Universidad Rey Juan Carlos

Máster en Redes y Servicios de Comunicación Móviles

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE MÁSTER

SERVIDOR DE ENLACE Y

PROGRAMACIÓN DE DISPOSITIVOS ZIGBEE

PARA UN PROTOTIPO DE

SISTEMA DE LOCALIZACIÓN EN INTERIORES

Autor: Diego Moñino Ramírez

Tutor: Juan José Vinagre Díaz

Co-tutor: Mark Richard Wilby

Curso académico 2008 /2009

ÍNDICE

1 INTRODUCCIÓN 4

2 ESTADO DEL ARTE 6

2.1 TECNOLOGÍAS INALÁMBRICAS Y LOCALIZACIÓN 6
2.1.1 REDES DE ÁREA PERSONAL 7
2.1.1.1 Bluetooth *7*
2.1.1.2 ULP Bluetooth (Wibree) *8*
2.1.1.3 ZigBee *9*
2.1.1.4 UWB (Ultra Wide Band) *9*
2.1.1.5 RFID *10*
2.1.1.6 NFC *12*
2.1.2 REDES DE ÁREA LOCAL 13
2.1.2.1 Wi-Fi *13*
2.1.3 OTRAS TECNOLOGÍAS 14
2.1.3.1 GPS *14*
2.1.3.2 TV *15*
2.1.3.3 Telefonía móvil *16*

2.2 LOCALIZACIÓN BASADA EN *ZIGBEE* 17
2.2.1 FUNDAMENTO DE LA ELECCIÓN 17
2.2.2 REDES DE SENSORES INALÁMBRICOS 18
2.2.3 REDES AD-HOC 19
2.2.4 REDES INALÁMBRICAS MALLADAS 20
2.2.5 IEEE 802.15.4 21
2.2.6 *ZIGBEE* 21

3 OBJETIVOS 24

4 MATERIALES Y MÉTODOS 26

4.1 LOCALIZACIÓN 26
4.1.1 SISTEMA DE LOCALIZACIÓN 27
4.1.2 DISPOSITIVOS *INALÁMBRICOS* 29
4.1.2.1 Modo 'Baliza' *31*
4.1.2.2 Modo 'Móvil' y 'temp-Móvil' *31*
4.1.2.3 Inicialización *31*

4.2 *ZIGBEE* 32
4.2.1 DISPOSITIVOS *ZIGBEE* 32
4.2.2 TINYOS 33
4.2.2.1 Esquema General de una Aplicación TinyOS *33*
4.2.2.2 Envío y Recepción de Mensajes en TinyOS *34*

4.3 PLAN DE TRABAJO 37
4.3.1 DEFINICIÓN DEL PROYECTO 38
4.3.2 PLANIFICACIÓN DEL PROYECTO 38
4.3.3 DISEÑO CONCEPTUAL 39
4.3.4 CODIFICACIÓN DE PROCESOS 39
4.3.5 ELABORACIÓN DE LA MEMORIA DEL PFM 40

5 RESULTADOS 42

5.1 *SERVIDOR DE ENLACE* 42
5.1.1 LAS INTERFACES ICE 43
5.1.1.1 La Interfaz Control *43*
5.1.1.2 La Interfaz Scanner *45*

5.1.1.3 La Interfaz Management *46*
5.1.2 COMPONENTES DEL *SERVIDOR DE ENLACE* 47
5.1.2.1 ZigBee-Manager *47*
5.1.2.2 ZigBee-Dispatcher *48*
5.1.2.3 ZigBee-Connection *49*
5.1.2.4 ZigBee-Server *50*
5.1.3 EL MECANISMO DE PARADA Y ESPERA 51
5.2 MOTAS 52
5.2.1 COMPONENTES *TINYOS* 52
5.2.1.1 MoteManager *52*
5.2.1.2 Parámetros 'Dinámicos': MoteParameters *52*
5.2.1.3 Componente MoteDissemination *57*
5.2.2 MENSAJES 58
5.2.2.1 Atendiendo mensajes de control *59*
5.2.3 PROGRAMACIÓN DE DISPOSITIVOS YA DESPLEGADOS: COMPONENTE DELUGE 60
6 CONCLUSIONES 62
7 TRABAJO FUTUROS 62
8 REFERENCIAS 63
9 ANEXOS 64
9.1 ANEXO 1: DIAGRAMAS 65
9.2 ANEXO 2: INTERFACES ICE DEL SERVIDOR DEL ENLACE 74
9.2.1 BRIDGE SERVER 74
9.2.2 GENERAL MANAGEMENT 80

1 INTRODUCCIÓN

En la actualidad, la localización se ha convertido en necesidad en el día a día de la sociedad debido principalmente a la aparición de una gran oferta de servicios cuyo pilar fundamental es el posicionamiento geográfico del usuario. Alrededor de esta idea y funcionalidad se ha desarrollado toda una pléyade de servicios que se extienden desde los navegadores de cálculo de rutas por carretera que se instalan por defecto en la mayor parte de los vehículos tanto de uso profesional (camiones de transporte de mercancías, autobuses públicos y privados, taxis, etc.) como personal, hasta guías turísticas virtuales que ofrecen al viajero la descripción de monumentos o lugares de interés sin más que aproximarse a ellos.

Sin embargo, la mayor parte de estos servicios comparten una característica común: sólo son utilizables en escenarios de exterior. La razón básica de este hecho es la plena operación de la tecnología *GPS* (*Global Position System*), que permite el posicionamiento de un dispositivo con una precisión en el orden de los metros, a través de una red de satélites. Esta tecnología, que podría resultar a priori cercana a la ciencia ficción debido al coste de operación que supone una red satelital y a la especificidad de su aplicación, ha conseguido penetrar en el mercado precisamente por la demanda de los servicios asociados a la misma y la ostensible reducción de los costes de fabricación de los dispositivos de usuario.

