión de mensajes.
- S → [*standby*, *acq*, *sending*, *receiving*, *dead*]
- T_{int} → todas las que no tengan que ver con los puertos de entrada, p. ej., en todas las que esté involucrado *intEvent*, puerto para transiciones debidas a eventos internos
- $T_{ext,}$ → transiciones en las que estén involucradas los puertos de entrada, *CommIn* y *Sense*
- Δ → en el caso del ejemplo sólo se general valores en *CommOut* cuando el modelo esta en el estado *sending*, que representaría a un nodo enviando un mensaje a la red.
- Estado inicial y final, *standby* y *dead*, respectivamente.

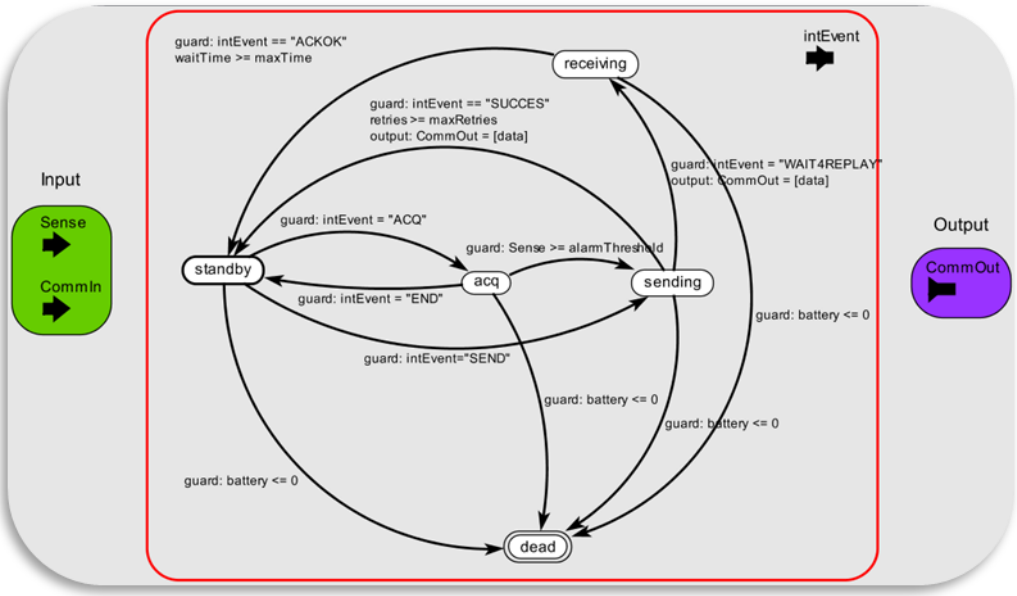


Figura 8: Modelo simple de un nodo.

Pero con la definición de la máquina de estados no queda definido el modelo. Es necesario definir el modelo de cada uno de los modos de operación del *Modal Model* (si el modo de funcionamiento es sencillo, esto no es obligatorio).

Para poder hacer este tipo de modelos, *VisualSense* proporciona los *Modal Models*. Estos elementos permiten hacer exactamente esto, es decir, crear una máquina de estados que defina los modos de funcionamiento y crear modelos complejos por debajo de cada estado, a estos modelos se les denomina *refinement*. En la Figura 9 se muestra un *Modal Moldel* genérico para un sistema con 2 estados, 2 entradas y una salida.

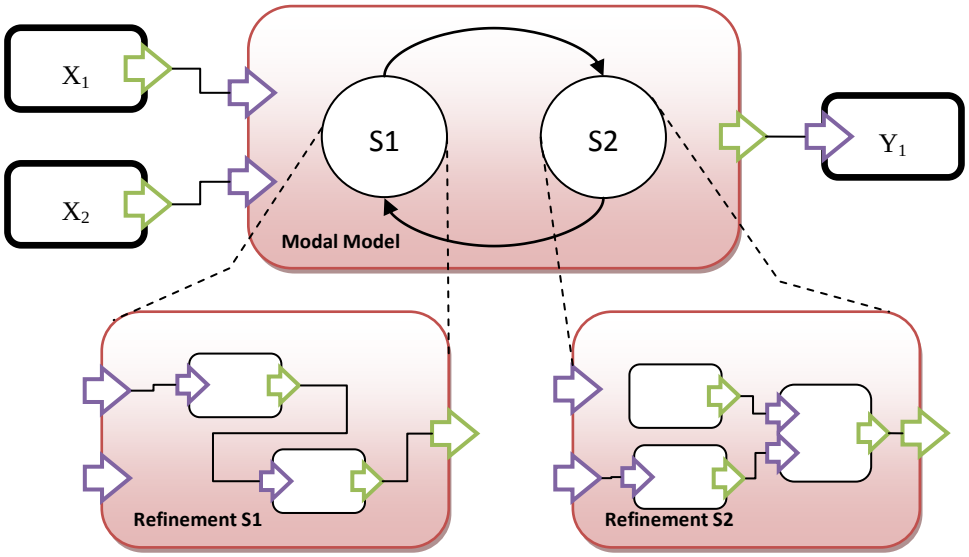


Figura 9: Modal Model.

Recapitulando, los pasos seguidos para definir los modelos han sido:

- Primero, definir la FSM con los estados y las funciones de transición entre ellas.
- Segundo, definir el comportamiento del modelo en cada estado (modo de funcionamiento).
- Tercero, definir las relaciones entre modelos para generar un modelo funcional completo o *Composited Actor*.

A continuación se presentará el modelo creado, empezando por el modelo global del nodo y la interfaz entre módulos. Después se describirán en profundidad los modelos creados para cada bloque funcional o, en el caso de las Cookies, capa de la plataforma *hardware*, tal y como aparece en la Introducción en la Figura 1.

Básicamente, se desarrollaran 4 modelos. Dichos modelos cuentan con una serie de puertos de entrada y salida que les permiten comunicarse entre sí.

La idea básica es que cada modelo ejecuta su propia funcionalidad, y cuando ocurra algún evento que afecte al estado o funcionalidad de los demás, este envía un mensaje a los módulos afectados. Los módulos que reciben dicho mensaje actualizan su estado y, por consiguiente su modelo de simulación activo.

Por ejemplo, cuando en el modelo de la capa de alimentación se genera un evento de fin de batería y les envía un mensaje a todos los demás módulos informando de este

evento. Cuando los demás módulos procesan el mensaje recibido, pasan a su estado inactivo.

II. Interfaz del Modelo desarrollado.

Como se presentó anteriormente se ha realizado un modelo del nodo basado en 4 submoldelos independientes uno por cada una de las capas que componen un nodo de de la plataforma *Cookies.*

Además, estos modelos han de poder ser intercambiados para las distintas versiones de las capas existentes. Por lo tanto es necesario generalizar la interfaz de estos modelos para que puedan intercambiarse sin que la integridad del sistema completo se vea afectada.

En este epígrafe se presentará dicha interfaz y los posibles tipos de mensajes y eventos que relacionan los módulos entre sí.

La interfaz básica de cada bloque será un *Modal Model* con puertos de entrada por los que recibirán los eventos de los demás modelos y/o del medio, y un puerto de salida para comunicarse con cada uno de los demás modelos y/o el exterior.

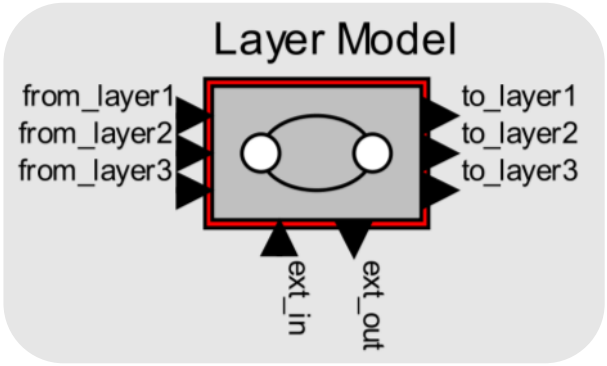


Figura 10: Interfaz genérica del modelo de una capa.

Con respecto al modelo de la plataforma *Cookies* que aparece en la Figura 11:

El módulo de procesamiento tendrá 3 puertos de entrada y 3 tres puertos de salida, para enviar y recibir mensajes a cada una de las capas restantes.

El módulo de sensado, cuenta con un puerto de entrada externo, con el que se comunica con el exterior para realizar las posibles medidas, un puerto de entrada por el que recibe los mensajes de control de la capa de procesamiento, un puerto de entrada de la capa de alimentación y 2 puestos de salida uno para la capa de procesamiento y otro para la capa de alimentación.

El módulo de alimentación cuenta con 3 puertos de entrada, por cada una de las capas restantes por el que le mandan mensajes con sus datos de consumo energético y un puerto de salida, por el que envía el mensaje de fin de batería, este puerto ha sido denominado *Dead*, ya que este es el único mensaje que manda el modelo de la capa de alimentación.

El módulo de comunicaciones cuenta con 2 puertos de entrada, un de la capa de alimentación y otro de la capa de procesamiento, 2 puertos de salida, uno para comunicarse con la capa de procesamiento y otro para la capa de alimentación. Para modelar la interfaz de recepción y envío de mensajes cuenta con 1 puerto de entrada y puerto de salida externos.

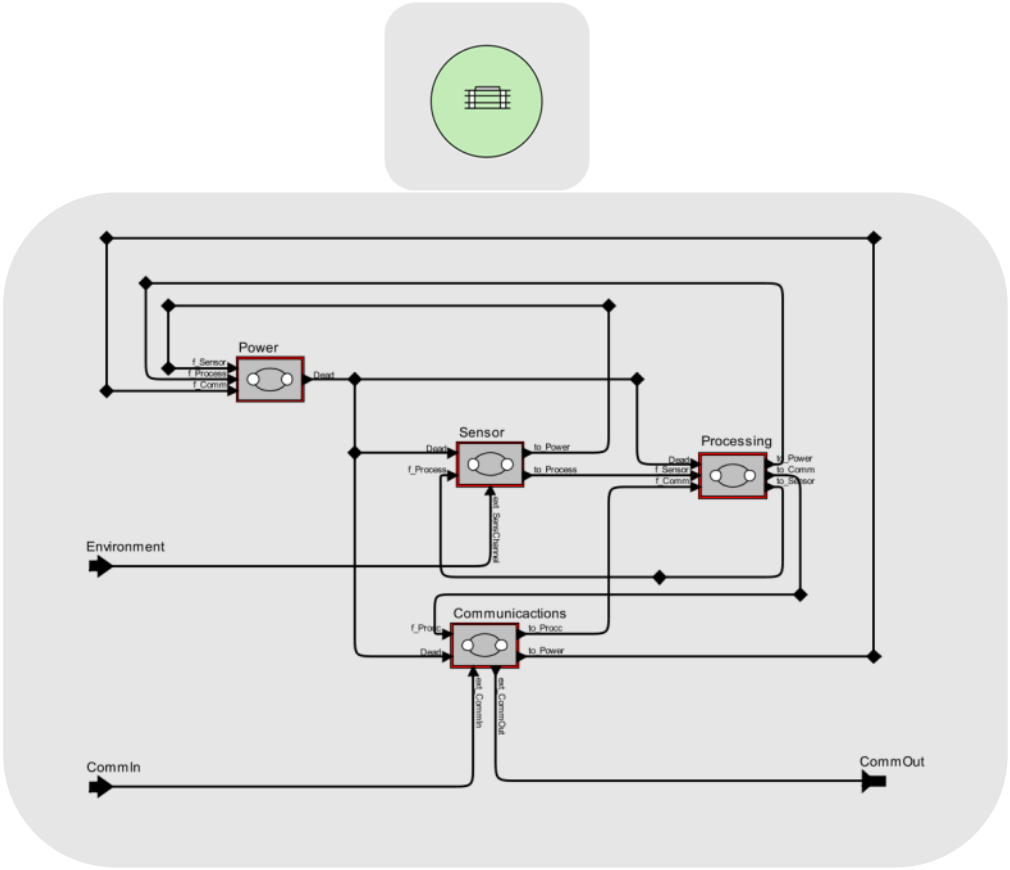


Figura 11: Cookie. Modelo y estructura interna.

Capítulo 3: Modelo de la capa de Alimentación.

En este capítulo se describe el modelo realizado para la capa de alimentación. Se trata de un modelo de descarga simple. Conocido el valor máximo de la vida útil de la batería, el resto de módulos envían un mensaje con el consumo correspondiente. El modelo va acumulando todos los valores enviados hasta que se alcanza el valor de la vida útil fijado. Cuando esto sucede el modelo de la capa de alimentación envía un mensaje indicando este suceso al resto de los bloques

I. Modelo realizado.

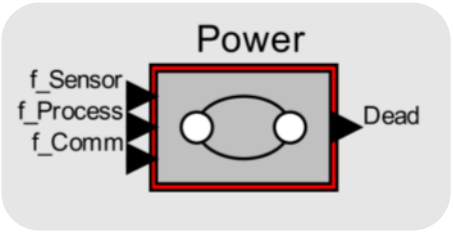


Figura 12: Interfaz del modelo de la capa de Alimentación.

La función de los puertos de la interfaz del modelo que aparece en la Figura 12 es:

- Entradas
  - *f_Sensor, f_Process, f_Comm:* Puerto de entrada para el modelo de la capa de sensores, procesamiento y comunicaciones respectivamente. recibe un número de tipo *double* con el valor de consumo acumulado entre eventos en cada submodelo.
- Salidas:
  - *Dead*: Es el puerto por el que el submodelo de la capa de alimentación comunica al resto de bloques que la energía se ha agotado y por lo tanto deben pasar a su estado de inactividad..

    Este puerto ha sido denominado *Dead*, ya que este es el único mensaje que manda el modelo de la capa de alimentación.

A continuación se describen la FSM utilizada para modelar la descarga de la batería y el *Refinement* del estado principal.

Como se ve en la Figura 13 el modelo para la capa de alimentación es bastante simple, sólo tiene 2 posibles estados; *Live,* si hay energía disponible en el nodo y *Dead,* cuando no hay.

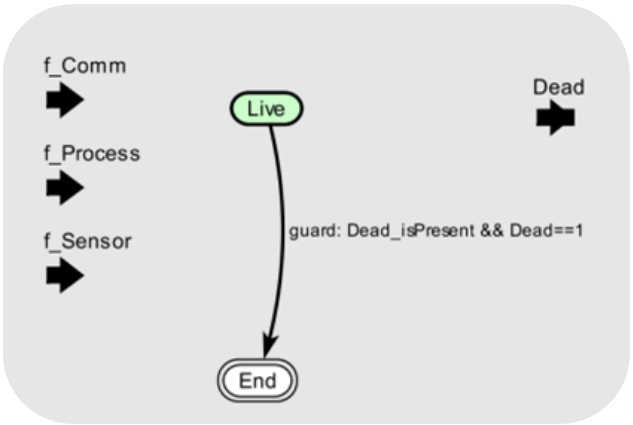


Figura 13: FMS del modelo de la capa de Alimentación.

Como se presentó al comienzo del capítulo, se utiliza un modelo de descarga simple. En este modelo las diferentes capas van mandando su consumo acumulado entre transiciones y se va almacenado, cuando el valor acumulado alcanza el de la batería modelada, se envía un mensaje al resto de los actores indicando que se ha superado el tiempo de vida.

En la Figura 14 aparece el modelo encargado de realizar la función descrita, de una manera muy simple.

Cuenta con un sumador que permite que si se reciben varios mensajes al mismo tiempo se sumen antes de añadir su valor al acumulador y se procesen todos los mensajes recibidos correctamente.

El bloque marcado como *Ramp,* realiza las funciones de acumulador y a la salida se compara el valor acumulado con una variable interna del modelo, denominada *lifeTime,* cuando el valor a la salida de *Ramp* es mayor o igual que *lifeTime,* se envía un "1" por el puerto *Dead,* para indicar que se superó el tiempo de vida fijado en el modelo.

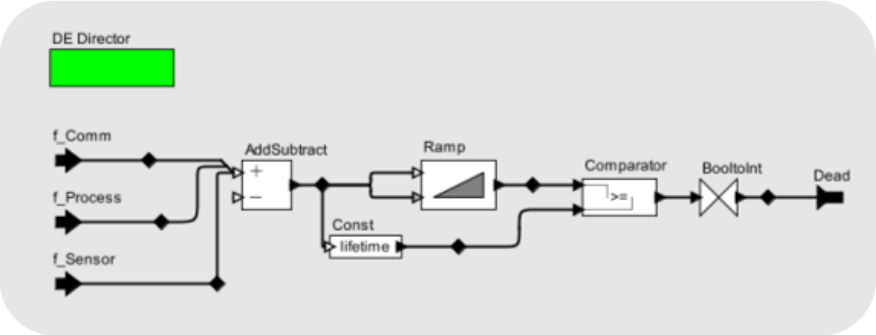


Figura 14: Refinement del estado Live.

II. Limitaciones y Líneas Futuras.

El modelo actual representa un funcionamiento lineal de la batería, sería conveniente hacer un modelo más realista incluyendo el comportamiento no lineal de una batería tales como periodos de recuperación, etc.

En la actualidad existe un capa de alimentación que utiliza paneles solares, por lo que hay que desarrollar un nuevo modelo para esta capa, este modelo requiere además modificar al interfaz del bloque ya que es necesario añadir una entrada externa que permita modelar el efecto de la energía solar en la carga disponible en las baterías

Capítulo 4: Modelo de la capa de Procesamiento.

En este capítulo se presenta el modelo desarrollado para la capa de procesamiento. Este modelo debe representar la aplicación que está ejecutando la unidad de procesamiento del nodo. Por tanto para cada aplicación habrá que, si no desarrollar un modelo nuevo, al menos si adaptarlo para la aplicación concreta.

En caso del modelo aquí presentado, se trata de una aplicación de adquisición de datos muy básica, en la que periódicamente manda monitorizar el entorno a los sensores, y una vez realizadas las medidas pertinentes se envían los datos recopilados a la capa de comunicaciones para que los envía al su destinatario.

Cuando este proceso finaliza, el procesador vuelve a su estado de bajo consumo, hasta que haya que repetir el proceso.

I. Modelo realizado.

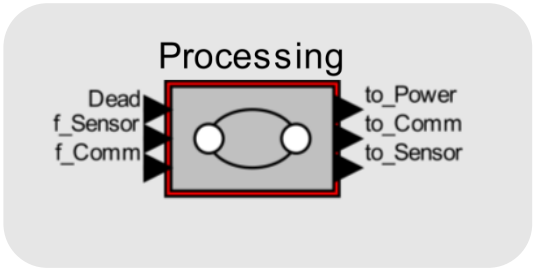


Figura 15: Interfaz de la capa de Procesado.

Como se puede ver en la Figura 15 el modelo cuenta con 3 entradas y 3 salidas ya que hace las funciones de controlador del sistema, por lo que es necesario que se comunique con el resto de bloques.

- Entradas:
    - *Dead:* Señal de entrada del modelo de la capa de Alimentación, que como se describió en el Capítulo 3:, sirve para notificar al modelo que se ha superado la vida útil de la batería.
    - *f_Sensor* Es la entrada por al que se recibirán los datos leídos del exterior.
    - *f_Comm:* Es la entrada por la que se reciben los mensajes destinados al nodo.