¿Por qué entonces la extensión natural de estos servicios hacia entornos de interior no llega a alcanzar un despegue similar? Evidentemente la causa no puede encontrarse en la ausencia de mercado potencial para aplicaciones basadas en localización. No sólo es fácil imaginar necesidades específicas sino que además éstas serían susceptibles de utilización en distintos entornos:

- *Hospitalario*: para el control de pacientes y la óptima gestión de material portátil, costoso, de uso compartido y crucial en determinadas situaciones (desfibriladores…).
- *Industrial*: para el seguimiento de piezas o equipos en cadenas de montaje de forma que se reduzcan los tiempos de producción.
- *Producción de energía* (gas, electricidad, etc.): para la perfecta operación del personal de mantenimiento, elevado en número de componentes y distribuido en superficies amplias.
- *Logística y distribución*: para la reducción de tiempos de operación, el registro y control del inventariado y la mejora de los procesos automáticos.
- *Seguridad*: entendida bajo dos vertientes, la del control de intrusión en la que la localización juega un papel fundamental para conocer los usuarios permitidos en cada uno de los recintos bajo supervisión, y la de la gestión de las fuerzas de seguridad en entornos amplios como aeropuertos, estaciones de tren, centros comerciales, etc.
- *Turismo*: para la extensión de los servicios de guía turística en entornos de interior como museos, salas de exposición, etc.

La razón para esta falta de impulso en el desarrollo de una serie de bienes y servicios alrededor de la localización en interiores debe buscarse por tanto en algún otro factor que no sea el propio mercado. A primera vista, la tecnología tampoco debería ser un obstáculo para la explosión definitiva de estas aplicaciones ya que muchas de las soluciones de comunicación en plena operación permiten su adaptación para lograr un servicio de localización. *Wi-Fi* y *RFID* son claros ejemplos de estas tecnologías sobre las que ya existen productos comerciales disponibles.

Quizás la única razón que reste por considerar sea que, aunque el mercado está preparado y existe la tecnología que podría implementar las aplicaciones necesarias, aún no se haya encontrado la solución que realmente brinde unas prestaciones suficientes para construir todo un conjunto de servicios que la hagan atractiva para el usuario final. De hecho, las tecnologías citadas anteriormente, adolecen de ciertas desventajas que debilitan su posicionamiento como base para este tipo de servicios. En el caso de *Wi-Fi*, su objetivo principal de utilización como tecnología de acceso hace que algunos de sus requerimientos generales se enfrenten con los específicos de localización. Entre ellos, el más claro es su amplio rango de cobertura, óptimo para su propósito

original, pero que hace difícil conseguir un alto nivel de precisión en la localización. Por su parte, *RFID* presenta como grandes ventajas competitivas la sencillez y el bajo coste, que, sin embargo, hacen que sus capacidades sean limitadas y por tanto tan sólo responda correctamente en aplicaciones donde se requiera conocer el paso de un dispositivo móvil por un determinado punto.

¿Cuál es por tanto la alternativa que haga finalmente viable la producción de servicios basados en localización en entornos de interior? En los últimos años, se está trabajando en una nueva tecnología inalámbrica para la monitorización ambiental, la llamada *ZigBee*. *ZigBee* permite la construcción de una red de sensores inalámbricos de corto alcance y configuración y mantenimiento autónomo, sobre las premisas de bajo consumo, baja complejidad, bajo coste y tasa de información moderada. Es decir, ocupa precisamente el espacio que se extiende entre *Wi-Fi* y *RFID*. El presente Proyecto Fin de Máster utiliza esta tecnología enmarcándose en el desarrollo de una aplicación de localización en interiores, que consigue un sistema en operación con dispositivos disponibles comercialmente, y que ofrece unas prestaciones interesantes en precisión, escalabilidad y adaptación a las condiciones específicas del escenario particular a considerar.

2 ESTADO DEL ARTE

Tal y como se ha referido anteriormente, la localización se ha convertido en los últimos años en fuente de atención para la sociedad, que demanda servicios basados en ella para el desarrollo de sus actividades cotidianas. Esta nueva necesidad ha hecho que la comunidad científica haya abierto una línea de investigación en este ámbito desde el punto de vista tecnológico. En la misma, una primera vertiente se centra en la propuesta de las modificaciones necesarias en tecnologías ya existentes para adaptarlas a los requisitos introducidos por este tipo de aplicación. Por su parte, una segunda vertiente opta por la creación de nuevas tecnologías directamente asociadas a los requerimientos procedentes de las aplicaciones de localización. En cualquier caso, por motivos evidentes, las tecnologías comparten siempre la característica fundamental de ser inalámbricas.

En el presente apartado se describen las distintas tecnologías inalámbricas actuales susceptibles de ser utilizadas con propósitos de localización, así como una breve discusión de las ventajas e inconvenientes de cada una de ellas. Finalmente, se expone *ZigBee* como la alternativa seleccionada en el Proyecto Fin de Máster que aquí se describe, detallando sus características técnicas y los fundamentos de dicha elección.

A diferencia de lo que sucede en entornos exteriores, en los que se ha afianzado el sistema *GPS* como tecnología de posicionamiento prácticamente universal, en los entornos interiores o mixtos (combinación de espacios interiores y exteriores) no existe una tecnología que desempeñe un papel equivalente y proporcione buenas prestaciones en todos los aspectos relacionados con: precisión, robustez, escalabilidad, facilidad de instalación, capacidad de auto-calibración, bajo consumo y tamaño de dispositivos, coste, privacidad, etc. Algunos sistemas utilizan estrategias ultrasónicas y acústicas, otros tecnologías de radio como *GSM*, *UWB*, *WLAN*, *Bluetooth*, *RFID*, *ZigBee*, algunos se basan en técnicas ópticas con infrarrojos y láser y, finalmente, existen soluciones que utilizan estrategias basadas en visión artificial o sistemas inerciales.

2.1 TECNOLOGÍAS INALÁMBRICAS Y LOCALIZACIÓN

Uno de los primeros requisitos para producir una aplicación de localización es que sea posible la interacción del usuario con el entorno. Teniendo en cuenta la definición anterior, esto conlleva algún tipo de comunicación entre dicho usuario y distintos dispositivos de infraestructura desplegados en un área de extensión variable (desde recintos cerrados con dimensiones reducidas hasta amplias áreas al aire libre) a través de dispositivos móviles. Esta comunicación implica un intercambio de información entre el conjunto de dispositivos de infraestructura, los diversos dispositivos móviles de los usuarios y, potencialmente, un repositorio de información remoto, que a su vez podría estar distribuido entre distintos puntos de la red.

Actualmente, algunas tecnologías como *Bluetooth* y *Wi-Fi*, que posibilitan la interacción con otros elementos del entorno, gozan de un gran auge entre un gran número de usuarios debido principalmente a su inclusión en los teléfonos móviles. Sin embargo, no son los únicos que están desarrollándose en la actualidad. Otras tecnologías de comunicación como *Near Field Communication* (*NFC*) o *ZigBee* están apareciendo en los últimos años y posicionándose en este entorno de manera natural. No obstante, la creación de estas nuevas tecnologías no debe entenderse como competidoras de las anteriores sino como soluciones alternativas y complementarias, abriendo el abanico de posibilidades en este ámbito.

A continuación se describen brevemente algunas de las tecnologías inalámbricas susceptibles de ser utilizadas para aplicaciones de localización clasificadas según la extensión del área en que trabajan.

2.1.1 REDES DE ÁREA PERSONAL

2.1.1.1 Bluetooth

La tecnología inalámbrica *Bluetooth* es un sistema de comunicaciones de corto alcance basado en radiofrecuencia que intenta reemplazar a los cables como elemento de conexión entre dispositivos electrónicos fijos y portátiles. Las características principales de *Bluetooth* son su robustez, bajo consumo y bajo coste. Muchas de las características del núcleo de las especificaciones son opcionales, permitiendo una diferenciación de los distintos productos.

El sistema principal de *Bluetooth* se compone de un transceptor de radiofrecuencia, una banda base y una pila de protocolo. El sistema ofrece servicios que permiten la conexión de dispositivos e intercambiar una gran variedad de clases de datos entre los mismos.

Alcance

En la siguiente tabla se muestran la potencia máxima permitida y el rango de cobertura de señal en función de la clase de dispositivo:

Clase	Máxima potencia permitida mW(dBm)	Rango de cobertura
Clase 1	100 mW (20 dBm)	~100 m
Clase 2	2.5 mW (4 dBm)	~10 m
Clase 3	1 mW (0 dBm)	~1 m

Tabla 1. Máxima potencia permitida y cobertura de las distintas clases en Bluetooth

Velocidad de transmisión

Según la versión de especificación *Bluetooth* en que se base la implementación, las velocidades varían entre 1 y 3 Mbps. La nueva especificación propuesta por la *WiMedia-Alliance* augura velocidades entre 53 y 480 Mbps, si bien no existirán a medio plazo dispositivos que soporten estas velocidades.

Versión de la especificación	Velocidad de transmisión
Bluetooth 1.2	1 Mbit/s
Bluetooth 2.0 con soporte EDR	3 Mbit/s
WiMedia Alliance (propuesta)	53 - 480 Mbit/s

Tabla 2. Velocidad de transmisión en función de la versión de especificación Bluetooth.

Rango de frecuencias

Bluetooth utiliza la banda *ISM* (*Industrial, Scientific and Medical*) de uso no regulado a 2,4 GHz (868 MHz en parte de Europa, 915 MHz en algunos países como EE.UU. y Australia, y 2.4 GHz en la mayoría de países del mundo), lo que facilita la consecución de calidad en la señal y la compatibilidad entre transceptores.

Apoyo en otras tecnologías

La versión 2.1 de *Bluetooth* admite el pareado entre dispositivos utilizando *NFC*, reduciendo así los tiempos para establecer la conexión entre dispositivos. Por otra parte, *Bluetooth* se apoyará en

Wi-Fi para el envío de grandes volúmenes de información cuando esté lista la especificación que está preparando el *Bluetooth SIG*, entidad encargada de estandarizar esta tecnología.

Aplicación para la localización

Las aplicaciones de localización sobre *Bluetooth* se basan en un indicador de la potencia recibida llamado *RSSI* (*Received Signal Strength Indication*) para poder realizar una triangulación de las distintas posiciones de los dispositivos de infraestructura dentro del rango de cobertura de los cuales se encuentra un dispositivo móvil.

Ventajas

- Terminales de amplia difusión y funcionalidad extendida a otras aplicaciones en entornos de interior.
- Complementariedad con otras tecnologías que permiten una mayor facilidad para su implantación.

Inconvenientes

- Número de dispositivos en la red muy bajo (8 dispositivos) lo que hace necesaria la utilización de una gran cantidad de subredes que deberán operar de manera conjunta y transparente para ofrecer localización.
- Sincronización local de los dispositivos que deberá ser extendida a otras redes para su utilización como sistema único de localización.