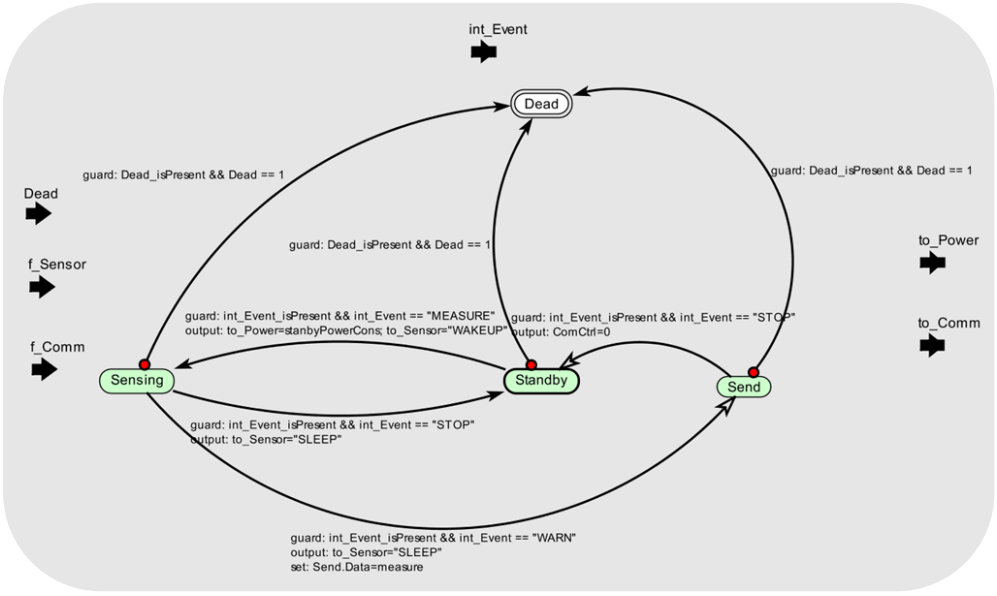


Figura 16: FSM del Modelo de la capa de Procesado.

- Salidas:
  - *to_Power*: Interfaz de comunicación con el modelo de alimentación, envía el consumo de los sensores en cada uno de los estados posibles.
  - *to_Sensor*: Envía los mensajes de control al modelos de la capa de sensores, en este modelo se trabajan con 2 mensajes básicos *"WAKEUP"*, para que esta empiece a medir y "*SLEEP*", para hacer que esta vuelva a su estado de bajo consumo
  - *to_Comm:* Es el puerto por el que la capa de procesamiento envía los datos que el modelo de la capa de comunicaciones enviará a la red.
- Puertos internos:
  - *int_Event:* No es obligatorio que esté, la única función que realiza es que resulte más fácil para el usuario identificar las transiciones más fácilmente.

    En la Figura 16 se pueden ver la FSM con las posibles transiciones que son:

    - *Sensing →Standby;* si el dato recibido es irrelevante simplemente se descarta (*int_Event = "STOP").*

- *Standby →Sensing;* el procesador se despierta y pasa a leer los datos recibidos por los sensores. (*int_Event = "MEASURE")*
- *Sensing →Send;* el dato leído supera el unmbral fijado para generar un mensaje de alrma, (*int_Event = "WARN")*
- *Send →Standby;* se recibe el mensaje de confirmación de recepción por la capa de comunicaciones, el procesador vuelve a dormir. (*int_Event = "STOP").*
- *Todos →Dead,* en cualquier momento que se reciba el evento de fin de batería el modelo pasa a su estado inactivo.

Las operaciones realizadas por cada uno de los estados son:

- *Standby*: Periódicamente un reloj (ver
- Figura 17) genera un evento que hace que el modelo vaya al estado *Sensing*. En la transición envía el consumo asignado al estado a modelo de la capa de alimentación.

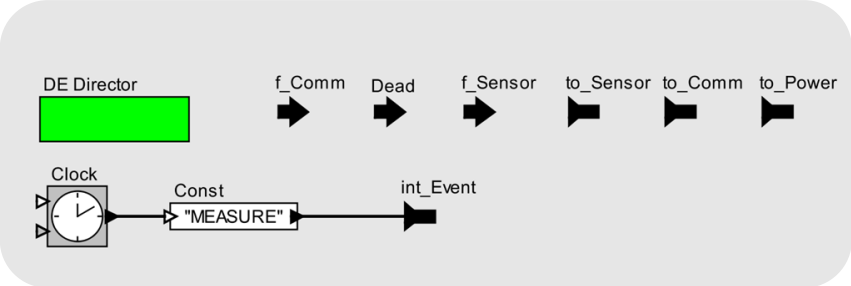


Figura 17: Refinement Standby.

- *Sensing*: Si transcurrido un tiempo, no se ha recibido ningún valor del modelo de la capa de sensado que supere el umbral, el modelo vuelve a su estado *Standby*. Si por el contrario se supera en algún momento el umbral, el modelo evoluciona al estado *Send,* donde genera un mensaje con el valor que genero leído. Estas funcionalidades quedan descritas con el modelo representado en la Figura 18.
- *Send:* Se genera el mensaje en el formato adecuado para que pueda ser entendido por el modelo de la capa de comunicaciones.

  Como el tiempo de envió no es determinista, se guardan los valores temporales al entrar y al salir del estado y se genera el valor de consumo con la diferencia de ambos. El valor generado es enviado al modelo de la capa de alimentación.

Una vez recibida la confirmación por el puerto *f_Comm,* el modelo pasa a su estado *Standby*, para volver a comenzar el proceso. Este proceso se implementa con el modelo de la Figura 19

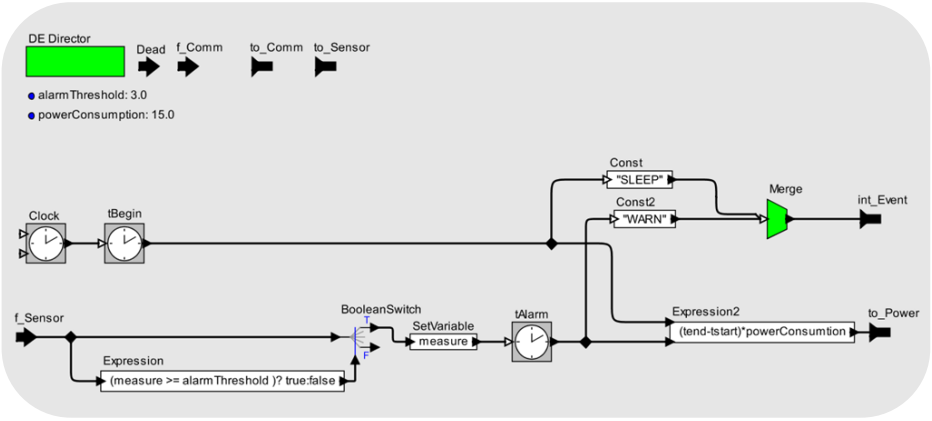


Figura 18: Refinement Sensing.

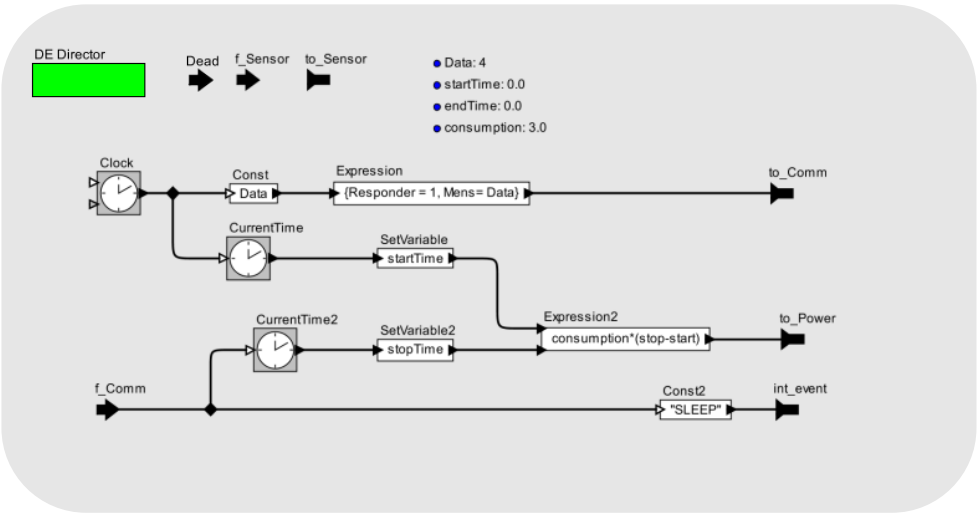


Figura 19: Refinement Send.

II. Limitaciones y Líneas Futuras.

En este caso las limitaciones del modelo son inherentes a la metodología de diseño utilizada, al modelar las funcionalidades del *hardware* obliga a crear diferentes modelos dependiendo de la configuración (aplicación) que ejecute el procesador.

Lo más apropiado seria poder crear mecanismo que permitan adaptar el código de la aplicación a un modelo de máquina de estados, o que el código de la aplicación se pueda comunicar con la interfaz del modelo.

La principal ventaja es que se pueden crear modelos de aplicaciones típicas parametrizables, y poner diferentes modelos dentro de un mismo escenario, ya que una de las limitaciones más típicas de los simuladores que trabajan con el código de la aplicación es que trabajan bajo la suposición de que todos los nodos ejecutan la misma aplicación.

Capítulo 5: Modelo de la capa de Sensado.

En este capítulo se describe el modelo realizado para la capa de sensores.

Se trata de un modelo genérico en el que se modela el consumo de unos sensores a nivel abstractos ya que no importa su funcionalidad concreta y permite leer los datos recibidos por el *SensorChannel*, permite diferenciar entre dos tasa de consumo, esto es útil por ejemplo para sensores que puedan tener un modo de funcionamiento bajo demanda.

I. Modelo realizado

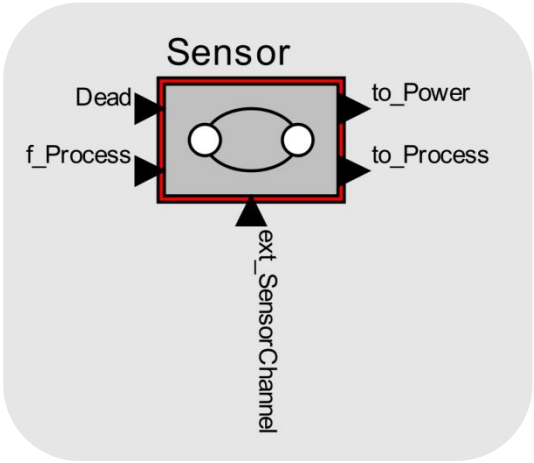


Figura 20: Interfaz del modelo de la capa de Sensado.

Como se puede ver en la Figura 20, el modelo cuenta con 2 entradas, 2 salidas y una señal de entrada externa.

- Entradas:
    - *Dead*: Señal de entrada del modelo de la capa de Alimentación, que como se describió en el Capítulo 3:, sirve para notificar al modelo que se ha superado la vida útil de la batería.
    - *f_Process*: Es la interfaz de entrada con el modelo de la capa de Procesamiento, por este puerto recibe los comandos de comienzo y fin de las operaciones de sensado, *"WAKEUP"* y *"STOP"*, respectivamente

- Salidas:
    - *to_Power:* Interfaz de comunicación con el modelo de alimentación, envía el consumo de los sensores en cada uno de los estados posibles.
    - *to_Process:* Envía al modelo de la capa de procesamiento el valor leído de la señal externa, que representa la variable o variables a medir.
- Señal de entrada externa:
    - *ext_SensChannel:* Representa la interfaz con el exterior, recibe los mensajes del canal de sensado.

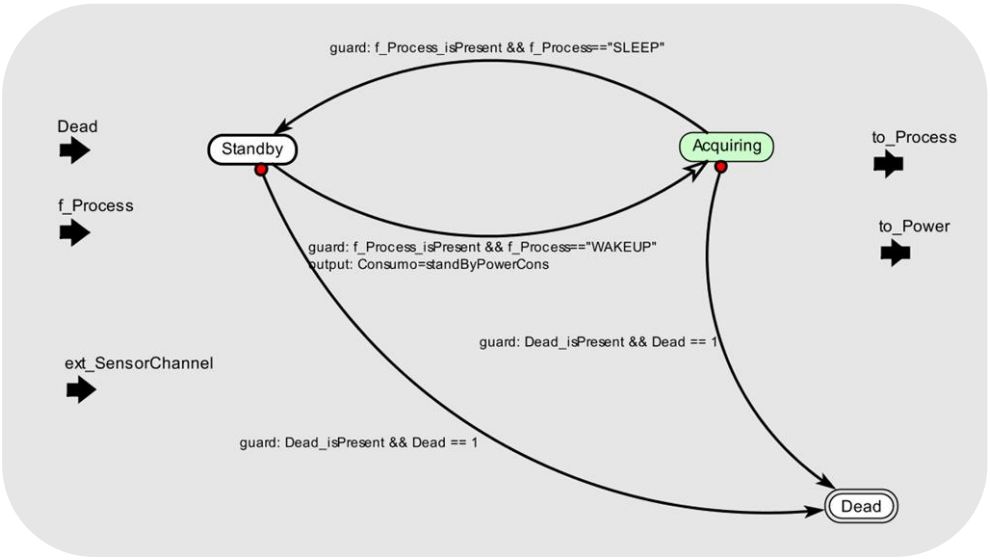


Figura 21: FSM del modelo de la capa de Sensado

En la *Figura 21* aparece la FSM con la que se modela la capa de sensado. Como se ve en la figura cuenta con 3 estados:

- *Standby*: Representa a la capa de sensores en modo de bajo consumo. Para salir de este estado debe recibir un mensaje *"WAKEUP"* de la capa de procesamiento que hace que pase al estado *Acquiring*. Cuando sale de este estado envían los datos de consumo a modelo de la capa de Alimentación.
- *Acquiring:* Como se ve en la Figura 22, en este estado se envían los datos recibidos por el puerto *ext_SensorChannel* al modelo de la capa de procesamiento, donde se realizaran las acciones pertinentes con la

información recibida. Además al recibir un evento por el puerto *f_Process* envía un mensaje con el consumo acumulado de los sensores.

Este modelo está pensado para un funcionamiento periódico, y por tanto los valores de consumo se suponen conocidos y constantes en cada uno de los estados.

- *Dead:* Es el estado inactivo del nodo, el modelo adopta este estado cuando recibe un "1" por el puerto de entrada del mismo nombre.

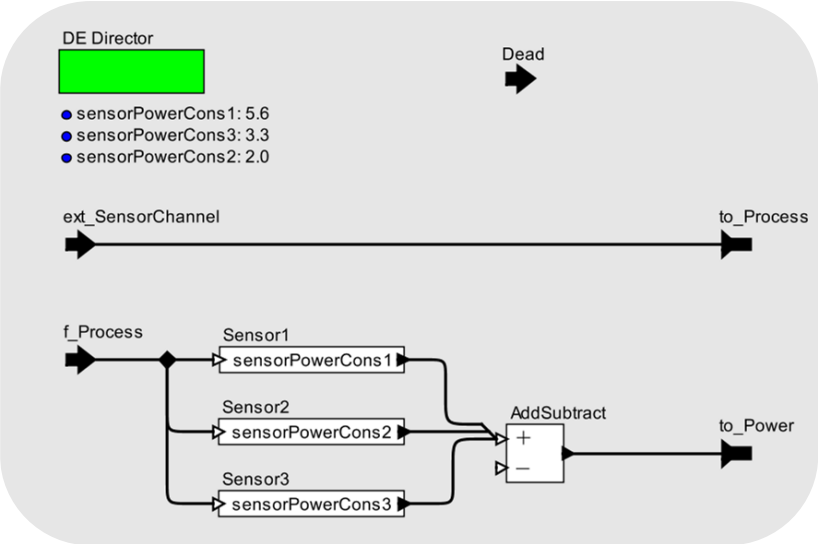


Figura 22: Refinemet Acquiring

II. Limitaciones y Líneas Futuras.

Actualmente no existen modelos en *VisualSense* ni del medio como tal ni de los sensores. Este modelo sólo representa el consumo de la capa de sensado y pretende representar una primera aproximación que proporcione las características básicas y que sirva de base de modelos más complejos.

Hay que valorar si el hecho de modelar las constantes físicas y las funcionalidades de los sensores aportan un valor añadido importante a la simulación. Dependiendo de la aplicación concreta puede ser importante o no.

Esto demuestra la utilidad de contar con modelos independientes ya que dependiendo de la aplicación a modelar se pueden utilizar modelos centrados en los aspectos más relevantes para un escenario concreto.

Capítulo 6: Modelo de la capa de Comunicaciones.

En este capítulo de describe el modelo de la capa de comunicaciones desarrollado. Pero antes en necesario presentar el protocolo de comunicaciones utilizado, *ZigBee* y hacer una descripción de los detalles más importantes implementados en el modelo, como son el protocolo de creación de rutas entre nodos utilizado y el control de acceso al medio y prevención de colisiones utilizados por los dispositivos basados en *ZigBee.*

I. ZigBee.

El estándar de *ZigBee* [26] ha sido diseñado por la *ZigBee Alliance*, un consorcio de empresas formado por más de 150 compañías entre promotores y participantes y utilizado en la actualidad por centenares de compañías. La finalidad del estándar es proporcionar un protocolo para dispositivos de redes inalámbricas, con bajo consumo de energía y que resulte competitivo en precio

El protocolo ha sido diseñado para ser utilizado principalmente en aplicaciones de control o monitorización por lo que conseguir un protocolo con bajo consumo energético ha sido una prioridad. Para poder obtener dispositivos de bajo consumo se fijó una restricción en la velocidad máxima de transferencia de datos. (Velocidad típica 25 kbits/s). Muy por debajo de, por ejemplo, *WiFi*, o *Bluetooht*.

De esta manera, ocupaba un nicho de mercado que no estaba cubierto, el de las aplicaciones de WSNs, en las que uno de los objetivos principales es poder tener dispositivos alimentados por baterías desechables (típicamente 2 pilas de uso domestico AA) funcionando de manera ininterrumpida durante varios meses o años.