2.1.1.2 ULP Bluetooth (Wibree)

Ultra Low Power Bluetooth es una tecnología diseñada para complementar *Bluetooth*. Nokia es su principal impulsor, si bien otras organizaciones se han unido a la especificación (*Nordic Semiconductor, Broadcom Corporation, CSR, Epson, Suunto y Taiyo Yuden*). La presencia de empresas que se dedican, por ejemplo, a la fabricación de relojes muestra el interés que deposita la industria en lo relativo a dispositivos portátiles de pequeño consumo y poca autonomía a los que se desea añadir la posibilidad de la conectividad inalámbrica.

Los dispositivos con capacidad *Wibree* son capaces de conectarse a sus análogos *Bluetooth*, con una velocidad máxima de transmisión producto de la limitación en el diseño del protocolo orientado a la minimización de consumo energético.

Recientemente, esta tecnología ha cambiado su nombre a *Ultra Low Power Bluetooth* (*ULP Bluetooth*), si bien popularmente es conocida por su nombre inicial.

Consumo

Está orientada principalmente a minimizar el consumo de energía, si bien aún no hay datos específicos.

Rango de frecuencias

Opera en la banda *ISM* de 2.4 GHz.

Velocidad de transmisión

Tiene una velocidad de transmisión en la capa física de 1 Mbps.

2.1.1.3 ZigBee

Se trata de una tecnología basada en el estándar IEEE 802.15.4 para redes de área personal inalámbricas (WPAN) que se marca como objetivo unos dispositivos con importantes limitaciones en cuanto a coste, consumo y tamaño.

ZigBee opera en las bandas de radio *ISM* y se considera que el tipo de nodo *ZigBee* más capaz necesita del 10% del código (en tamaño) incluido en un nodo típico *Bluetooth*. Los nodos con capacidades más reducidas necesitan aproximadamente tan sólo del 2%. A pesar de estas referencias teóricas, la realidad es que las implementaciones *ZigBee* actuales se acercan más bien a tamaños del orden del 50% del código necesario para las implementaciones *Bluetooth*. Este tipo de comparativas pretende indicar la diferencia en cuanto a complejidad entre ambos estándares.

Por otra parte, es una tecnología más apropiada para las denominadas redes de sensores, ya que una red *ZigBee* puede constar de hasta 65535 nodos distribuidos en subredes de 255 nodos, frente a los 8 máximos de una subred *Bluetooth*.

El precio típico de un transceptor *ZigBee* se encuentra en el orden de un dólar, comparado con el precio de referencia de un chip *Bluetooth* de un orden de magnitud en torno a los tres dólares.

Rango de frecuencias

ZigBee utiliza las bandas *ISM*; en concreto, 868 MHz en Europa, 915 en Estados Unidos y 2,4 GHz en todo el mundo.

Consumo

Menor consumo que *Bluetooth*, en comparación. En datos concretos: 30mA transmitiendo y de 3uA en reposo.

2.1.1.4 UWB (Ultra Wide Band)

Tecnología inalámbrica para transmisión de información en un gran ancho de banda (superior a los 500 MHz). La idea inicial es la de implementar sobre este ancho de banda un sistema de transmisión basado en pulsos en el que cada pulso se transmite en todo el espectro del ancho de banda. El tiempo entre pulsos será convenido en función de la aplicación: por ejemplo, radares basados en pulso o sistemas de visión de imágenes pueden utilizar ritmos de repetición bajos (entre 1 y 100 megapulsos por segundo), mientras que sistemas de comunicaciones pueden requerir hasta 200 gigapulsos por segundo.

La modulación de la señal permite cambiar alguno de sus parámetros, de forma que pueda codificarse la información variando la polaridad del pulso, su amplitud y utilizando pulsos ortogonales.

Banda de transmisión

Opera en 7'5 GHz.

Velocidad de transmisión

Consigue tasas entre 100 Mbps y 1 Gbps con alcances de hasta 20 metros.

Aplicación para la localización

Aunque se pretende que el uso de *UWB* no sea exclusivo de sistemas de localización, éste es por el momento el mercado principal de esta tecnología. *UWB* por este motivo representa una situación anómala en este escenario dado que su mayor problema es conseguir extender su uso desde su empleo en localización hacia otros relativos a la propia comunicación entre dispositivos.

La principal razón para este uso de *UWB* casi restringido a aplicaciones de localización es su capacidad de generar precisiones muy altas sin que ello conlleve una densidad elevada de dispositivos en la infraestructura. Además, *UWB* es susceptible de utilizarse en entornos de interior y exterior lo que la hace especialmente recomendable para el desarrollo de aplicaciones de movilidad, sin necesidad de realizar operaciones de conmutación entre tecnologías al cambiar de escenario.

Sin embargo, aún tiene pendiente de resolución algunos problemas de tipo regulatorio ya que, aunque la potencia de transmisión es muy baja, su rango de frecuencias es muy extenso, solapándose con multitud de tecnologías, algunas de las cuales de uso muy extendido como *GPS*, que presentan un nivel de señal muy bajo, que pudiera sufrir de interferencias significativas por *UWB*.

Ventajas

- Prestaciones independientes de la presencia de obstáculos.
- Precisión alta, por debajo de centímetros.
- Baja densidad de dispositivos de infraestructura para conseguir alta precisión.

Inconvenientes

- Posible interferencia con sistemas *GPS*.
- Fuertes restricciones de regulación en Europa y Japón.
- Coste elevado de los equipos.

2.1.1.5 RFID

RFID (R*adio Frequency IDentification*) es un sistema de almacenamiento y recuperación de datos remotos que usa dispositivos denominados *etiquetas*, *transpondedores* o *tags RFID*. El propósito fundamental de la tecnología *RFID* es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio.

Se puede distinguir entre *tags* activos, alimentados por batería, o *tags* pasivos, que emiten utilizando la energía que han recibido del lector.

Existen fundamentalmente 3 tecnologías dentro del *RFID* en función de la banda de frecuencia de trabajo: *LF*, *HF* y *UHF*.

Alcance

El alcance de un sistema *RFID* depende fundamentalmente de la tecnología usada y del entorno en el que se encuentre, sobre todo en el caso de *UHF*, que es absorbida por los líquidos y reflejada y resintonizada por los metales.

Tecnología	LF	HF	UHF	Activo
Alcance	<0,1m	<0,8m	<3m	<200m

Tabla 3. Alcance RFID en función de la tecnología

Estos valores varían dependiendo del tamaño de la antena emisora, la antena del *tag* o del chip que éste incluya.

Rango de frecuencias

Se puede englobar el *RFID* en 4 rangos de frecuencia diferentes, aunque, en algunos casos como con la tecnología *chipless* la diversidad es mayor. Los rangos más frecuentes son:

Tecnología	LF	HF	UHF	Activo
Rango de frecuencias	125~134 KHz	13,56MHz	868 ~928 MHz	2,45 GHz

Tabla 4. Rango de frecuencias RFID en función de la tecnlogía

En el caso de *UHF*, el rango de frecuencias regulado es diferente según el país, aunque todos están dentro del rango arriba indicado.

Aplicación para la localización

RFID se desarrolló como una tecnología para el seguimiento de animales en libertad. Este origen condicionó el tipo de aplicaciones que se han derivado de esta tecnología con el paso de los años, hasta que en la actualidad su uso está limitado a la llamada localización "de paso". Por otra parte, *RFID* no presenta unas prestaciones suficientes como para extender su uso hacia el plano de la comunicación entre dispositivos dado que presenta fuertes restricciones en términos de capacidad de procesamiento y transmisión que limitan drásticamente su ámbito de aplicación.

Así pues, la aplicación de *RFID* para localización es directa. El tipo de aplicación en concreto hará uso de una de las dos opciones que ofrece la tecnología –*tags* activos o pasivos- dependiendo esencialmente del rango de cobertura que sea necesario.

En la actualidad, existe una multitud de aplicaciones comerciales que utilizan *RFID* para ofrecer servicios basados en localización como el *telepeaje* en autopistas, o la apertura de puertas y encendido de motores en automóviles.

Sin embargo, la localización de cierta precisión no parece del todo viable con *RFID* principalmente por su muy corto alcance y la imposibilidad de comunicación entre dispositivos de forma que se permita la configuración de una red mallada.

Ventajas

- Coste muy bajo de los dispositivos.

- Sencillez de los dispositivos y su operación.
- Gran actuación en localización "de paso".

Inconvenientes

- Incapacidad para formar una red mallada.
- Tasas de transmisión extremadamente bajas, que hacen imposible la transferencia de información relevante.
- Capacidad de cómputo muy reducida o inexistente.

2.1.1.6 NFC

Near Field Communication (*NFC*) es una tecnología de comunicación inalámbrica de alta frecuencia y corto alcance, principalmente orientada a teléfonos móviles, que permite el intercambio de datos entre dispositivos a corta distancia (*touchcomputing*).

La tecnología es una extensión sencilla del estándar de proximidad ISO 14443 (comúnmente conocido como *RFID*) que combina la interfaz de una tarjeta inteligente y un lector en un solo dispositivo. Un dispositivo *NFC* puede comunicarse tanto con tarjetas y lectores *RFID* como con otros dispositivos *NFC*. Esto lo hace compatible con infraestructuras de transmisión sin contacto, relacionadas principalmente con sistemas de pago o de conteo (por ejemplo, en transporte público).

Una de las características más importantes en los módulos *NFC* integrados en los móviles es la capacidad que tienen de ser a la vez un sistema lector y un sistema pasivo receptor. Al igual que en *RFID*, existe un modo de comunicación activo y otro pasivo.

Rango de frecuencias

Comunicación mediante inducción de campo magnético donde dos antenas en bucle son ubicadas dentro de sus respectivos campos de proximidad. Opera en la banda *ISM* de radio frecuencia de los 13,56 MHz (*HF*), con un ancho de banda cercano a los 2 MHz.

Alcance

Distancia de trabajo con antenas estándar compactas: inferior a los 20 cm.

Velocidad de transmisión

Velocidad	Modulación del dispositivo activo	Modulación del dispositivo pasivo
424 kBd	Manchester, 10% ASK	Manchester, 10% ASK
212 kBd	Manchester, 10% ASK	Manchester, 10% ASK
106 kBd	Modified Miller, 100% ASK	Manchester, 10% ASK

Tabla 5. Velocidad de transmisión frente a las modulaciones de los dispositivos NFC

2.1.2 REDES DE ÁREA LOCAL

2.1.2.1 Wi-Fi

Wi-Fi es la tecnología que más relevancia ha tenido en las comunicaciones inalámbricas en los últimos años. En 1999 se creó la *WECA* con la finalidad de asegurar la compatibilidad entre equipos y fomentar esta tecnología. Desde 2000 certifica la interoperabilidad de equipos según la norma IEEE 802.11b bajo la marca *Wi-Fi*.

Por lo general en *Wi-Fi* existe un *router* al que se conectan los terminales que cumplen la especificación. El *router* por si solo sirve para establecer una Red de Área Local (LAN, *Local Area Network*), adicionalmente suele configurarse de forma que ofrezca otras funcionalidades. Así las más típicas son actuar como puente entre una red inalámbrica y otra cableada, o también, en conjunción con un módem como una pasarela hacia Internet.