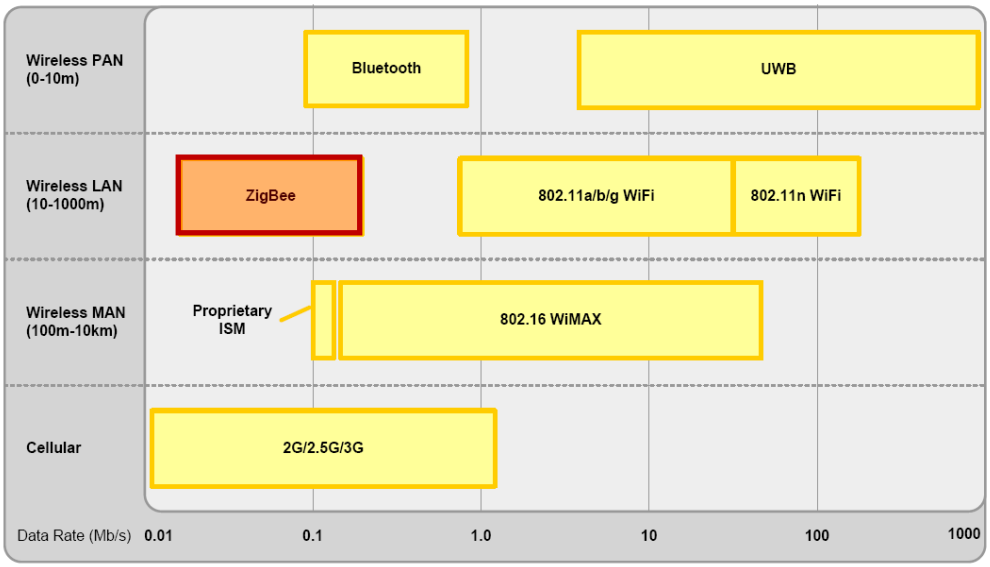


Figura 23: Comparación de Tecnologías de Comunicaciones[27]

Las principales características de *ZigBee* son [28]:

- **Fiabilidad**:

  Se apoya en el estándar IEEE 802.15.4 [29], que se trata de un protocolo robusto para comunicaciones inalámbricas de corto alcance.

  Este protocolo utiliza CSMA-CA (*Carrien Sense Multiple Access Collision Avoidance*) para evitar las colisiones producidas por dos nodos que, en la misma área, intentan enviar datos a la vez.

  Utiliza un CRC (*Cyclic Redundant Checksum*) de 16 bits, para asegurar la integridad en la recepción de los mensajes.

  La topología de redes en maya, lo que le permite reparar las rutas entre dos puntos de forma dinámica.

- **Bajo consumo:**

  Se pretende que un dispositivo *ZigBee* que esté alimentado por baterías esté durmiendo la mayoría del tiempo, activando su módulo de radio solamente cuando se produce algún evento que despierte al módulo.

  Poniendo como ejemplo 2 posibles aplicaciones. Si se utiliza para controlar la iluminación de una sala la radio sólo entraría en funcionamiento al pulsar el botón, lo que sería menos de 10 veces al día durante 5 segundos (aproximadamente). O en el caso de un detector de incendios, sólo activaría

la radio cuando el sensor de humo generara la alarma, pongamos, una vez en 5 años.

Esto es demasiado pretencioso, ya que los nodos necesitan comunicarse periódicamente con sus vecinos para mantener la estructura de la red. Pero, aun teniendo en cuenta estos mensajes, pueden alcanzarse periodos de inactividad de la radio muy elevados con el consiguiente ahorro energético.

- **Seguridad:**

  Utiliza el estándar AES-128 (*Advanced Encryption Standard*) [30].

  Este estándar libre de patentes, es suficientemente fiable y conocido internacionalmente y se puede implementar en microcontroladores de 8 bits.

  En una red *ZigBee* segura, un nodo que no forme parte de la red no puede comunicarse con los nodos de la red (el resto de nodos descartaran los mensajes que este envíe) y no puede decodificar los mensajes que circulan por la red.

- **Baja tasa de datos:**

  Se fijó una baja tasa de datos para poder conseguir bajo consumo y un precio competitivo del *hardware*.

  En la banda de frecuencias en la que operan la mayoría de transceptores *ZigBee,* 2.4 GHz[1], idealmente podrían alcanzarse tasas de 250 kbps. Pero teniendo en cuenta los reintentos, el tiempo de propagación punto a punto y tiempo de encriptación des encriptación la velocidad actual está más cerca de 25 Kbps.

La arquitectura de la pila del estándar *ZigBee* consta de cuatro capas distintas. Las dos capas inferiores (PHY y MAC en la Figura 24) están definidas por el estándar IEEE 802.15.4-2003.

[1] Existen otras dos posibles bandas de frecuencias, en Europa, 868-868.8 MHz, que permite un canal de comunicación, y en EEUU 902-928 MHz, que permite hasta 10 canales.
La banda de 2.4 GHz, es libre a nivel mundial y permite tasas de transferencias más altas. Razones por las cuales es la más utilizada.

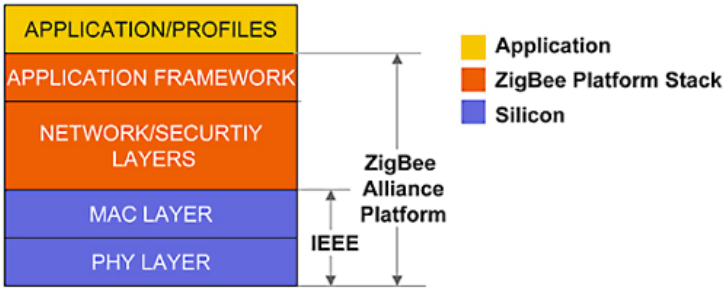


Figura 24: Arquitectura de capas ZigBee.

A continuación se describen brevemente las características más reseñables de cada una de las capas:

Capa Física (*PHY LAYER)*: gestiona el transceptor de radio a nivel físico y realiza la selección de canales y banda de frecuencias. Realiza las funciones de gestión de señal y energía.

Capa de control de acceso al medio (*MAC LAYER*): El control de acceso al medio (MAC) permite la transmisión de las tramas tipo MAC utilizando el canal físico. Es decir, garantizar que el acceso al medio se hace de forma ordenada evitando que dos nodos intenten ocupar el canal al mismo tiempo.

Se encarga de gestionar el acceso a los canales físicos y el balizamiento de la red, si fuera necesario. Para ello este protocolo utiliza CSMA-CA (*Carrier Sense Multiple Access with Collision Avoidance*). El CSMA-CA será descrito en mayor profundidad posteriormente.

Existen dos tipos básicos de redes en el estándar *IEEE802.15.4 beaconing netwoks,* (redes con balizas), y *non-beaconing networks* (redes sin baliza).

En las *beaconing netwoks*, en el intervalo entre *beacons*[2] la radio de un dispositivo sólo están activos una porción breve de tiempo Durante este periodo de actividad los nodos intentan acceder al medio utilizando CSMA-CA y permaneciendo inactivas durante el tiempo restante del intervalo, permitiendo así un importante ahorro energético. El inconveniente principal es que requieren un control de tiempo y sincronización entre nodos bastante preciso, elevando el coste del *hardware*.

En las *non-beconing networks,* la radio de los FFDs estará siempre active y los nodos utilizan una variante del CSMA-CA especial para redes sin balizamiento (*unslotted CSMA-CA*). Esto no significa que los dispositivos n utilicen este tipo de tramas (*las*

[2] Una *beacon* es un tipo de trama especial de la capa MAC. De corta duración, es utilizada para transmitir información de la red y sincronizar dispositivos, sólo pude ser enviada por FFD. En *ZigBee*, se utiliza para transmitir información de la red y se utiliza en la búsqueda de una red cuando un dispositivo quiere unirse a una PAN.

beacons) para otras tareas que también las requieren, como por ejemplo unirse a una red.

En este documento se trataran las *non-beaconing networks*, ya que son las utilizadas por el módulo de radio utilizado en la plataforma *hardware Cookies,* usada como referencia la modelar.

Asimismo, implementa controles de validación tramas y controla los mecanismos de asociación entre coordinadores y dispositivos finales (RFDs y FFDs).

Esto significa que un dispositivo que sólo tenga estas dos capas definidas sólo podría realiza comunicaciones entre coordinadores y RFDs o FFDs. En ningún caso puede haber comunicación entre dos RFDs. Estos servicios entre otros quedaran definidos en la capa de red.

En la Figura 25 aparecen las posibles topologías de red basadas en IEEE 802.15.4.

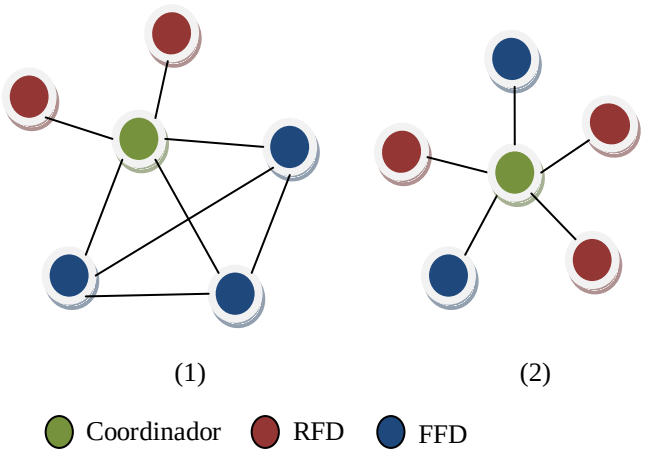


Figura 25: Topologías Peer-to-Peer (1) y en Estrella (2).

Capa de Red (*NWK LAYER*): Esta capa añade la funcionalidad de enrutamiento a la red. Mediante la adición de enrutamiento se posibilita que los dispositivos con capacidad de crear rutas (el coordinador y los *routers*) encaminen correctamente los mensajes que no son para ellos, de manera que estas alcancen su destino de la manera más óptima posible. Esto posibilita que puedan utilizarse redes en malla, como se comentó en la Introducción (ver Figura 2.).

El protocolo de creación de rutas utilizado en *ZigBee,* e implementado en el modelo de simulación desarrollado en el presente trabajo, es el *Ad Hoc On-demand Distance Vector* (AODV). Más adelante, en este capítulo, se presenta todo el proceso seguido por un dispositivo *ZigBee* para la creación de rutas entre dos nodos distantes de una red.

Por otro lado, están las acciones de control llevadas a cabo por la capa de Red, que se utiliza para manejar la configuración de nuevos dispositivos y establecer nuevas redes. Puede determinar si un dispositivo vecino pertenece a la red y descubrir nuevos vecinos y *routers*.

Capa de Aplicaciones (*APP LAYER*): Es posible que un mismo dispositivo *ZigBee* sea utilizado por varias aplicaciones para enviar y recibir datos. La capa de aplicación tiene la responsabilidad de hacer esto de manera invisible para las capas inferiores de la pila. Cuando los datos se han recibido por la capa de red y son entregados a la capa de aplicación, la capa de aplicación se asegura de que llegue a la aplicación correspondiente.

Un ejemplo podría ser una habitación con varios puntos de luz, y también más de un interruptor. Habría un dispositivo *ZigBee* para la habitación luz, y otro para todos los interruptores. La unión, es decir que interruptor controla cada punto de luz, es manejado por la capa de aplicación.

I.1 *Control de acceso al medio en redes sin balizas. Unslotted CSMA-CA.*

Como se ha comentado anteriormente existen dos mecanismos utilizados, dependiendo del tipo de comunicaciones, con balizas o sin balizas.

En la redes sin balizas, los nodos utilizan un acceso de tipo *Unslotted CSMA-CA.* Cada vez que un dispositivo desea transmitir tramas de datos o comandos MAC, este deberá esperar un período de tiempo aleatorio (dentro de unos márgenes establecidos). Si transcurrido este tiempo el canal está libre (no hay ningún nodo en el rango de escucha del nodo emitiendo), el dispositivo transmitirá sus datos.

Si por el contrario el canal está ocupado, el dispositivo esperará otro tiempo aleatorio antes de intentar acceder al canal de nuevo.

Los mensajes de tipo *Acknowledge* (confirmación de recepción correcta de un mensaje) son enviados sin utilizar *CSMA-CA*.

En la Figura 26 se puede observar el flujograma del *Unslotted CSMA-CA.* Las variables y constantes que aparecen en el diagrama se corresponden con:

- NB→ *Backoff Number.* Hace referencia al número de intentos realizados de acceder al canal
- BE→ *Backoff exponent*. Se utiliza para calcular el tipo de espera aleatorio antes de comprobar si el canal está libre. La unidad temporal mínima es el *backoffperiod* cuya duración está definida en el estándar.
- CCA→ *Clear Channel Assessment.* Define el tiempo que el dispositivo esta a la escucha antes de ocupar el canal. La duración del CCA está definida en el estándar.
- *macMinBE*→ Es una constante definida en el protocolo y se utiliza como valor inicial para BE.
- *macMaxCSMABackoffs*→ Es una constante del protocolo que define el número máximo de reintentos para enviar un mensaje.

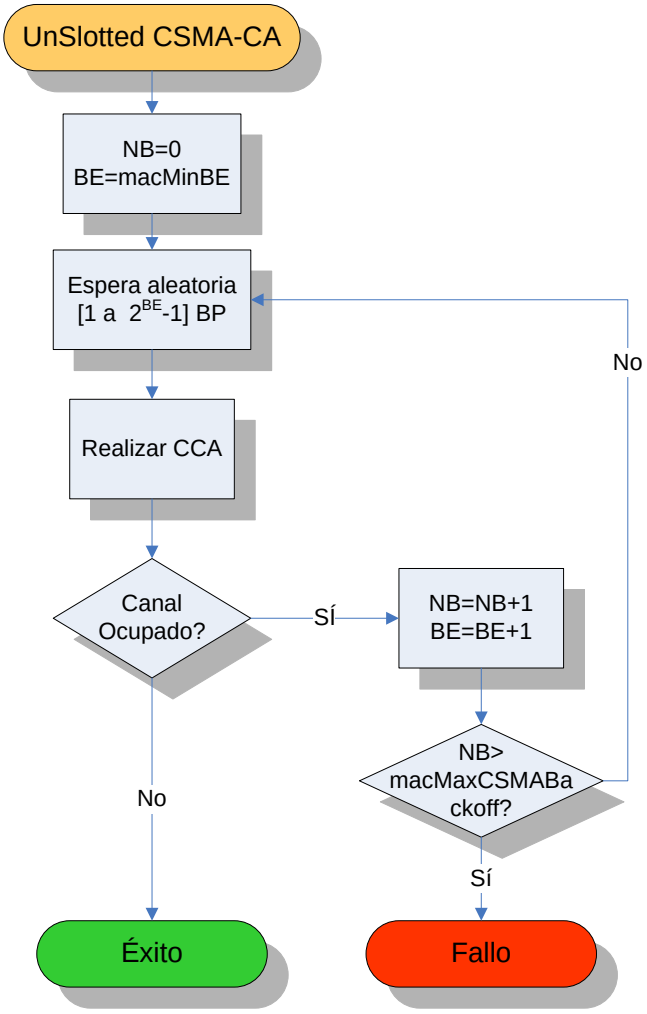


Figura 26: Unsoltted CSMA-CA.

I.2 *Protocolo de rutado de mensajes Ad Hoc On-demand Distance Vector.*

Ad Hoc On-demand Distance Vector (AODV) [31] es un protocolo de enrutamiento de mensajes en redes móviles. Permite que los nodos de la red puedan comunicarse con nodos distantes (con los que no tienen comunicación directa) enviando mensajes a través de los nodos vecinos (con los que tienen comunicación directa).

Para lograr este objetivo, en AODV se crean rutas en la red a través de las cuales puedan comunicarse nodos distantes. También se garantiza que las rutas creadas no contienen bucles y tratado además, de obtener la ruta más corta posible. Este protocolo tiene la capacidad de manejar cambios dinámicos en las rutas y se puede crear nuevas rutas si se detecta un error en la comunicación entre dos nodos.

El proceso de creación de rutas entre nodos se inicia solamente cuando dos nodos quieren comunicarse. El nodo que quiere enviar un mensaje a un nodo lejano envía un *broadcast* a la red solicitando un camino para llegas al nodo lejano, este *broadcast* es reenviado por todos los nodos hasta que alcanza al destinatario. Cuando el mensaje llega al destinatario, este envía la respuesta a través del nodo del que le llegó la solicitud de ruta, de esta manera se evita que se genere tráfico innecesario dentro de la red.

Otra característica importante es que los nodos no conocen la ruta completa, sólo la dirección del receptor, y la del próximo nodo al que hay que enviar el mensaje. De este modo se reduce el tamaño de los datos almacenados y la información que se transmite en un mensaje, de esta manera se optimizan al máximo los recursos *hardware* y el uso de la red, reduciendo el consumo, ya que cuanto más corto se a un mensaje antes se puede pasar a su estado de bajo consumo el módulo de radio.

El algoritmo de creación de rutas de AODV, de una forma muy simplificada es el siguiente:

1. El nodo que desea comunicarse con un nodo distante por primera vez, transmite una solicitud de ruta a todos los nodos en su radio de alcance (*Broadcast*). Esta solicitud se denomina RREQ (*Route Request*). Contiene información acerca del nodo que la envía, el nodo que la originó (remitente), nodo al que se dirige (destinatario), el número de saltos que lleva viajando por la red, así como un identificador inequívoco de dicho RREQ.
    a. Inicialmente el nodo que la envía y el que la origino son el mismo y el número de saltos es 1.
2. Esta solicitud llega a los nodos del entorno, si ninguno de ellos es el nodo destinatario de la solicitud vuelven a retransmitir dicho RREQ almacenando los datos de la solicitud, nodo desde el llegó, destinatario y remitente

   a. Ahora el nodo que lo envía y el nodo que la origino son distintos y se incrementa el número de saltos.

   b. Pude ser que un mismo RREQ llegue dos veces al mismo nodo, pero por caminos distintos. Para eso existen el identificado del RREQ, ya que si el nodo detecta que se trata de un RREQ repetido simplemente lo ignora.

   c. Este proceso se repite hasta que alcanza al nodo de destino o supera el número de saltos máximo fijado.

3. Cuando esta solicitud llega al destinatario, este responde con una confirmación de ruta o RREP (*Route Replay*). Esta en enviada únicamente al nodo desde el que llegó el RREQ con menos número de saltos.

   a. La ruta va viajando hacia el origen gracias a los datos que tienen almacenados los nodos intermedios. Estos nodos intermedios también aprenden la ruta con el destinatario al recibir el RREP.

   b. El resto de nodos que recibieron el RERQ pero que no forman parte del camino optimo, transcurrido un tiempo, borran los datos relativos al RREQ que ha quedad sin contestación.

El proceso descrito está muy simplificado ya que realmente tiene mayor complejidad en cuanto a tráfico de mensajes, manejo de datos de la memoria de los nodos, optimización de rutas, etc. Sólo pretende servir como base para entender mejor el proceso de creación de rutas de *ZigBee*.

En la Figura 27 se puede ver una representación grafica de este mecanismo de creación de rutas.

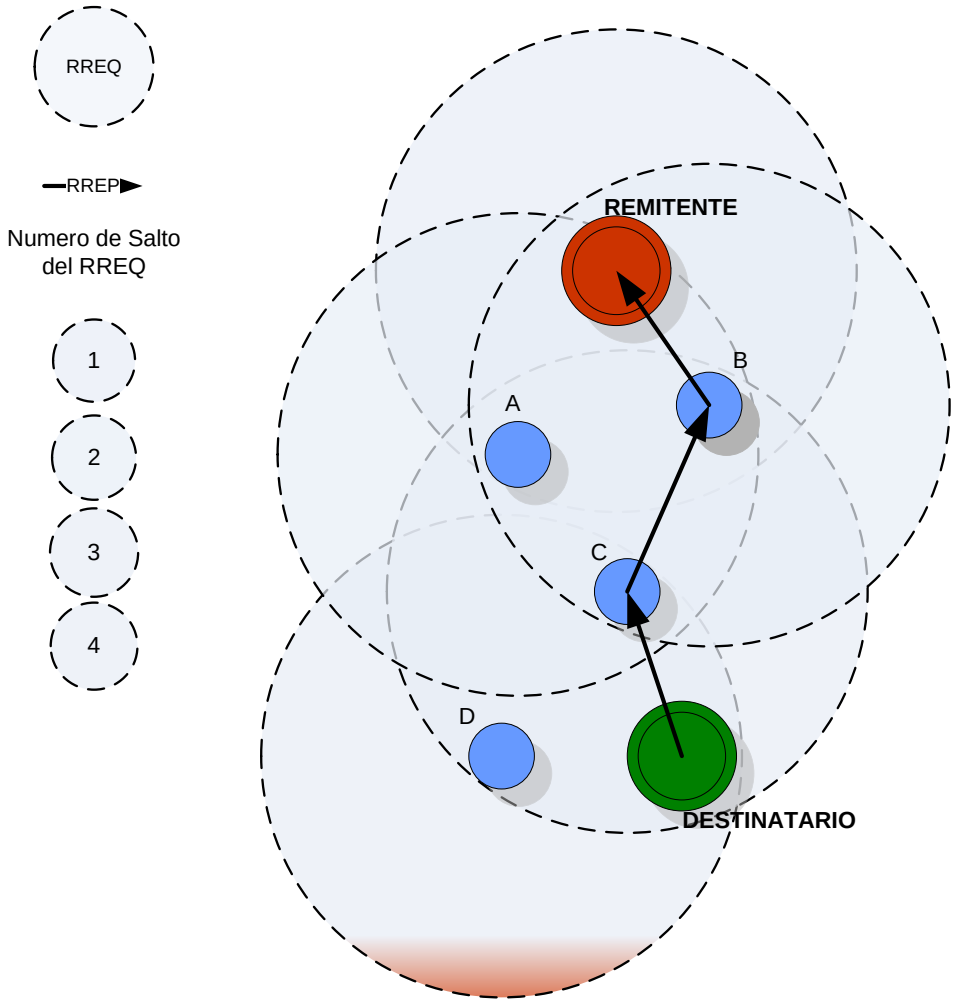


Figura 27: Creación de rutas con AODV.

Para el ejemplo de la figura el proceso sería el siguiente.

1. El REMITENTE envía una solicitud de ruta con el DESTINATARIO.
2. A y B no son DESTINATARIO por tanto ambos incrementan el contador de número de saltos y reenvían la solicitud a su entorno y almacenan la solicitud de REMITENTE.
3. B consigue ocupar primero el canal, por tanto es el primero en reenviar la solicitud que llega a C.
    3.1. También llega a REMITENTE y A pero la identifican como repetida y la descartan.

4. C no es DESTINATARIO por tanto repite el proceso y esta vez si alcanza a DESTINATARIO y D.

   4.1. D reenvía la solicitud que llega a destinatario con un número de saltos mayor, por tanto desecha la solicitud.

5. DESTINATARIO recibe el RREQ desde C, y le contesta a C con un RREP, que C a su vez envía a B, que es desde el que le llegó el primer RREQ. Finalmente B lo reenvía a REMITENTE.

   5.1. Ahora REMITENTE sabe que para enviar un mensaje a DESTINATARIO tiene que enviar primero el mensaje a B, esa es toda la información sobre direcciones que necesita, ya que el resto se va resolviendo en cada nuevo salto

I.3 *Creación de rutas entre nodos ZigBee.*

Anteriormente se ha presentado el protocolo de rutado AODV, protocolo en el que se basa *ZigBee* para crear las rutas, aunque introduciendo algunos cambios con respecto a este. A continuación se presenta en detalle el protocolo implementado por *ZigBee*, con todo el proceso de creación de rutas, mensaje y estructuras de datos que un módulo basado en *ZigBee* debe manejar.

En una red *ZigBee* existen 3 tipos básicos de mensajes dependiendo del tipo de direccionamiento que utilizan, estos son:

- ***Broadcast***: Es un mensaje que se dirige a todos los nodos de la red. Normalmente están limitados a una profundidad de red concreta (número de saltos que puede viajar), ya que un nodo que recibe un *Broadcast*, debe retransmitirlo para que llegue a todos los nodos de la red, o al menos hasta que alcance la profundidad establecida.
- ***Multicast***: Cuando un mensaje se dirige a un grupo concreto de nodos se denomina *Multicast*. *ZigBee* permite creación de grupos de nodos, aunque esta característica aun no ha sido modelada y no serán tratados este tipo de mensajes.
- ***Unicast***: Se trata de un mensaje que sólo se dirige a un nodo concreto de la red.

En la creación de rutas entre 2 nodos de una misma red se utilizan 2 tipos de mensajes básicos, y 2 tablas de datos. Estos son:

- ***Route Request* (RREQ)**, cuando un nodo quiere enviar información a un nodo que del que no conoce la forma de llegar a él, envía este mensaje en forma de *Broadcast* a la red, buscando que llegue al nodo destino.

- ***Route Replay* (RREP),** cuando el RREQ alcanza a su destino, o al nodo padre, este genera un mensaje RREP que es enviado en forma de *Unicast* y que en su viaje de vuelta sirve para confirma la ruta entre Origen→Destino si se supone que los links no son simétricos (por defecto en la especificación *ZigBee*).

  En el caso de links simétricos, los nodos por los que viaja el RREQ también aprenden la ruta con el Origen por tanto cuando se confirma el RREP se crea una ruta bidireccional Origen⇔Destino, en el caso de *ZigBee*Pro es así por defecto.

- ***Routing Table***, es la tabla que contiene la información de la ruta con un nodo concreto (destino, siguiente salto, etc).

Field Name	Size	Description
Destination address	2 octets	The 16-bit network address or Group ID of this route. If the destination device is a ZigBee router, ZigBee coordinator, or an end device, and *nwkAddrAlloc* has a value of 0x02, this field shall contain the actual 16-bit address of that device. If the destination device is an end device and *nwkAddrAlloc* has a value of 0x00, this field shall contain the 16-bit network address of the device's parent.
Status	3 bits	The status of the route. See Table 3.52 for values.
No route cache	1 bit	A flag indicating that the destination indicated by this address does not store source routes.
Many-to-one	1 bit	A flag indicating that the destination is a concentrator that issued a many-to-one route request.
Route record required	1 bit	A flag indicating that a route record command frame should be sent to the destination prior to the next data packet.
GroupID flag	1 bit	A flag indicating that the destination address is a Group ID.
Next-hop address	2 octets	The 16-bit network address of the next hop on the way to the destination.

Tabla 1: Routing Table (del estándar ZigBee).

En el modelo sólo se tendrán en cuenta por el momento, el *Destination Adress*, el *Status* y el *Next Hop Adrress*, ya que por el momento no se han modelado, ni el *Route Record*, ni los *Many-to-one Broadcast*, ni el enrutamiento *Source Route*, ni los grupos de nodos.

- Los valores posibles para *Status* pueden ser:

Numeric Value	Status
0x0	ACTIVE
0x1	DISCOVERY_UNDERWAY
0x2	DISCOVERY_FAILED
0x3	INACTIVE
0x4	VALIDATION_UNDERWAY
0x5 – 0x7	Reserved

Tabla 2: Status Values (del estándar ZigBee)

- ***Route Discovery Table,*** contiene la información necesaria para completar el proceso de creación de una ruta. Una vez que se complete el proceso de creación de una ruta en concreto (satisfactoriamente o no), los datos correspondientes a esa búsqueda se borran (o se sobrescriben).

Field Name	Size	Description
Route request ID	1 octet	A sequence number for a route request command frame that is incremented each time a device initiates a route request.
Source address	2 octets	The 16-bit network address of the route request's initiator.
Sender address	2 octets	The 16-bit network address of the device that has sent the most recent lowest cost route request command frame corresponding to this entry's route request identifier and source address. This field is used to determine the path that an eventual route reply command frame should follow.
Forward cost	1 octet	The accumulated path cost from the source of the route request to the current device.
Residual cost	1 octet	The accumulated path cost from the current device to the destination device.
Expiration time	2 octets	A countdown timer indicating the number of milliseconds until route discovery expires. The initial value is *nwkcRouteDiscoveryTime.*

Tabla 3: Route Discovery Table (del estándar ZigBee)

I.3.1 Route Discovery en la especificación de ZigBee.

Existen tres tipos de *Route Discovery*:

- ***Unicast Route Discovery***, el más habitual, un nodo tiene que enviar datos a otro nodo lejano y desconoce una ruta, por lo tanto envía un *Broadcast*, con el nodo final como destinatario (Responder en el modelo.de AODV clásico creado)
- ***Multicast Route Discovery,*** tiene como destinatario un *multicast group* es decir, un grupo de nodos concreto, la *destination adress* de un *Route Request* de este tipo debe ser un *multicast gruop ID* (o identificador del grupo). No se contempla este tipo de rutas en el modelo, no existen *multicast groups* por el momento.
- ***Many-to-one Route Discovery,*** se inician en un concentrador se utilizan para que estos dispositivos conozcan las rutas con todos los coordinadores y *routers* dentro de un radio dado. Tampoco está contemplada en el modelo actualmente.

I.3.2 Cuando se inicia un Route Discovery.

Según la especificación de *ZigBee*. Este proceso se inicia cuando se cumplen las siguientes condiciones:

- Un nodo recibe un mensaje que tiene como destinatario final un nodo diferente al dispositivo actual y este mensaje no es de tipo *Broadcast*.
- El campo *discoverRoute*, del mensaje vale 1, con lo que se habilita la creación de rutas y en la tabla de rutas no existe una entrada correspondiente con el destinatario del mensaje.

Cuando se inicia un RREQ, se pueden llevar a cabo un total de *nwkcInitialRREQRetries +1* intentos, estos deben estar espaciados temporalmente *nwkcInitialRREQRetryInterval* milisegundos.

Tanto *nwkcInitialRREQRetries* como *nwkcInitialRREQRetryInterval* son valores constantes definidos en el éstandar.

Si el dispositivo que inicia el *Route Discovery* no tiene la dirección del destinatario en la *Routing Table*, se debe crear una nueva entrada en la *Routing Table* con status *DISCOVER_UNDERWAY*.

Si el dispositivo ya tiene al nodo destino en su *Routing Table*, con un status *ACTIVE* o *VALIDATION_UNDERWAY*, se deben utilizar los datos correspondientes de la tabla manteniendo el valor del status tal y como esta.

Si existe pero con otro *status* diferente a los anteriores, se deben utilizar los datos almacenados, pero cambiando el estado a *DISCOVER_UNDERWAY*. También se debe añadir la entrada correspondiente a la *Route Discovery Table*, si no existe ya una entrada con el mismo valor de nodo Solicitante e Identificador de la solicitud (*Source Address* y *Route Request ID* en la Tabla 3).

Los dispositivos tienen un contador utilizado para generar el *Route Request Identifier*, cuando se crea una nueva solicitud, este contador se incrementa y se almacena en la *Route Discovery Table* como el *Route Request Identifier*.

I.3.3 Procesado de los mensajes Route Request (RREQ).

A continuación se describen las distintas acciones que debe realizar un nodo dependiendo de las condiciones en la que se recibe el RREQ.

Si el dispositivo es un *EndDevice* debe descartar el mensaje, si no, debe decidir si tiene capacidad para crear rutas (es un FFD o coordinador con espacios libres en sus *Routing Table* y *Route Discovery Table*).

Si el dispositivo tiene capacidad para crear rutas, y se trata de un *Unicast RREQ* (el único que se trata en el modelo), el dispositivo debe comprobar si él es destinatario del comando comparando su dirección con el *Destination Address* del mensaje. También ha de comprobar si el comando es para alguno de sus dispositivos hijo de la misma manera.

Si no lo es. El dispositivo debe comprobar si existe algún elemento en la *Route Discovery Table* que tenga el mismo *Route Request Identifier* y *Source Address* que el RREQ. Si no existe, se añade este elemento a la tabla.

Si se está trabajando bajo la suposición de enlaces simétricos, cuando se recibe un RREQ, se busca en la tabla de vecinos si existe una entrada correspondiente con el dispositivo transmisor.

Si dicho dispositivo no está en la tabla, o el coste de transmisión asociado a dicho dispositivo es 0, la trama se descarta y finaliza el procesado del RREQ.

Para calcular el coste de la ruta se utiliza el valor máximo del coste de emisión o de recepción de mensajes con los vecinos, es decir como estamos suponiendo caminos simétricos se hace una suposición de peor caso. Esto incluye al valor utilizado para incrementar el coste del camino del RREQ antes de retransmitirlo.

Cuando se añade un elemento a la *Route Discovery Table*, los campos han de tener los mismos valores que los del RREQ correspondiente, con la excepción del *Forward Cost* que se determina utilizando al remitente anterior para calcular el coste del enlace (*Link Cost*), este valor se añade al *Path Cost* del RREQ.

Este nuevo valor calculado se almacena en el campo *Forward Cost* del nuevo elemento añadido a la *Route Discovery Table*.

Para el caso de enlaces simétricos, un dispositivo que recibe un RREQ debe añadir un elemento a la *Routing Table* con:

- *Destination Address* ← Creador del RREQ
- *Next Hop Address* ← Dirección del nodo del que se recibió el RREQ
- *Status* ← *ACTIVE*

Cuando un dispositivo responde con un RREP en nombre de uno de sus nodos hijo dependientes, debe poner como *Reponder Address*, la del nodo al que iba dirigido el RREQ.

Cuando un dispositivo con capacidad para crear rutas no es destinatario del RREQ, debe consultar en su *Route Discovery Table* si ya ha recibido un RREQ con la misma *Source Address* y el mismo *Route Request Identifier*.

Si no es así, se debe añadir un nuevo elemento a la tabla, con *nwkcRouteDiscoveryTime* milisegundos como tiempo de vida y un *Status* de *DISCOVERY_UNDERWAY.*

Si ya existía un entrada con esos valores y su S*tatus* no era ni *ACTIVE* ni *VALIDATION_UNDERWAY*, se debe fijar su estatus a *DISCOVERY_UNDERWAY*.

Cuando el tiempo de vida de alguno de los elementos de la *Route Discovery Table* llega a 0 este elemento debe ser eliminado de la tabla.

Si hay un elemento en la *Routing Table* con estatus *DISCOVERY_UNDERWAY* y no hay más solicitudes activas en la *Route Discovery Table* que tengan como objetivo dicha *Destination Address* también debe ser eliminado.

Si ya existía un elemento en la *Route Discovery Table* para un RREQ recibido, se debe comparar el *Path Cost* del RREQ con el *Forward Cost* almacenado en la tabla, esta comparación se debe hacer calculando el *Link Cost* con el dispositivo predecesor y sumándolo con el *Path Cost* acumulado en el RREQ.

Si se obtiene un menor *Forward Cost* que el que se tenía en la tabla se debe cambiar el *Sender Address* y el *Forward Cost* de dicho elemento de la *Route Discovery Table*, si el resultado calculado es peor que el almacenado en la tabla se descarta el RREQ. También se debe actualizar el *Path Cost* del RREQ antes de retransmitirlo de nuevo.

Si se está trabajando con links simétricos también se deben actualizar los campos de la *Routing Table* relacionados con el *Source Address* del RREQ, cambiando en su caso el *Next Hop Adress* por el nuevo valor.

Cuando se envía un RREQ la *NWK LAYER* debe retrasar la transmisión un tiempo aleatorio de:

$$2 \times R[nwkcMinRREQJitter, nwkcMaxRREQJitter](ms)$$

Donde R[a, b] es una función aleatoria en el intervalo [a, b].Se debe ajustar de forma que este tiempo pseudoaleatorio sea mayor cuanto mayor sea el *Path Cost* del RREQ. La *NWK LAYER* debe intentar la emisión, nwckRREQRetries después del primer intento, lo que deja un máximo de nwckRREQRetries + 1 posibilidades de una transmisión correcta. Si mientras está en el compás de espera de un reenvío de RREQ llega otro con un mejor Path Cost se puede elegir descartar el que está a la espera y enviar el nuevo.

El dispositivo debe fijar el Status del elemento de la *Routing Table* correspondiente al *Destination Address* del RREQ a *DISCOVER_UNDERWAY*. Si no existe dicho elemento se debe añadir.

I.3.4 Como se genera un RREP

Cuando se responde a un RREQ con un RREP, el dispositivo debe tener un elemento en la *Route Discovery Table* correspondiente a la *Source Address* y *Route Request Identifier* del RREQ al que se va responder.

El RREP debe tener las siguientes características:

- *Network Header*:
    - *Frame type:* NWK Command.
    - *Source Address* debe ser la dirección de red del dispositivo que está generando el RREP.
    - *Destination Address* debe ser la del dispositivo que creó el RREQ.
- *Command Payload:*
    - *NWK Command Identifier:* RREP
    - *Route Request Identifier*, el mismo que el del RREQ al que se está contestando
    - *Originator Address,* la dirección de red del dispositivo que creó el RREQ.
    - Utilizando la *Route Discovery Table* para saber la dirección del nodo del que llegó el RREQ al que se está respondiendo.
    - El dispositivo debe calcular el *Link Cost*. Este valor se utiliza como primer *Path Cost* del RREP. El *Path Cost* se irá incrementando a

medida que el RREP viaje de vuelta por la red hacia el nodo que inició el *Route Discovery*.

Una vez que se ha generado el RREP adecuado se envía hacia el nodo que inició el *Route Discovery*, utilizando como *Destination Address* en la *MAC LAYER Header* la *Sender Addres* de la solicitud correspondiente de la *Route Discovery Table*. Es decir la del nodo desde el que llega el RREQ al que se está contestando.

I.3.5 Procesado de los mensajes Route Replay (RREP)

Cuando un dispositivo recibe un RREP, debe realizar las siguientes operaciones.

Si el dispositivo no tiene capacidad de crear rutas, debe descartar el mensaje. En caso contrario, debe comprobar si es el destinatario del mensaje, comparando la *Originator Adress* del RREP con la suya propia.

Si es así, se comprueba en la *Route Discovery Table* si existe algún elemento con el mismo *Route Request Identifier* que el enviado en el RREP.

Si existe un elemento que coincida con el *Route Request Identifer* del RREP se busca en la *Routing Table* un elemento que tenga como *Destination Address* la *Responder Address* del RREP.

Si no existe, el mensaje se descarta y finaliza el procesado del mensaje. Si no existe dicho elemento en la *Routing Table*, se elimina de la *Route Discovery Table*, se descarta el mensaje y se termina con el procesamiento del RREP.

Si se cumplen las 2 condiciones anteriores, y el *Status* de la ruta encontrada es *DISCOVERY_UNDERWAY*, se cambia a *ACTIVE*, además se fijan la *Next Hop Address* de la ruta a la *Sender Address* contenida en el *NWK HEADER* del RREP y el *Residual Cost.al Path Cost* contenido en el mensaje.