Su auge se ha debido principalmente a dos factores. Por un lado la facilidad para su instalación en redes domésticas. Por otro lado gracias al incremento del mercado de portátiles, los cuales, por sus necesidades de portabilidad, llevan esta tecnología incorporada de serie. Las redes *Wi-Fi* son ampliamente conocidas y existe una multitud de proyectos para desplegar *routers* de este tipo por ciudades, campus, edificios públicos, etc. Muchos de estos proyectos se han ejecutado ya y es habitual hoy día encontrar redes *Wi-Fi* en la mayor parte de las zonas urbanas.

La tendencia parece indicar que la tecnología *Wi-Fi* se integra en dispositivos móviles personales cada vez en mayor medida. Por otra parte, el hecho de que su futuro y el de *Bluetooth* vayan de la mano, pudiendo ser utilizado *Wi-Fi* en ciertas aplicaciones (envío de grandes volúmenes de datos) y *Bluetooth* en otras (envío de pequeña información, bajo consumo, etc.), parece indicar asimismo una futura compatibilidad entre ambas tecnologías.

Rango de frecuencias

Principalmente opera en la banda libre de los 2.4 GHz (802.11b, 802.11g, etc.) La versión 802.11a opera en la banda de los 5GHz, así como también será permitida dicha banda en la versión 802.11n.

Velocidad de transmisión

Hoy en día la versión del estándar 802.11g es la más extendida, contando con una tasa de transferencia de hasta 54 Mbps (brutos). La futura versión (en la actualidad ya se fabrican equipos compatibles con esta versión) 802.11n soportará el doble (108 Mbps), o incluso superior.

Aplicación para la localización

En *Wi-Fi* la técnica de localización más ampliamente usada es la basada en la *RSSI* (*Received Signal Strength Indication*). Esto se debe sobre todo a que muchas de las tarjetas que cumplen con el estándar 802.11, independientemente de la aplicación para la que se usen, proporcionan el valor de esta variable. Con el mismo se puede determinar la distancia a las estaciones base cercanas. A partir de un conjunto de distancias, podremos estimar así la posición del objeto móvil.

Una segunda opción es la de aprovechar el hecho de que la cobertura *Wi-Fi* es cada vez mayor, especialmente en las zonas urbanas. Así, es posible realizar una localización basada en la proximidad a algún *router Wi-Fi*. Un ejemplo de esta alternativa es el sistema *Place Lab* [Place], en el cual se detecta el *router* más cercano a la posición del objeto móvil y se consulta su posición en una base de datos. Finalmente, es también reseñable el sistema de localización *RADAR* [Bahl], que emplea técnicas de análisis del escenario a partir de las potencias de señal de la red *Wi-Fi* medidas en un recinto determinado.

Ventajas

- Tecnología de bajo coste.
- Amplia implantación en el mercado.
- Estándares popularmente reconocidos.

Inconvenientes

- Consumo elevado frente a tecnologías competidoras.
- Precisión moderada.
- Necesidad de recalibraciones frecuentes y problemas de cálculo en presencia de obstáculos.

2.1.3 OTRAS TECNOLOGÍAS

2.1.3.1 GPS

El sistema de posicionamiento global o *GPS* es la tecnología de localización por antonomasia. Aunque ideado por los gobiernos de Francia y Bélgica, el sistema finalmente lo desarrolló Estados Unidos y éste es el país que lo gestiona. Los dispositivos receptores de *GPS* son capaces de calcular su posición con un error del orden de metros, acercándose a centímetros si se combina con *GPS-Diferencial.*

La infraestructura del *GPS* consiste en una red de 27 satélites, de los cuales 24 permanecen operativos mientras que los otros 3 son de respaldo, con un período de vida de siete años y medio. Se encuentran orbitando a 20200 km de la Tierra y siguiendo trayectorias sincronizadas de forma que la superficie de ésta se encuentra cubierta en todo momento. Además, desde tierra existen estaciones que controlan la órbita de éstos y el mantenimiento de la red.

Entre sus aplicaciones se encuentran principalmente la navegación tanto terrestre como marítima, topografía y acciones de salvamento.

En *GPS* la señal transmitida por los satélites es una señal de radiofrecuencia. Se envían dos señales, una pública para uso civil y otra codificada sólo disponible para uso militar. Para uso civil la portadora es de 1575 MHz, mientras que para militar es de 1227 MHz. La precisión es de unos 5 metros en uso civil y alrededor del metro en aplicaciones militares (al ser un sistema desplegado por el Departamento de Defensa de los Estados Unidos, este gobierno se reservaba el derecho de introducir un error aleatorio en la versión civil, con lo que el error podía llegar a ser entre 30 y 100 metros; desde el año 2000 esto ha dejado de utilizarse, dado el gran número de aplicaciones civiles que hacen uso del *GPS*).

Entre las limitaciones más destacadas del *GPS* se encuentra su uso prácticamente exclusivo para exteriores, que se debe a la señal de radiofrecuencia usada por el *GPS*, cuya longitud de onda que le impide atravesar obstáculos. Debido a esto, un receptor *GPS* necesita generalmente una línea de visión más o menos directa con los satélites de los que recibe la señal. Su uso en interiores es por lo general inviable, pero además, en zonas exteriores con un gran número de obstáculos, el nivel de servicio ofrecido puede deteriorarse significativamente. Así suele haber grandes problemas de cobertura en zonas boscosas, zonas de edificios altos, etc. Esto no ha evitado, no obstante, que *GPS* sea el sistema global más utilizado para actividades de localización.