Si el citado elemento de la *Routing Table*, ya tenía *Status* en ACTIVE, se compara *Residual Cost* en la tabla y el *Path Cost* en el mensaje, si el contenido en el mensaje es menor, se cambian los datos de la ruta, en concreto la *Next Hop Adddress* y el *Residual Cost*, por los del RREP. Si no es así simplemente se descarta el RREP.

Si el dispositivo no es destinatario final del RREP, este consultará su *Route Discovery Table* en busca de un elemento con la misma *Originator Addres* y *Route Request Identifier* que los contenidos en el mensaje.

Si no existe ningún elemento coincidente, se descarta el RREP y si se encuentra un elemento adecuado, se comparan el *Path Cost* del mensaje y el *Residual Cost* del elemento almacenado en la Tabla, si el *Residual Cost* es menor, se descarta el mensaje.

Por otro lado, el dispositivo debe buscar en la *Routing Table* un elemento cuya *Destination Addres* coincida con la *Responder Addres* del RREP.

Se modificará el elemento correspondiente de la *Routing Table*, fijando la *Next Hop Address* a la *Sender Address*(en el *MAC HEADER*) del RREP. También se fijará *Residual Cost* del elemento correspondiente de la *Route Discovey Table* al *Path Cost* del REEP.

Si existe un elemento que cumpla las condiciones en la *Route Discovery Table* pero no se encuentra en la *Routing Table*, se ha producido un error y debe descartarse el RREP.

Cuando la recepción de un RREP hace que la *Routing Table* deba modificarse, se ha de reiniciar el *Expiration Time* del elemento correspondiente del la *Route Discovery Table*, haciendo que este vuelva a ser *nwkcRouteDiscoveryTime* milisegundos.

Tras actualizar los datos de la ruta en el propio dispositivo, este debe enviar el RREP hacia el destinatario, (*Requeseter Address*).

Para conocer la dirección del siguiente nodo el dispositivo debe extraer de la *Route Discovery Table*, la *Sender Address* correspondiente al los datos contenidos en el RREP, esta es la dirección del siguiente nodo al que se enviara él RREP y que se utilizará también para calcular el *Link Cost*. El valor calculado se añade al *Path Cost* ya que antes de reenviar el RREP, se debe actualizar el *Path Cost* acumulado.

Si se está trabajando con enlaces simétricos, además de reenviar el RREP, y sólo en el caso de que no exista ya, hay que crear un ruta inversa en la *Routing Table*, es decir además de confirmar la ruta con enviado el RREP a la *Responder Address* correspondiente, se ha de crear un ruta con el *Requester* del RREP, que tendrá como *Destination Address* la *Responder Address* del RREP, como *Next Hop Address*, la *Destination Address* del *MAC HEADER* del RREP que va a ser enviado y *Status ACTIVE*.

En la Figura 28 aparece la estructura y los campos más importantes de los mensajes RREP y RREQ utilizados durante el proceso de creación de rutas.

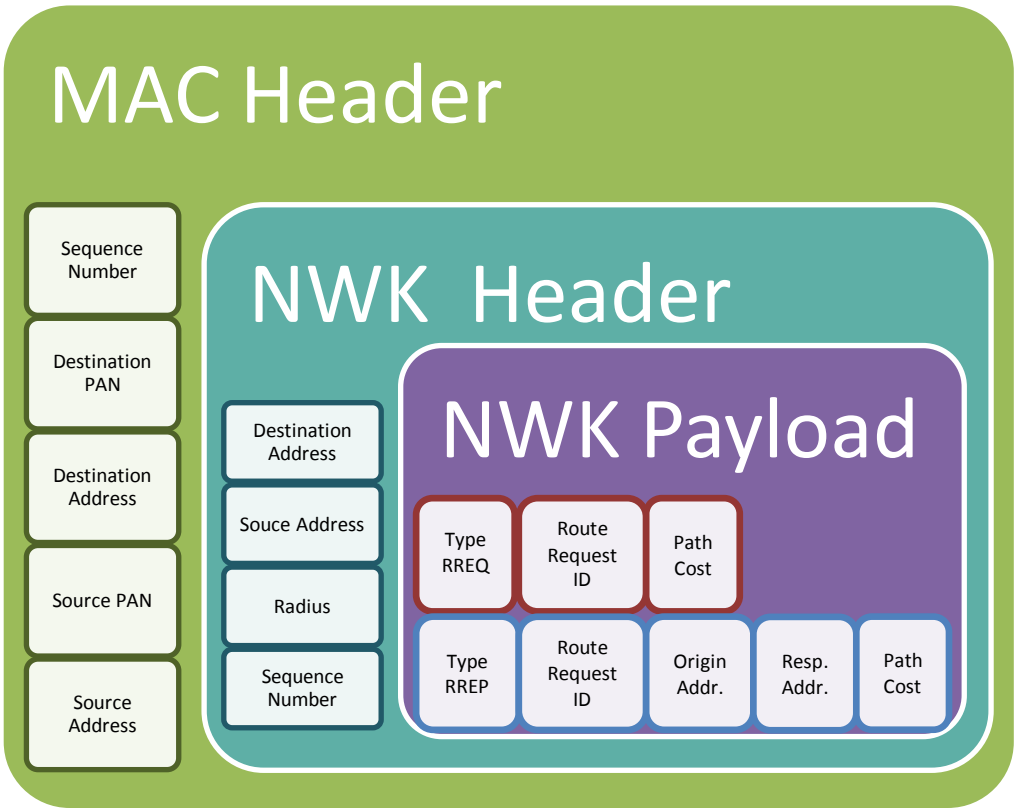


Figura 28: Campos principales de los mensajes RREQ y RREP

I.3.6 Como gestionar el Expiration Time de la Route Discovery Table

Cuando se añade un elemento o algún mensaje hace que se cambie alguno de los campos de uno del los elementos de la tabla se fija su *Expiration Time* a *nwkcRouteDiscoveryTime*.

Cuando se supera el tiempo, el dispositivo debe borrar el elemento correspondiente de la tabla.

Si el elemento correspondiente de la *Routing Table*, tiene un *Status* que no sea *ACTIVE* o *VALIDATION_UNDERWAY* y no hay otra entrada en la *Route Discovery Table* que haga referencia al mismo elemento, debe borrarse también la entrada correspondiente de la *Routing Table.*

I.3.7 Como se gestiona un Link Failure

En la *NWK LAYER* de cada dispositivo existe un contador de fallos con cada uno de los vecinos a los que puede enviar mensajes, si en un momento dado este enlace se considera como fallido,[3] entonces el dispositivo debe responder de la siguiente manera.

Si durante la transmisión de un *Unicast* en alguno de los saltos se detecta un *Link Failure,* el dispositivo que lo ha detectado debe enviar un mensaje al dispositivo que generó la trama, indicando la razón del fallo.

Cuando este mensaje llega al dispositivo que envió el mensaje, o en caso de los RFD, al nodo padre, se elimina de la *Routing Table* la entrada correspondiente a la ruta que origino el fallo. Y en el caso de que el mensaje sea para uno del los nodos hijo se le envía el mensaje a dicho nodo.

En caso de que el mensaje que se iba a transmitir tenga la opción de generar solicitudes de rutas, se comenzará un nuevo proceso de *Route Discovery*

Cuando el enlace que ha fallado es un enlace entre padre-hijo, el dispositivo se lo indicara a las capas superiores, es decir la *APP LAYER*, que se ha perdido en enlace con el padre.

I.3.8 ZigBee Routing cost. Path Cost y LQI

El algoritmo de creación de rutas de *ZigBee* utiliza una medida de coste de la ruta (*Path Cost*), para elección del camino más óptimo durante la creación de camino de datos. Para calcular esta medida, un coste, conocido con el coste de enlace (*Link cost*), asociado con cada enlace en el camino, en cada nuevo enlace se van sumando los constes de enlace acumulados para producir el coste del camino completo.

Para calcular este valor es necesario conocer la longitud del camino y la probabilidad de recepción, que refleja el número de intentos requeridos para enviar un paquete en un link cada vez que se utiliza dicho enlace, la forma más directa de calcular esta probabilidad es utilizar el LQI.

En general el coste se calcula como:

$$C\{P\} = \sum_{i=1}^{L-1} C\{[D_i, D_{i+1}]\}$$

[3] Las condiciones en las que se considera que el enlace está roto, dependen del implementador, pero se recomienda que no se tome esta decisión demasiado pronto dado que las operaciones de reparación de un ruta, suponen un aumento drástico del tráfico en la red, aumentando por tanto la congestión de la red.

Siendo Di, Di+1, el enlace entre 2 dispositivos, y C{l}, el coste de este enlace. El coste es una función entre [0… 7] definida como:

$$C\{l\}\begin{cases} 7, \\ \min\left(7, round\left(\frac{1}{p_l{}^4}\right)\right) \end{cases}$$

Siendo pl4, la probabilidad de recepción. En el estándar de *ZigBee* se dice que la forma más directa para calcular el coste es introducir el valor promediado del *Link Quality Indicator* (LQI).

El estándar de 802.15.4, donde se hace referencia al LQI, deja abierta al diseñador la forma en la que quiere estimar el LQI.

El estándar IEEE802.15.4 define el *Link Quality Indicator* como una caracterización de la intensidad y/o calidad de un paquete recibido.

Según se especifica, esta medida puede ser implementada usando receiver *Energy Detection* (ED), usando estimación de la relación Señal/Ruido o una combinación de ambas.

En el estándar no está especificado cual debe ser usada. La medida debe estar entre 0x00 y 0xFF, para el mínimo y el máximo respetivamente, al menos se deben usar 8 valores diferentes de LQI.

Cost	LQI
1	**254 - 255**
3	**247 - 253**
5	**200 - 246**
7	**0 - 199**

Tabla 4: Relación Coste/LQI

En el caso de *VisualSense,* el coste se puede modelar, de varias formas.

- Definiendo de una forma adecuada la probabilidad de pérdida de un mensaje en el canal (que debería ser recalculada para cada mensaje, o hacerla dependiente de un fuente de ruido externa a la red)
- Con la potencia de la señal, o relación Señal-Ruido que se ve entre dos nodos.

El modelo presentado a continuación, describe las funciones básicas de un módulo *ZigBee* para un dispositivo de tipo FFD (*router*). Las funciones implementadas, son creación de rutas según AODV, tanto suponiendo enlaces simétricos como no. Control de acceso al medio para redes de sin balizas descrito en el estándar IEEE 802.15.4 y envío de mensajes de datos de forma *broadcast* y *unicast,* en redes con topologías de maya.

II. Modelo realizado.

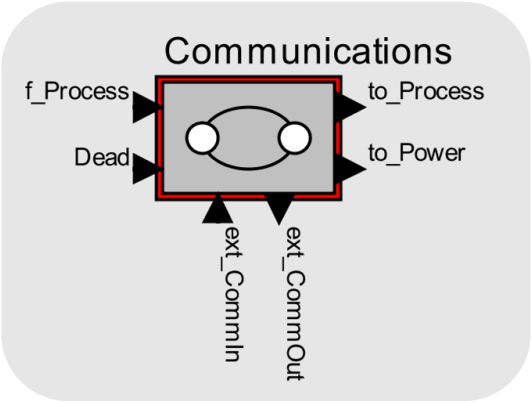


Figura 29: Interfaz del modelo de la capa de comunicaciones.

En la Figura 29 se ve la interfaz creada para el modelo de la capa de comunicaciones, esta cuenta con 2 puertos de entrada, 2 de salida y 2 puertos para comunicaciones externas.

- Entradas:
  - *f_Process:* es el puerto por el que se reciben los mensajes de la capa de procesamiento. En el modelo realizado se reciben sólo los mensajes creados tras procesar la información procedente del modelo de la capa de sensado, no existen mensajes de control y configuración para el modelo.
  - *Dead*: es el puerto por el que se recibe el mensaje de fin de batería del modelo de la capa de alimentación.
- Salidas:
  - *to_Process:* Es el puerto por el que se envían los mensajes recibidos al modelo de la capa de procesamiento.

- Interfaz con el exterior:
  - *ext_CommIn:* Hace las funciones de receptor del módulo de radio.
  - *ext_CommOut*: Hace las funciones de emisor del módulo de radio.

II.1 Nodo Router.

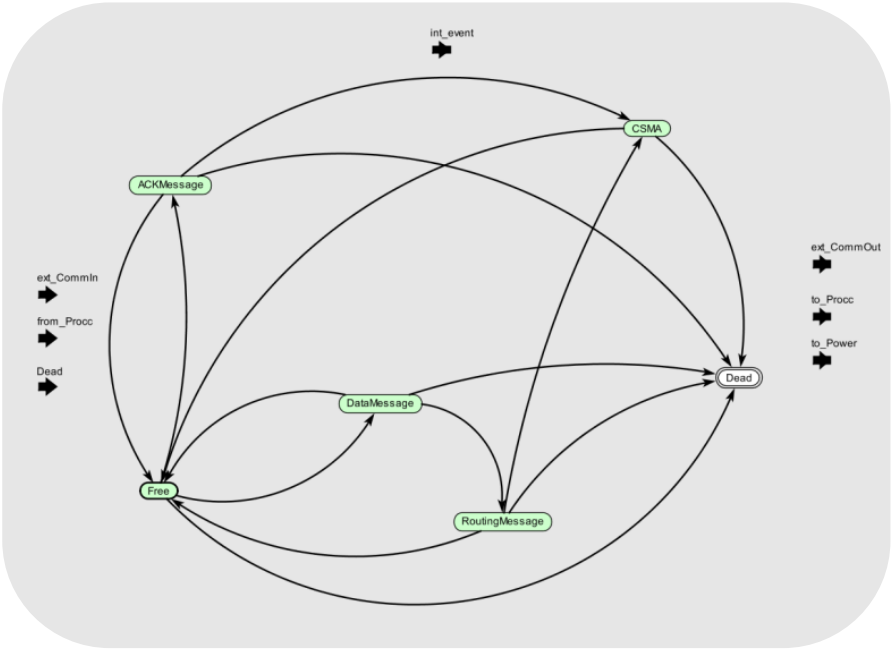


Figura 30: Diagrama funcional del modelo para la capa de de Comunicaciones (router)

En la Figura 30 aparece una representación funcional de la FSM del modelo, en realizada es mucho más compleja, se puede ver completa en el Anexo 2.

En la Figura 31y la Figura 34 aparece la FSM que modela los estados de *CSMA* y *RoutingMessage,* ya que están formados a su vez por pequeñas máquinas de estados.

A continuación se describen las características modeladas en cada uno de los estados que componen el modelo.

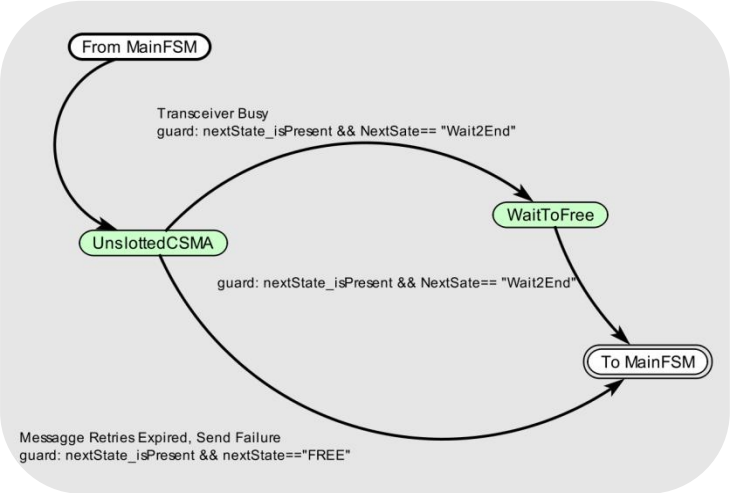


Figura 31: FSM detallada para CSMA

El modelo para el CSMA está basado en 2 estados:

- *UnslottedCSMA*: En la Figura 33 aparece el diagrama utilizado para implementa este el *UnslottedCSMA*, tal como se describió el apartado I.1 de este capítulo.

  La mayor dificultad radica en manejar correctamente los tiempos correspondientes a cada parte del flujo de CSMA dado que en un sistema basado en eventos discretos los eventos en este caso mensajes, por definición, no tienen duración.

  Pare ello se añade la un atributo a los mensajes, la duración que no es más que un número que indica cuánto duraría en realidad, y se opera con ellos para saber si dos eventos con esta duración ficticia coincidirán en a lo largo del tiempo son coincidentes.

  Por ejemplo, poder saber si había actividad en el canal durante el CCA que en el caso de que se produjera antes de empezar con este proceso no se podría detectar sin utilizar esta propiedad ya que el evento se habría extinguido antes de escuchar el puerto de entrada.

  Por otra parte, hay que calcular los valores aleatorios en cada una del las iteraciones en caso de que el medio este ocupado al intentar acceder.

  Para ello existe un generador de eventos en forma de reloj controlado, que crea el primer evento cuando el modelo llega a este estado. Luego espera a que sea disparado, en caso de que se detecte mensajes en el canal de entrada, (*ext_CommIn*), con un lazo de realimentación interno.

En caso de que el medio este accesible se crea un evento con el mensaje en el puerto de salida (*ext_CommOut*).

Cuando el mensaje se envía finalmente al medio, el modelo pasa al estado *WaitToFree.*

- *WaitToFree:* Este estado simplemente crea una ocupación de los canales ficticio basado en el atributo de duración del mensaje que se está enviando. Durante este tiempo el modelo espera a que pueda se termine de enviar el mensaje y así dejar libre el medio para poder volver a recibir o enviar más mensajes. En la Figura 32 se representa el diagrama correspondiente al *refinement* de este estado.

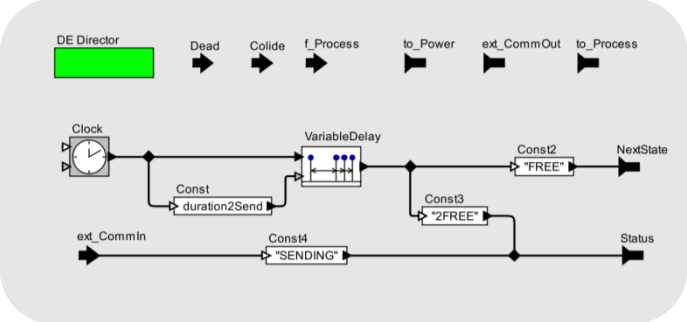


Figura 32: Refinement WaitToFree

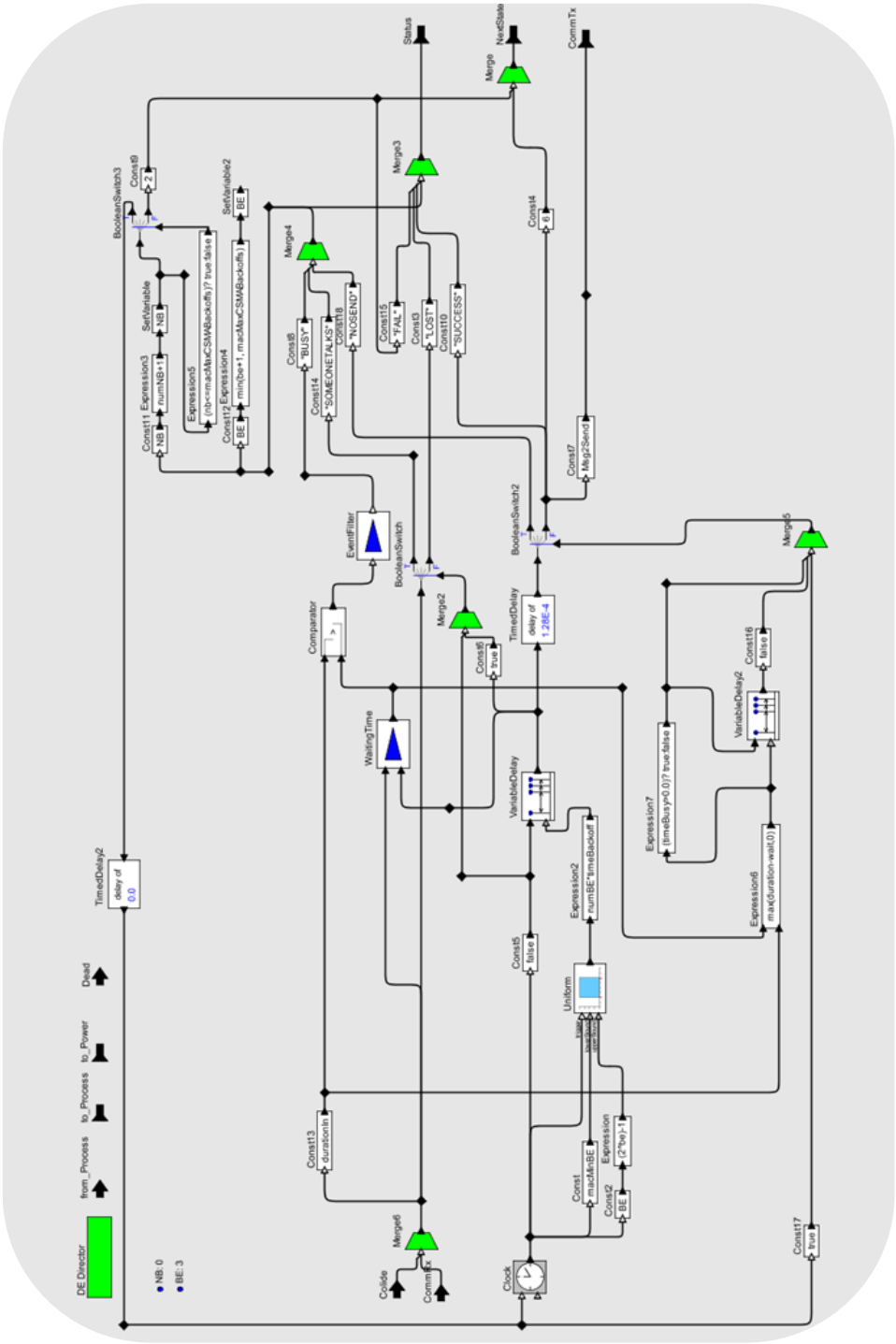


Figura 33: Refinement UnSlottedCSMA

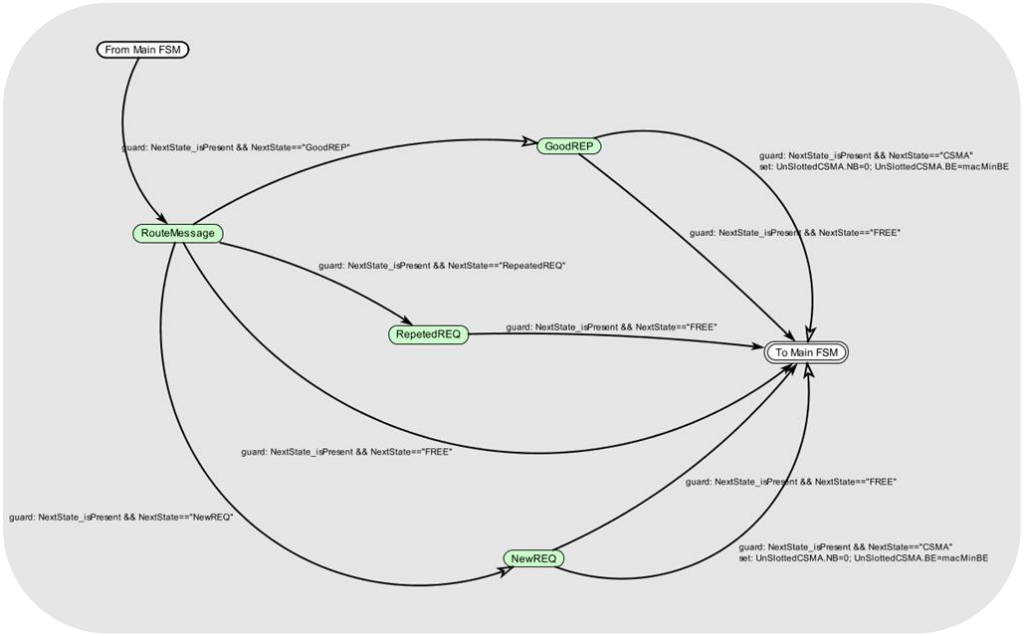


Figura 34: FSM detallada para RoutingMessage

El modelo para creación de rutas está basado en 4 estados, *RouteMesagge, GoodRep, RepetedREQ* y *NewREQ*. Que cumplen con las siguientes funciones.

- *RouteMessage:* Este estado, representado en la Figura 35 sirve para discriminar el tipo de procesado que debe realizarse al mensaje recibido, dependiendo del contenido del mensaje y los datos almacenados en las tablas de solicitudes de ruta activas (*Active Request*).

  Dependiendo de estos valores nos podemos encontrar en 4 casos diferentes, el más básico es en el que no se requiere procesar el mensaje y simplemente se descarta volviendo al estado *Free*, que será presentado posteriormente y simula que el módulo esta libre, y no está enviando o recibiendo datos. Los otros tres se corresponden con los estados listados anteriormente.

- *RepetedRequest*: En caso de recibir un RREQ que ya está en la tabla de *Active Request,* simplemente se comprueba si este nodo desde el que llega dicho mensaje ya existe en la tabla de rutas (*Routing Table*) y si no lo esta se añade.

  El objetivo es conocer todos los nodos vecinos es decir que están al alcance de la radio, que se utilizará en próximas mejoras del modelo, para completar la *NeigbourTable,* descrita en el estándar que contiene información referente al tipo de nodo y calidad del enlace con los nodos vecinos.

Una vez realizado la actualización de los datos, el modelo pasa de nuevo al estado *Free*. La Figura 36 contiene el diagrama implementado en este estado.

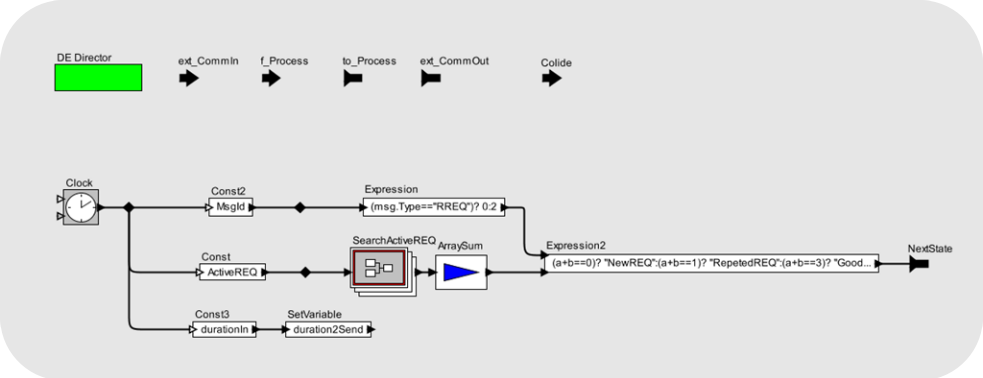


Figura 35: Refinement RouteMessage

- *GoodREP:* Este estado, cuya implementación se representa en la Figura 37, modela el caso en el que se recibe un mensaje de repuesta adecuado a una solicitud de creación de ruta, un RREP.

  Las operaciones realizas son, por un lado actualizar la tabla *Active Request* para borrar la solicitud que se resuelve con el mensaje. Por otro lado, se actualiza la tabla de rutas, para añadir una nueva ruta con el nodo *Responder* a través del nodo que reenvía el RREP.

  Si el nodo que recibió en mensaje no era el *Requester* de la ruta, tiene que reenviarlo hacia el nodo *Requester,* para ello extrae los datos de la tabla de *Active Request.* Crea el nuevo mensaje RREP y el modelo pasa al estado *CSMA* para continuar el proceso.

  Si el nodo que recibe el mensaje es el *Requester* de la solicitud, significa que tiene un mensaje de dato que no pudo enviar al desconocer la ruta por tanto hay mensajes sin entregar en la cola de envió, se extrae el mensaje a enviar y el modelo pasa al estado *CSMA*.

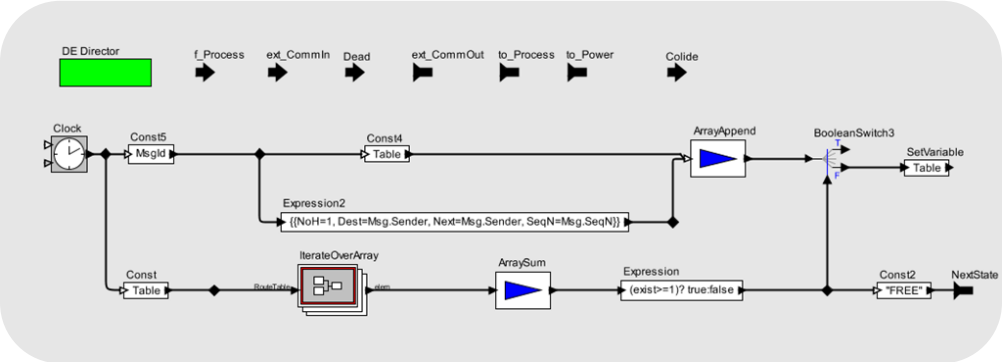


Figura 36: Refinement RepetedREQ.

- *NewREQ:* Este estado modela el caso en el que se recibe un nuevo mensaje RREQ o bien, el nodo no tiene una ruta con el destinatario del mensaje de Datos a enviar.

  En el primer caso, el nodo comprueba si él es el destinatario del nodo, o si conoce una ruta con el destinatario, si es así contesta con un mensaje RREP.

  Si no es el destinatario o no conoce una ruta, reenvía el RREQ al medio, actualizando los campos pertinentes del RREQ.

  Además tiene que añadir a la *Routing Table* una nueva entrada con el *Requester* a través del nodo que reenvía el RREQ.

  Tanto si el nodo es el *Requester* como si es un nodo intermedio hay añadir una entrada a la tabla de *Active Request.*

  El modelo que implementa este estado aparece representado en la Figura 38.

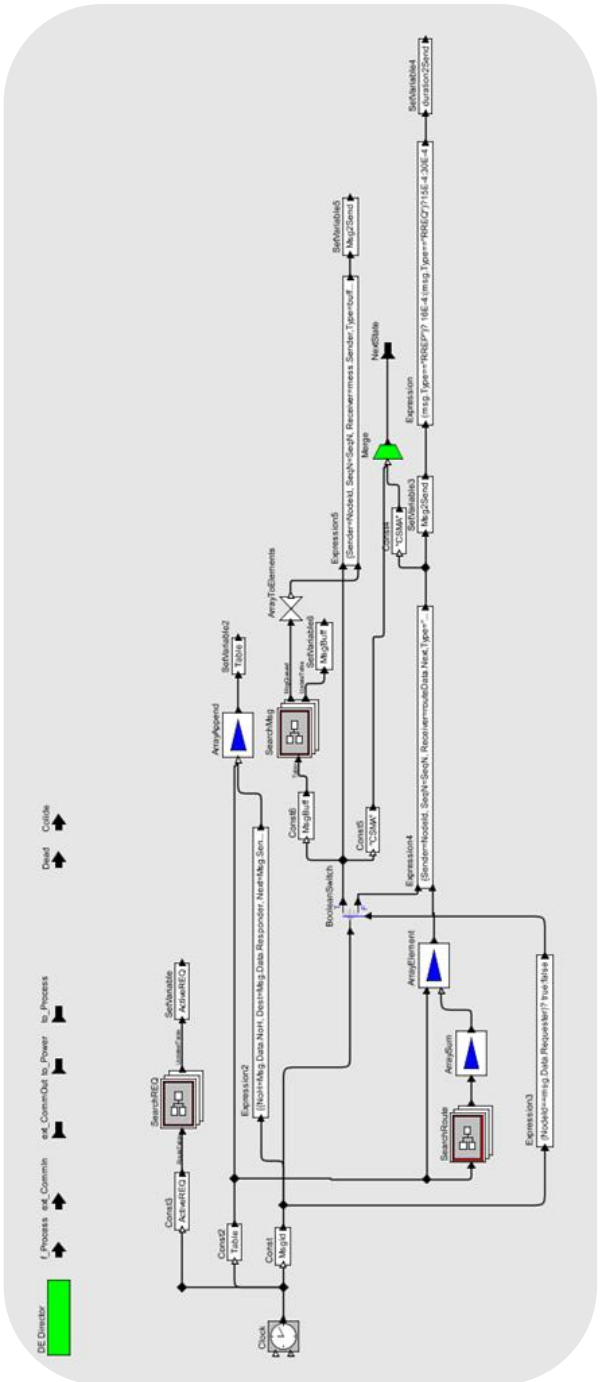


Figura 37: Refinement GoodREP

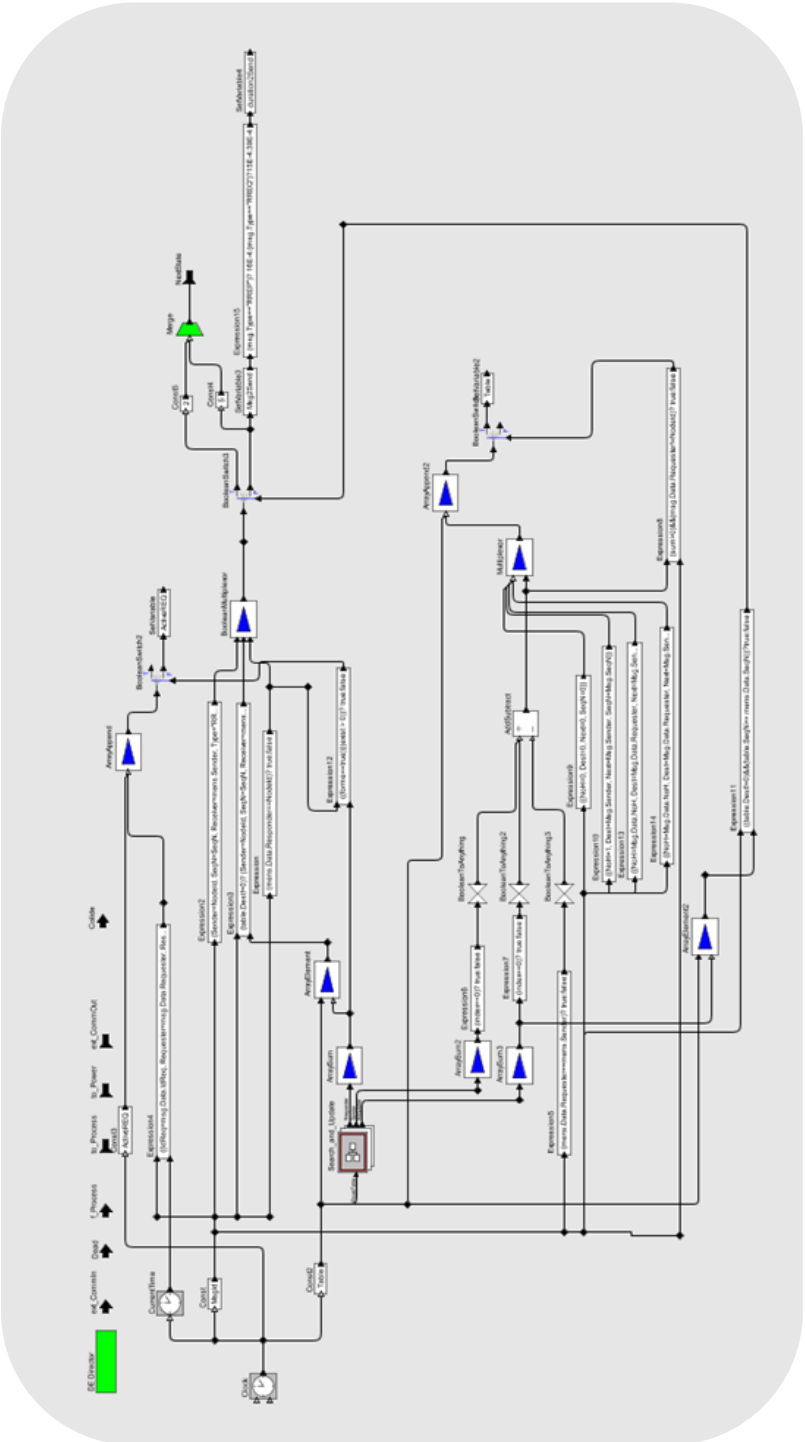


Figura 38: Refienement NewREQ

El resto de estados de la Figura 30, *DataMessage y Free* se implementan con su propio *refinement*, a continuación se describen estos estados. El estado *Acknowledge,* utilizado para procesar mensajes tipo *"MACACK"*, no había sido implementado en el momento de redactar este documento.

- *DataMessage:* Modela el procesamiento de un mensaje de datos, existen varias posibilidades.

  Primero, que se trate de un mensaje de dato recibido del modelo de la capa de procesamiento, en este caso hay que comprobar si el nodo conoce una ruta con el destinatario del mensaje, si es así se procede su envió pasando al estado CSMA.

  Si no existe una ruta activa, hay que crear una solicitud de ruta, por lo que se almacena el mensaje en la cola de mensajes pendientes, y se procede a crear una nueva solicitud de ruta.

  Si el mensaje no es para el nodo, se envía al nodo adecuado para hacerlo llegar a su destinatario. Si no existe ruta activa, se genera un mensaje *RouteError*, que se envía de vuelta al Remitente del mensaje, que al recibirlo, creará un nuevo RREQ con el Destinatario.

  Los mensajes de *ACKNOWLEDGE*, para mensajes de datos son tratados como un mensaje de datos normales.

  El diagrama de procesamiento de los mensajes de datos se presenta en la Figura 39.