Aplicación para la localización

El cálculo de la posición se realiza mediante la técnica llamada Tiempo de Llegada, *TOA* (*Time Of Arrival*). Para medir con precisión el tiempo de llegada es necesario tener tanto emisor como receptor sincronizados: al usar señales electromagnéticas, un error de 1 milisegundo supone una diferencia de 300 km. Lograr que los receptores *GPS* se encuentren perfectamente sincronizados con los satélites es uno de los principales problemas de este sistema. Dichos satélites están provistos de relojes atómicos a los que incluso se les corrige un desfase del tiempo debido a una menor fuerza gravitatoria. Lógicamente, no se puede dotar de un reloj atómico a los receptores: para subsanar esto se procesa la información proveniente de al menos cuatro satélites, con lo que se consigue una precisión cercana al reloj de éstos.

Existen otros errores en el cálculo debido al marco tan impredecible en el que se usa. Los de menor importancia son los ocasionados por el ruido en los receptores o los errores en los satélites. Otros de mayor relevancia proceden de la velocidad de propagación no constante que presenta la señal al atravesar diferentes capas de la atmósfera. Además, aunque este efecto puede ser modelado, no es posible predecir con exactitud las condiciones climáticas, lo que redunda en una nueva pérdida de precisión. Una vez en la superficie, la señal recibida puede sufrir el efecto del *multitrayecto* con la consiguiente degradación. Por último, para determinar la distancia al satélite, éste envía información sobre su posición orbital, la cual es actualizada por las estaciones terrestres. En este proceso vuelve a añadirse un error debido al desfase de la actualización y la imprecisión en el cálculo de la posición de los satélites.

No obstante, en la actualidad se ha logrado mejorar las prestaciones del sistema utilizando el esquema tradicional de *GPS* con el llamado *GPS-Diferencial*, capaz de corregir la mayor parte de los errores descritos, alcanzando una precisión cercana al metro.

<u>Ventajas</u>

- Receptores de bajo coste.
- Amplia implantación en el mercado.
- Estándares reconocidos.
- Bajo consumo.

<u>Inconvenientes</u>

- Precisión moderada.
- No operativo en interiores.
- Problemas en presencia de obstáculos, en particular, en zonas urbanas con edificios altos.

2.1.3.2 TV

La televisión es un sistema de comunicación para la transmisión y recepción de señales de audio y vídeo. El medio de transmisión utilizado típicamente ha sido las ondas de radiofrecuencia en las bandas de *VHF* y *UHF* en las que se soportan los diferentes canales, teniendo cada una un ancho de banda asignado. Los canales son gestionados por las operadoras de televisión, existiendo en cada país un organismo que regula el espectro electromagnético y concede las licencias de uso.

La transmisión mediante radiofrecuencia emplea repetidores de señal con los que se abarcan extensas áreas. No obstante, con este medio se sufre una fuerte degradación de la señal, llegando a

ser crítica en muchos casos. En zonas rurales alejadas de repetidores o en la "zona de sombra" de éstos, es necesario el uso del satélite. La difusión de una señal digital proporciona robustez además de ciertas funcionalidades de las que carecía la señal analógica. En Europa se utiliza en el sistema DVB (llamado TDT en España), que funciona en las mismas bandas que la analógica pero con tal optimización que se permiten cuatro canales donde antes había uno solo analógico. Para una comunicación eficaz, se sincronizan los repetidores valiéndose de la señal de reloj del sistema *GPS*.

Aplicación para la localización

En sistemas de localización, la TV es una tecnología muy poco explorada. No ha sido concebida para estas tareas y en general siempre han existido alternativas como el *GPS* más orientadas en ese sentido. Sin embargo, la señal recibida por los repetidores de TV puede usarse para realizar un posicionamiento aproximado. La colocación de varios receptores de señales de TV en una zona geográfica permite el análisis de la diferencia de temporización entre los mismas y por tanto el conocimiento de las características de la señal en esa área (es decir, sería posible la construcción de un "mapa de señales" de la zona). La localización se implementaría mediante técnicas de análisis del escenario con las que un dispositivo móvil que incorpore receptor de señal de TV puede procesar la información de sincronización y determinar así su posición. El sistema comercial *Rosum TV-GPS* [Rosum] incluye el diseño de un sistema basado en esta filosofía, que, además se combina con la tecnología *GPS* para lograr una mayor exactitud en exteriores.

Ventajas

- Infraestructura necesaria ya desplegada.
- Consumo bajo.
- Frecuencia menor que *GPS*, ya que el sistema de TV está diseñado para alcanzar interiores, atravesar paredes, edificios, mientras que *GPS* requiere utilización en espacio abierto.
- Ancho de banda mayor, con lo que es posible conseguir mayor exactitud en el posicionamiento.
- Potencia usada por los repetidores de TV mayor que la de los satélites *GPS* y se encuentran más cerca de los receptores, por lo que la señal recibida es más fácil de detectar.

Inconvenientes

- Precisión moderada
- Muy escasa implantación en el mercado (limitado a escenarios muy concretos).

2.1.3.3 Telefonía móvil

La telefonía móvil es un sistema de comunicaciones de voz y datos basado en señales transmitidas por radiofrecuencia, lo que permite la movilidad de los terminales. El sistema está formado por una red de comunicaciones inalámbricas con estaciones base y los terminales que acceden a la misma. La telefonía móvil ha evolucionado fuertemente desde sus comienzos. Las diferentes evoluciones se conocen como "generaciones".