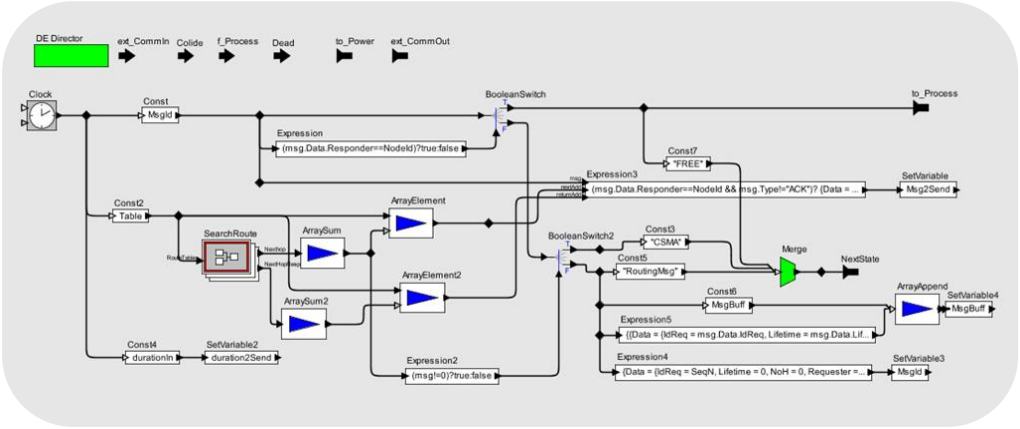


Figura 39: Refinement DataMessage

- *Free*: es el estado que modela al módulo de comunicaciones en reposo, o libre. Cuando se recibe un mensaje se guarda una marca temporal para, generar mensajes de consumo para el modelo de la capa de alimentación tras cada mensaje recibido.

  Con respecto al procesado de la información, hay que comprobar si el mensaje se trata de un *broadcast* o un *unicast*, y si es un *unicast*, si es para el nodo. El tipo de procesamiento a llevar a cado se decide según el contenido del campo *Type*, que marcará el estado al que evolucionara el modelo.

  Además se envía un mensaje tipo "MACACK", que en un futuro se utilizará para realizar lo que se describen en el estándar como *Indirect Acknowledge*, para mensajes tipo *broadcast*, aunque actualmente no está implementada dicha funcionalidad. Este tipo de mensajes no utilizan CSMA, sino que se envían directamente al puerto de transmisión (*ext_CommOut*). La Figura 39 contiene el diagrama que modela este estado.

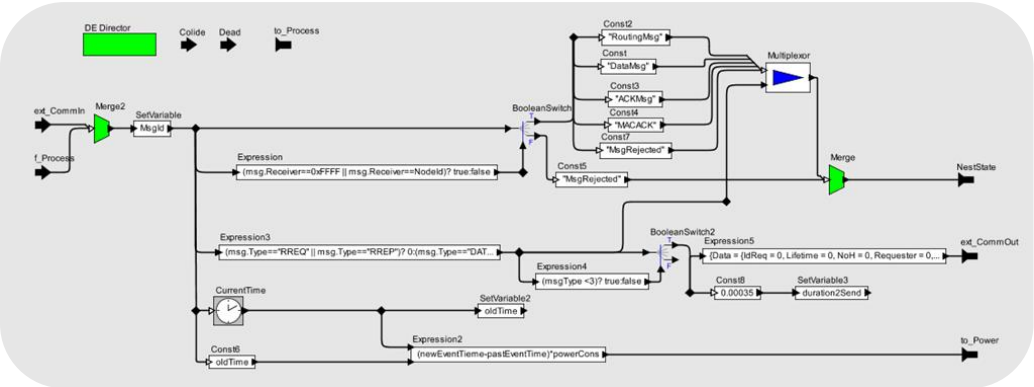


Figura 40: Refinement Free

III. Limitaciones y Líneas Futuras.

El modelo actual, integra partes muy importantes como son el control de acceso al medio a nivel de *MAC LAYER*, y el proceso de creación de rutas entre nodos, perteneciente a la *NWK LAYER*. Pero a pesar de ser un modelo complejo adolece de ciertas características muy importantes como el proceso de creación de redes y los mecanismos de entrada y salida de nodos en una red, etc.

Esto hace pensar que podría ser interesante un enfoque del modelo del protocolo de comunicaciones separado en capas. Esto requeriría un mayor coste computacional y definir una semántica estricta para comunicación entre los modelos de cada una de las capas del protocolo. Aunque si se hacen modelos precisos permitiría, por ejemplo, poder intercambiar diferentes modelos de protocolos que utilicen la misma *MAC LAYER*.

También existe la posibilidad de realizar modelos funcionales para cada una de las operaciones que puede realizar un módulo de, en el caso de este proyecto fin de máster, *ZigBee*. En lugar de crear modelos precisos de la estructura de las capas, realizar modelos independientes de las funcionalidades. Este sería el camino marcado por el modelo realizado en el presente trabajo, pero sería necesario realizar una separación de funcionalidades más clara.

Este tipo de modelos permitiría disponer de sólo una parte de las funcionalidades y simular solamente los aspectos que fueran necesarios en un escenario dado, disminuyendo así el coste computacional. Por ejemplo, simular por un lado la creación de la red, por otro lado la creación de los caminos de datos y poder utilizar los datos obtenidos como datos de partida de nuevos escenarios.

También es muy importante modelar los diferentes tipos de dispositivos en la red, actualmente sólo existe modelos de un FFD, que podría servir también de coordinador, añadiendo la características respecto a la creación de redes. Por tanto sería necesario realizar el modelado de los RFD.

A continuación se presenta un adelanto de dicho trabajo.

III.1 Nodo End Device.

En la Figura 41 aparece la FSM propuesta para el modelo de un RFD, las principales diferencias en cuanto al modelo funcional con respecto al modelo presentado en este capítulo, es la aparición de un estado *Sleep* ya que la principal característica de estos nodos es que están durmiendo la mayor parte del tiempo.

Además el estado de *RoutingMessage* que ha sido sustituido por *PollParent,* ya que un RFD no cuenta con capacidad de rutar mensajes por la red y dado que la mayoría del tiempo este tipo de dispositivos están durmiendo, los mensajes que van dirigidos a los RFDs son almacenados en sus nodos padres.

Para recibir los mensajes pendientes, cuando un RFD se despierta pregunta a su padre si tiene algún mensaje para él. A este proceso se le denomina *Poll Parent.*

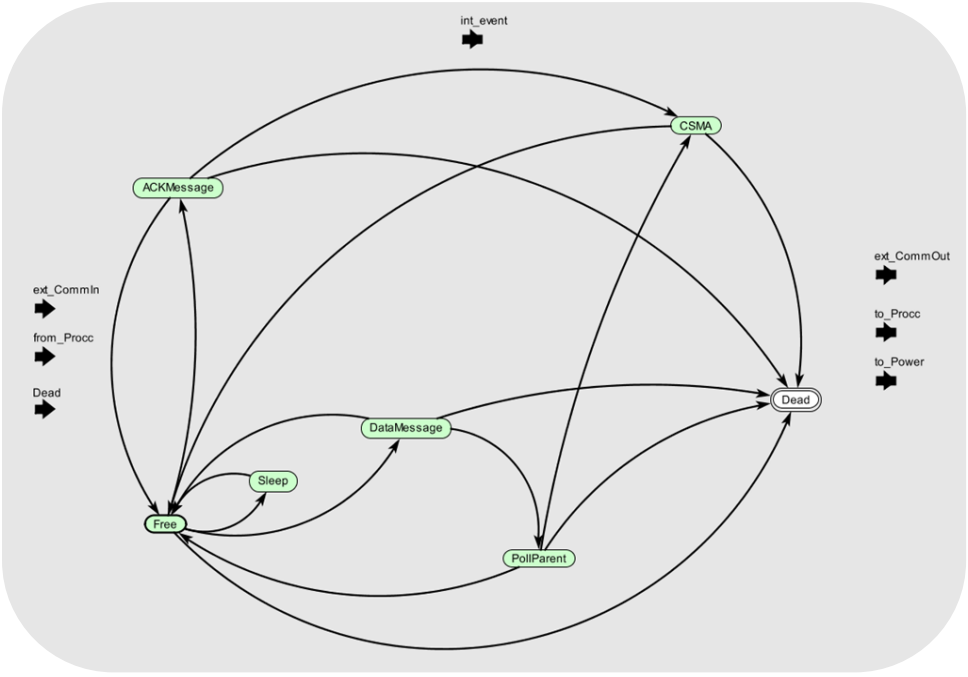


Figura 41: Diagrama funcional del modelo para la capa de de Comunicaciones (RFD)

Capítulo 7: Caso de uso.

A continuación se describe el escenario planteado para realizar las pruebas de los modelos desarrollados y se presentan los resultados obtenidos.

Los resultado obtenidos de la simulación en el estado actual son útiles a nivel de depuración de los modelos y difícilmente podrían ser utilizados para validar los modelos con el comportamiento real de una red, ya que el consumo de cada uno de los módulos que componen el modelo aún no ha sido ajustado, y el modelo de la capa de comunicaciones no incluye todas las características deseable para realizar un validación completa.

Se plantea una red mostrada en la Figura 42, compuesta por 2 nodos sensores alimentados por baterías, 1 nodo *Sink* (receptor de los mensajes de datos, enviados por los sensores), 5 nodos *routers*. Tanto el *Sink* como los *routers* están modelados como si obtuvieran energía de una fuente permanente.

Además hay un bloque que hace las funciones de *sniffer* que permite ver todos los mensajes que recibe y envía el *Sink*, de esta manera se obtiene una comprobación rápida de si las rutas se generan apropiadamente y si llegan los mensajes de datos y se envían los correspondientes *ACKNOWLEDGE*.

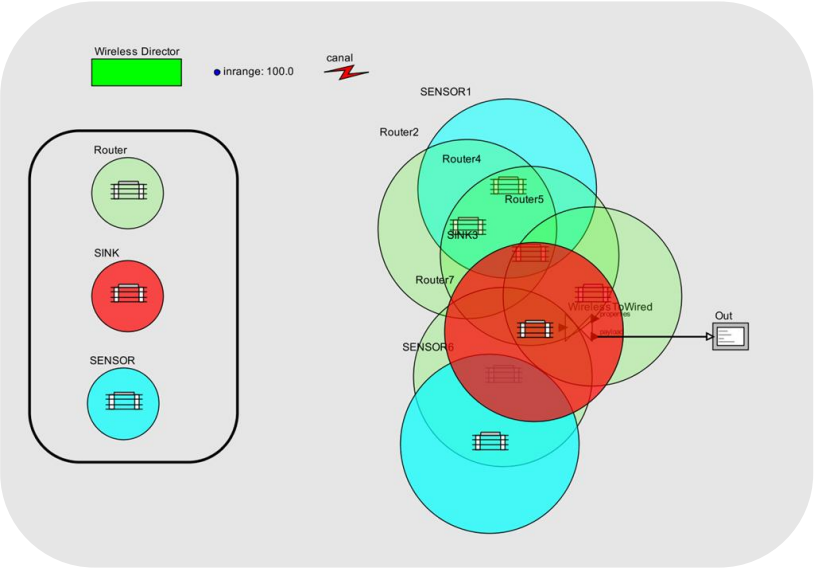
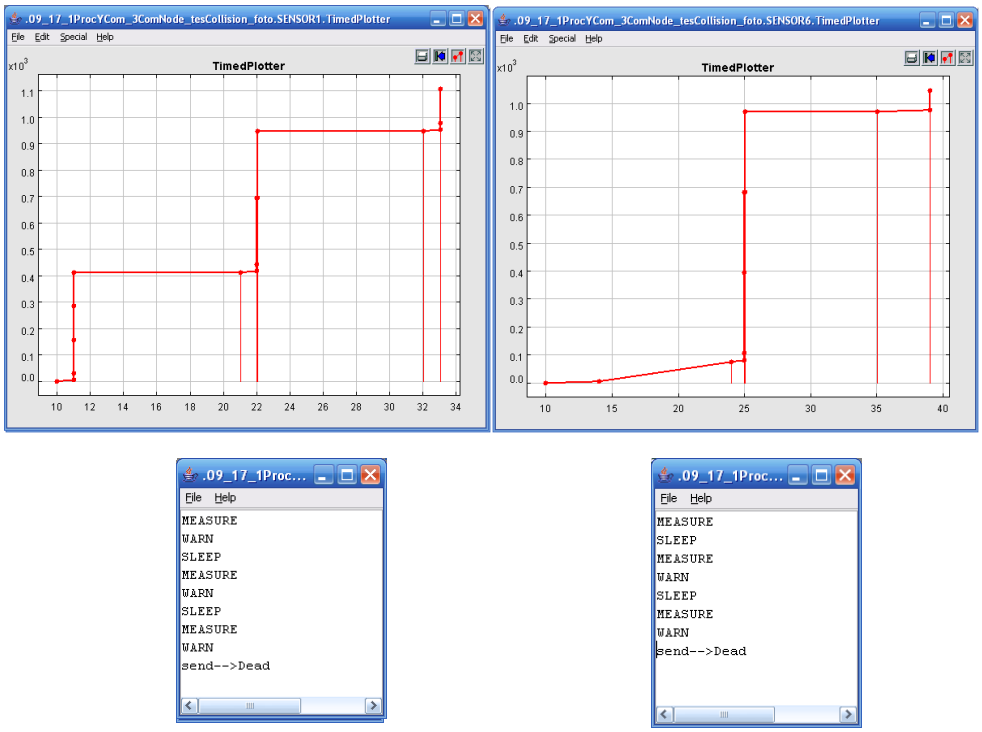


Figura 42: Escenario de Simulación.

Analizando la Figura 43, que muestra las curvas de consumo de los nodos sensores y la evolución de la máquina de estados de la capa de procesamiento, se puede especular con bastante acierto cuál ha sido el comportamiento del escenario simulado.

- Los picos que se observan en la figura, se corresponden con el momento en que se produce un evento de "*WARN*" en la capa de procesamiento, momento en el que se genera un evento en la capa de comunicaciones para que envíe el mensaje a SINK *3*.
- Por tanto si tomamos como referencia esta figura, en el SINK se han debido recibir 3 mensajes de SENSOR 1 y otro de SENSOR 6.
- Pero si nos fijamos en la gráficas de consumo, al producirse el último evento *WARN* en ambos nodos, se supera el valor fijado como tope (1000), y por tanto el nodo pasa a su estado inactivo, y este my probable que este mensaje no sea enviado, por tanto habrá 2 mensajes de datos desde SENSOR 1 y otro desde SENSOR 6.
- Si se ha enviado al menos 1 mensaje desde cada nodo sensor debe existir en la tabla de *Active Request* de cada nodo una solicitud de ruta en estado "*SOLVED*" con *Responder* igual a SINK 3
- Para SENSOR 1, debe ser de 2 saltos, con Router 5 como siguiente nodo, y en el caso de SENSOR 6, debe ser igualmente de 2 saltos, con el Router 7 como primer salto.
- Los nodos Router *2* y *R*outer 7, deben tener en su tabla de *Active Request,* una solicitud "*ACTIVE*", desde SENSOR 1, hasta SINK 3 , dado que esta solicitud nunca será resuelta, dado que la ruta más óptima pasa por Router 4.



a) Sensor 1 b) Sensor 6

Figura 43: Curva de consumo y evolución del modelo de la capa de procesamiento.

En la Tabla 5 se muestra el contenido de la *RoutingTable* y la situación de la tabla de *Active Request,* de los nodos sensores, *Sink y* el nodo Router 2.

Nodo	*Routing Table*	*Active Request*
Sensor1	{{Dest = 0, Next = 0, NoH = 0, SeqN = 0} {Dest = 2, Next = 2, NoH = 1, SeqN = 497} {Dest = 4, Next = 4, NoH = 1, SeqN = 521} {Dest = 3, Next = 4, NoH = 2, SeqN = 510}}	{{IdReq = 0, Requester = 0, Responder = 0, Status = "SOLVED", TimeStamp = 0.0}, {IdReq = 477, Requester = 1, Responder = 3, Status = "SOLVED", TimeStamp = 11.0}}
SENSOR6	{{Dest = 0, Next = 0, NoH = 0, SeqN = 0}, {Dest = 7, Next = 7, NoH = 1, SeqN = 500}, {Dest = 3, Next = 7, NoH = 2, SeqN = 513}}	{{IdReq = 0, Requester = 0, Responder = 0, Status = "SOLVED", TimeStamp = 0.0}, {IdReq = 476, Requester = 6, Responder = 3, Status = "SOLVED", TimeStamp = 25.0}}
SINK	{{Dest = 0, Next = 0, NoH = 0, SeqN = 0}, {Dest = 1, Next = 4, NoH = 2, SeqN = 477},	{{IdReq = 0, Requester = 0, Responder = 0, Status = "SOLVED", TimeStamp = 0.0}}

	{Dest = 4, Next = 4, NoH = 1, SeqN = 521}, {Dest = 5, Next = 5, NoH = 1, SeqN = 497}, {Dest = 6, Next = 7, NoH = 2, SeqN = 476}, {Dest = 7, Next = 7, NoH = 1, SeqN = 500}}	
Router 2	{{Dest = 0, Next = 0, NoH = 0, SeqN = 0 {Dest = 1, Next = 1, NoH = 1, SeqN = 477} {Dest = 4, Next = 4, NoH = 1, SeqN = 521}}	{{IdReq = 0, Requester = 0, Responder = 0, Status = "SOLVED", TimeStamp = 0.0} {IdReq = 477, Requester = 1, Responder = 3, Status = "ACTIVE", TimeStamp = 11.003}}

Tabla 5: Contenido de Routing Table y Active Request.

Por último, en la Figura 44 se muestran los mensajes recibidos y enviados por SINK, donde aparecen resaltados en azul (1), las repuestas a la solitud de rutas y en amarillo (2) los *ACKNOWLEDGE* que envía SINK a los nodos sensores.

Analizando los datos en la imagen y teniendo en cuenta que el identificador de cada nodo coincide con el número que aparece a continuación de su nombre en la Figura 42 se puede afirmar que los resultados coinciden con los esperados. Obteniendo como conclusión más importante que el modelo de la capa de comunicaciones se comporta de manera similar al AODV.

.

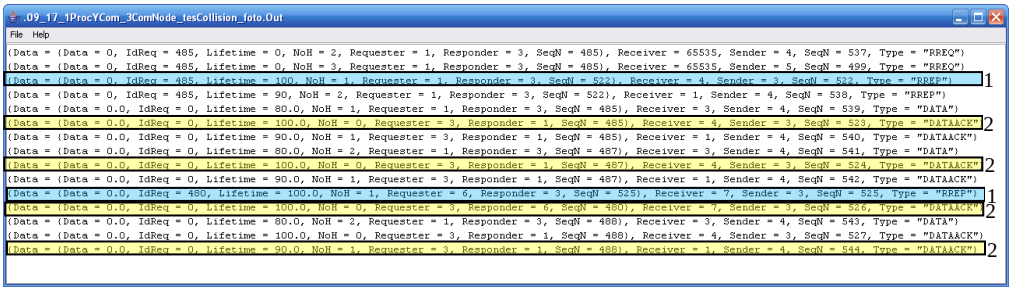


Figura 44: Mensajes enviados y recibidos por SINK

Conclusiones y Líneas Futuras.

En este proyecto fin de máster se ha definido una interfaz de diseño enfocado a las redes de sensores inalámbricas, en particular para la plataforma *hardware Cookies,* desarrollada el Centro de Electrónica Industrial, teniendo en mente reflejar en los modelos implementados la característica más representativa de esta plataforma, la modularidad.

Tras una primera etapa de selección del entrono de trabajo, se eligió *VisualSense* como marco de trabajo, ya que esta plataforma no tiene restricciones a nivel de plataforma *hardware*, sistema operativo, etc. De hecho, el modelo básico, el *Wireless Composite Actor* (Presentado en el Capítulo 1), no es más que una interfaz a la que deben añadírsele las funcionalidades necesarias.

Tomando este bloque como base para la interfaz de un nodo, se desarrollaron 4 modelos independientes entre sí que representan a cada una de las capas que forman un nodo de las *Cookies,* utilizando máquinas de estados "complejas" (denominadas *Modal Models* por *VisualSense)* para definir las funcionalidades básicas de cada una de las capas.

Todos estos modelos han sido presentados, en los Capítulo 3, 4, 5 y 6 de este documento remarcando las limitaciones de cada uno de ellos así como las líneas futuras a seguir en el caso particular de cada uno de ellos.

En general, el hecho de utilizar modelos basados en máquinas de estados es una buena manera de modelar este tipo de sistemas.

Primero porque resulta fácil inferir modelos basados en estados para la funcionalidad de un nodo. Segundo, estos modelos pueden hacerse tan complejos como requiera la aplicación, además si se parte de una interfaz genérica para cada modelo, podría realizarse una biblioteca de modelos que nos permitiera utilizar modelos adaptados a un escenario concreto.

Además, al estar separadas las funcionalidades en diferentes bloques, si alguna de ellas no es necesaria en nodo, simplemente se elimina dicho bloque con el consiguiente ahorro de tiempo de cálculo, por ejemplo, en nodos que sólo realizan funciones de *router* se podría eliminar el bloque de la capa de sensado y si hubiera nodos alimentados de la red eléctrica, podría pensarse en prescindir del bloque referente a la alimentación.

Dado que las redes están formadas por decenas o cientos de nodos, un pequeño cambio en el modelo de los nodos puede suponer un gran ahorro en tiempo de simulación.

En cuanto al modelo del nodo, resaltar el papel crucial del bloque de comunicaciones ya que sin un modelo suficientemente preciso será difícil poder hacer estimaciones apropiadas a la hora de planificar un despliegue.

Sin duda alguna el desarrollo del modelo de comunicaciones ha sido la mayor dificultad encontrada a lo largo del desarrollo de este proyecto, ya que el enfoque de modelo funcional utilizado hace que el modelo se complique exponencialmente cada vez que añaden características nuevas al estar todas ellas muy relacionadas.

Otro problema, con respecto a la herramienta utilizada para el modelado de los nodos, es la dificultad para manejar estructuras dinámicas, como las *Routing Table* y *Active Request,* descritas en el Capitulo 6 y utilizadas por *ZigBee* en el proceso de creación y manteamiento de rutas entre nodos

En general este es el mayor inconveniente con respecto a la plataforma, si bien es muy potente, ya que dentro de un mismo escenario se pueden tener cuantos modelos diferentes queramos, integrar diferentes canales, añadir obstáculos, etc., y contar con una interfaz de usuario relativamente sencilla y fácil de manejar permitiendo gran flexibilidad en cuanto a los tipos de datos que maneja, presenta el gran inconveniente de no contar con bloques de alto nivel ya implementados. En general proporciona modelos y clases con funcionalidades básicas para que el usuario las amplíe y/o combine para formar modelos más complejos.

En el estado actual de los modelos sería muy difícil sacar conclusiones más allá de funcionalidades básicas, véase, saber el camino que seguirán los mensajes de un punto a otro o hacer una estimación con bastante margen de error de la vida útil del nodo.

Por tanto es necesario mejorar la precisión de los modelos, y completar el modulo de comunicaciones. Esto quizá lleve a plantear, al menos en parte, la estrategia de diseño utilizada con el modelo de las comunicaciones, quizá sea oportuno separar las funcionalidades en diferentes modelos, que hiciesen que el conjunto resultara más fácilmente comprensible y reutilizable, algo imprescindible si se pretende continuar trabajando en esta línea.

Este punto es extensible a la semántica de comunicaciones entre modelos, ya que si está bien definida será más fácil crear nuevos modelos y que el diseño resulte más robusto.

También es necesario trabajar en el modelo del entorno, ya que el funcionamiento de una red depende mucho del los elementos que lo rodean así como de las condiciones ambientales.

Para finalizar, resaltar que el contenido de este proyecto pretende aportar los elementos básicos que permitan avanzar en la investigación sobre modelado y simulación de WSNs, con el objetivo final de crear una metodología apropiada que permita planificar con precisión el despliegue de una red de sensores inalámbrica.

Referencias.

[1] MIT Technology Review: http://www.technologyreview.com/article/13060/

[2] UC Berkeley, MLB Co *"Tracking vehicles with a UAV-delivered sensor network"*. July 2000 http://robotics.eecs.berkeley.edu/~pister/29Palms0103/

[3] K. Greene, *"A wireless Sensor City"*, MIT Technology Review, April 2007. http://www.technologyreview.com/communications/18533/?a=f

[4] *"µSWN Integration and Test Results"*. January 2009, FP6-2005-IST-034642 http://www.uswn.eu/

[5] Clarkson University *"[…]Wireless Technology to Monitor Bridge Integrity"* October 2006 http://www.clarkson.edu/news/view.php?id=1566

[6] Proyecto *Tunel-Care*, Marzo 2008, TSI-020100-2008-699 http://www.motas.es/tunel-care/index.html

[7] *ZigBee Alliance*. http://www.*ZigBee*.org/

[8] *Wi-Fi Alliance*. http://www.wi-fi.org/

[9] J. Portilla, A. de Castro, E. de la Torre, T. Riesgo, "A Modular Architecture for Nodes in Wireless Sensor Networks", Journal of Universal Computer Science (JUCS), vol. 12, nº 3, pp. 328-339, March 2006.

[10] *The ns Manual (formerly ns Notes and Documentation)*, Editor: K. Fall, K. Varadhan, January, 2009, http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf

[11] *Mannasim Framework Classes Manual*, The Manna Research Group, February 2006, http://www.mannasim.dcc.ufmg.br/download/mannasim-classes-manual.pdf

[12] P. Levis, N. Lee, M. Welsh, D. Culler, "*TOSSIM: accurate and scalable simulation of entire TinyOS applications*", Proceedings of the 1st international Conference on Embedded Networked Sensor Systems (SenSys '03), Los Angeles, California, USA, November 2003.

[13] S. P. Fekete, A. Kroller, S. Fischer, D. Pfisterer, "*Shawn: The fast, highly customizable sensor network simulator*", Fourth International Conference on Networked Sensing Systems (INSS '07), pp.299-299, June 2007.

[14] G. F. Riley, "*The Georgia Tech Network Simulator*". Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research, August 2003.

[15] ElMoustapha Ould-Ahmed-Vall, George F. Riley, Bonnie S. Heck, Dheeraj Reddy, "S*imulation of Large-Scale Sensor Networks Using GTSNetS*," mascots, pp.211-218, 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005.

[16] B. L. Titzer, D. K. Lee, J. Palsberg, "*Avrora: scalable sensor network simulation with precise timing*", Proceedings of the 4th international Symposium on information Processing in Sensor Networks, April 2005.

[17] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, "*ATEMU: a fine-grained sensor network simulator*", First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, pp. 145-152, October 2004.

[18] Chen, G., J. Branch, M. J. Pflug, L. Zhu and B. Szymanski, "*SENSE: A Sensor Network Simulator*", Advances in Pervasive Computing and Networking, Springer: 249-267, November 2004.

[19] Sobeih, A., Chen, W., Hou, J. C., Kung, L., Li, N., Lim, H., Tyan, H., and Zhang, H. 2005. "*J-Sim: A Simulation Environment for Wireless Sensor Networks*", Proceedings of the 38th Annual Symposium on Simulation, IEEE Computer Society, April 2005.

[20] P. Baldwin, S. Kohli, E. A. Lee, X. Liu, Y. Zhao, "*VisualSense: Visual Modeling for Wireless and Sensor Network Systems*" Technical Memorandum UCB/ERL M05/25, University of California, Berkeley, CA 94720, USA, July 2005.

[21] Ptolemy II project: http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm

[22] C. G. Cassandras, S. Lafortune, "*Introduction to Discrete Event Systems*" 2nd Ed., Springer, 2008, ISBN: 978-0-387-33332-8.

[23] T. Antoine-Santoni, J. Santucci,E. De Gentili, and B. Costa, "*Discrete Event Modeling and Simulation of Wireless Sensor Network Performance*". Simulation 84 Issue 2-3, pp.103-121, February 2008.

[24] F. Maraninchi, L. Samper, K. Baradon, A. Vasseur, "*Lustre as a System Modeling Language: Lussensor, a Case-Study with Sensor Networks*", Proceedings of the International Workshop on Model-driven High-level Programming of Embedded Systems (SLA++P 2007), pp 95-110, June 2008.

[25] Samper, L., Maraninchi, F., Mounier, L., and Mandel, L. 2006. "*GLONEMO: global and accurate formal models for the analysis of ad-hoc sensor networks*". Proceedings of the First international Conference on integrated internet Ad Hoc and Sensor Networks, InterSense '06, vol. 138, May 2006.

[26] *ZigBee Specification, ZigBee* Standards Organization, January 2008.

[27] Wireless Technology Comparison: http://www.dpactech.com/docs/evaluation_support/Wireless_Technology_Comparison.pdf

[28] D. Gislason, "*ZigBee* Wireless Networking", Elsevier Inc., 2008, ISBN: 978-0-7506-8597-9.

[29] *Wireless Medium Access Contro (MAC) and Physical Layer (PHY) Specification for Low-Rate Wireless Personal Area Networks (WPANs)*, IEEE Std. 802.15.4, 2003.

[30] J. Daemen, V. Rijmen, "*The Design of Rijndael: AES - The Advanced Encryption Standard.*" Springer-Verlag 2002, ISBN 3-540-42580-2.

[31] C. E. Perkins, E. M. Rogers, "*Ad hoc On-Demand Distance Vector Routing.*" In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, pp. 90-100, February 1999.

Anexo 1: Información relevante sobre *ZigBee.*

I. Duración de los mensajes RREQ y RREP:

PHY (MHz)	Frequency band (MHz)	Spreading parameters		Data parameters		
		Chip rate (kchip/s)	Modulation	Bit rate (kb/s)	Symbol rate (ksymbol/s)	Symbols
868/915	868–868.6	300	BPSK	20	20	Binary
	902–928	600	BPSK	40	40	Binary
2450	2400–2483.5	2000	O-QPSK	250	62.5	16-ary Orthogonal

Tabla 6: Bandas de frecuencia y tasa de datos.

	Tamaño de campo (en octetos)									
Type	Frame Control	Destination Addr.	Source Addr.	Radius	Sequence Number	Destination IEEE Add.	Source IEEE Addr.	Multicast control	Source Route Sub Frame	Total
RREQ	2	2	1	1	1	0	8	0	0	15
RREP	2	2	1	1	1	8	8	0	0	23

Tabla 7: Tamaño de NWK Header en octetos (del estándar ZigBee).

	Tamaño de campo (en octetos)					
Type	Command Options	Route Request Id	Destination Add	Path Cost	Destination IEEE	Total
RREQ	1	1	2	1	0/8	13 (General)

Tabla 8: Tamaño de NWK Payload para un RREQ (del estándar ZigBee).

	Tamaño de campo (en octetos)							
Type	Command Options	Route Request Id	Originator Add	Responder Add	Path Cost	Originator IEEE	Responder EEE	Total
RREP	1	1	2	2	1	0/8	0/8	7 (General)

Tabla 9: Tamaño de NWK Payload para un RREP (del estándar ZigBee).

	Tamaño de campo (en octetos)							
Type	Frame Control	Sequence Number	Destination PAN id	Destination Address	Source PAN Id	Source Address	Command Type	Total
RREQ	2	1	2	2	2	2	1	12
RREP	2	1	2	2	2	2	1	12

Tabla 10: Tamaño de MAC Header para RREQ y RREP (del estándar IEEE 802.15.4).

	Tamaño de campo (en octetos)			
Type	PHYl	MAC	NWK	Total
RREQ	6	12+2	13+15	48
RREP	6	12+2	23+7	50

Tabla 11: Tamaño total de RREQ y RREP.

- $Duración_{RREQ} = \frac{1}{bitrate} * Size = \frac{1}{250Kbits} * 8 * 48 = 1.5\, ms$
- $Duración_{RREP} = \frac{1}{bitrate} * Size = \frac{1}{250Kbits} * 8 * 50 = 1.6\, ms$

II. Constantes de la NWK LAYER

Constant	Description	Value
nwkcCoordinatorCapable	(Boolean)Indica si el dispositivo puede ser coordinador *ZigBee*. Un valor de 0x00 indica que el dispositivo no puede ser coordinador; un valor de 0x01 indica que puede ser coordinador.	Depende de la configuración
nwkcDiscoveryRetryLimit	El máximo número de intentos que se reinterará un *route discovery*.	0x03
nwkcMinHeaderOverhead	El número mínimo de octetos añadidos por la *NWK layer* al NSDU.	0x08
nwkcProtocolVersion	La versión del *ZigBee NWK protocol* en el dispositivo	0x02
nwkcWaitBeforeValidation	Duración en milisegundos, en el *Requester* del RREQ, entre que se recibe el RREP y se envía un mensaje de validación.	0x500
nwkcRouteDiscoveryTime	Tiempo en milisegundos hasta que caduca un *Route Discovery*	0x2710
nwkcMaxBroadcastJitter	El *jitter time* máximo para un *broadcast* en milisegundos.	0x40
nwkcInitialRREQRetries	El número de reintentos para el primer mensaje RREQ de una ruta.	0x03
nwkcRREQRetries	Idem. Para un nodo intermedio (*router* o *cordinador*)	0x02
nwkcRREQRetryInterval	Milisegundos entre reintentos de retransmisión de un RREQ.	0xfe
nwkcMinRREQJitter	El mínimo *jitter*, en huecos de 2 milisegundos, para el *broadcast* de un RREQ.	0x01
nwkcMaxRREQJitter	El *jitter time* máximo en huecos de 2 milisegundos, para el *broadcast* de un RREQ.	0x40
nwkcMACFrameOverhead	Tamaño de la cabecera MAC utilizado en NWK layer.	0x0b

Tabla 12: NWK Layer. Constantes (del estándar ZigBee).

III. Datos de los mensaje ACK

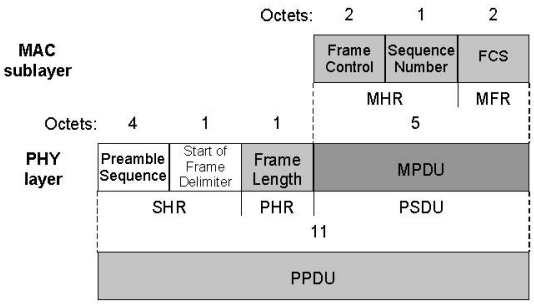


Figura 45: Formato de la trama MAC Acknowledge (de estándar IEEE 802.15.4).

Básicamente responde con el *MAC Sequence Number* del mensaje del que se confirma la recepción.

- Tamaño 11 Octetos (88 bits).
- $Duración_{RREP} = \frac{1}{bitrate} * Size = \frac{1}{250Kbits} * 8 * 11 = 0.35\ ms$

Anexo 2: Modelo completo de la capa de comunicaciones de un nodo *Router.*

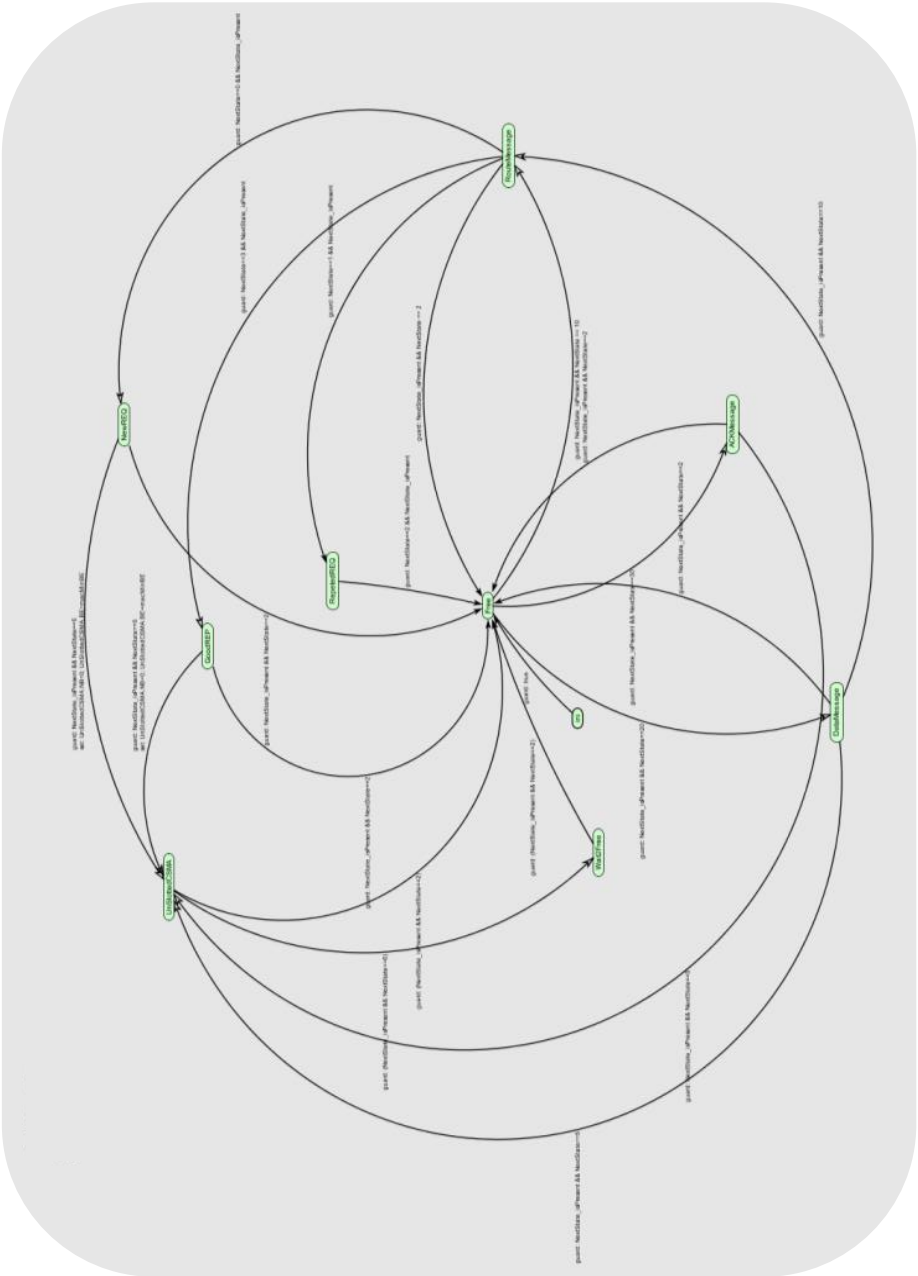


Figura 46: Capa de Comunicaciones. Modelo completo nodo Router.

Índice de Figuras.

Índice de Tablas.