Aplicación para la localización

El uso de la telefonía móvil (especialmente de *GSM*) para la localización está siendo objeto de estudio desde hace bastantes años. Dado que en la telefonía móvil se definen celdas (regiones) hexagonales controladas por una estación base, es posible aplicar técnicas de localización por proximidad. Estas celdas pueden tener un radio de cientos de metros, siendo éste inversamente proporcional a la demanda en la zona que cubren: a mayor demanda, menor tamaño de celda y por tanto, mayor precisión en la localización.

En los sistemas de segunda generación y en adelante, cada teléfono móvil notifica a la estación base la celda en la que se encuentra. Por lo tanto, una primera aproximación de la localización se puede implementar viendo en qué celda se encuentra un móvil y por tanto la zona en la que está. Un refinamiento mayor se consigue mediante la medición de la *RSSI*: las estaciones base aledañas también detectan la señal del móvil y por la intensidad de ésta se puede aproximar la distancia. Conocidas varias distancias, puede determinarse la posición del móvil por triangulación.

En cualquier caso, la precisión alcanzada es baja dado que es impredecible el número de obstáculos que puede atravesar la señal. Por tanto, como sistema de localización sólo tiene viabilidad en exteriores, donde la precisión ronda las decenas de metros. El sistema *Place Lab* [Place] mencionado anteriormente combina el uso de *Wi-Fi* en interiores con el de *GSM* en localizaciones exteriores, proporcionando así amplia cobertura.

Ventajas

- Terminales de bajo coste y de amplia difusión.
- Amplia implantación en el mercado.
- Estándares popularmente reconocidos.
- Bajo consumo.
- Detección de presencia inmediata.

Inconvenientes

- Precisión del orden de decenas de metros.
- Muy baja precisión en interiores.
- Detección de presencia con precisión de cientos de metros

2.2 LOCALIZACIÓN BASADA EN *ZIGBEE*

2.2.1 FUNDAMENTO DE LA ELECCIÓN

La localización de objetos o personas en entornos de interior presenta cuatro requerimientos principales para obtener unas prestaciones de garantía:

- Dispositivos inalámbricos de tamaño reducido y fácil implantación.

- Capacidad de comunicación entre dispositivos móviles y fijos y fijos entre sí de tal forma que se construya efectivamente una red de comunicaciones de tipo mallado entre los mismos.
- Precisión en el entorno del metro, dependiente de la aplicación en concreto.
- Adaptabilidad al escenario particular tanto en la precisión requerida como en la estructura del área a cubrir. En este sentido, es necesaria la escalabilidad de la solución de forma transparente a la aplicación en operación.

Estos requerimientos restringen la búsqueda de tecnologías principalmente a aquéllas capaces de operar en entornos de interior por lo que alternativas como *GPS* quedan descartadas desde el primer momento. Por otra parte, alternativas como la TV o la telefonía móvil no son capaces de ofrecer los requisitos relativos al tamaño de los dispositivos o la precisión necesaria.

Por su parte, *RFID* presenta una incapacidad para formar redes malladas, lo que limita enormemente sus prestaciones para este tipo de aplicaciones. A su vez, la solución basada en *Bluetooth* presenta el inconveniente citado anteriormente sobre el número reducido de dispositivos dentro de una misma red, lo que no la hace especialmente indicada para cumplir el requisito de escalabilidad impuesto.

Finalmente, aunque existe una multitud de trabajos de investigación e incluso desarrollos comerciales de localización basados en *Wi-Fi*, se observa que, aun admitiendo precisiones suficientes para ciertos escenarios, esta tecnología no presenta la capacidad de adaptación a la situación específica bajo estudio. En primer lugar, la disposición de los puntos de acceso y su reducido número, no posibilita el ajuste de la red a las particularidades del recinto. Adicionalmente, no existe una relación directa entre la precisión de la aplicación y el número de puntos de acceso desplegados por lo que se dificulta en gran medida la escalabilidad de la solución propuesta.

De este modo, las tecnologías que podrían ajustarse de forma óptima a los requisitos descritos, son *UWB* y *ZigBee*. Aun a pesar de que la precisión teórica alcanzable es mayor en *UWB* que en *ZigBee*, la elección final de esta última se debe principalmente a la existencia de un desarrollo tecnológico y comercial pleno en la actualidad junto con los problemas regulatorios que presenta la primera.

En los siguientes apartados se presentan las características principales de esta tecnología.

2.2.2 REDES DE SENSORES INALÁMBRICOS

Gran parte de las limitaciones que existen en la actualidad para extender el paradigma de la Inteligencia Ambiental (AmI) a todos los entornos del hábitat construido provienen del elevado tamaño de los dispositivos.

En este contexto surge la tecnología emergente de Redes de Sensores Inalámbricos (RSI), que tras varios años de experimentación y refinamiento en laboratorios de investigación, está haciendo su primera aparición en el mercado a través de una oferta de componentes y sistemas aptos para la sensorización y/o actuación densa en áreas a gran escala y a bajo coste. Algunas de las características generales básicas de las RSI se exponen a continuación:

- Redes en malla (*mesh*), basadas en dispositivos de radiofrecuencia, llamados comúnmente "motas", de muy pequeña dimensión (cm^2 o mm^2) y bajo coste.
- Construidas con componentes electrónicos disponibles comercialmente (*COTS*, *Common Off The Shelf*) conformes con estándares y fabricados en serie.
- Elementos con tecnología de fabricación *MEMS* (*Micro Electro-Mechanical Systems*), con un factor de forma del orden de cm^2 en evolución hacia dimensiones de mm^2 a medida que se van incorporando nuevas tecnologías de nanofabricación de semiconductores.

- Integran una amplia variedad de sensores (mecánicos, electromagnéticos, químicos) asociados a las motas, dando lugar a sistemas de sensorización densa, de fácil despliegue y autoconfiguración.
- Sin necesidad de cableado, se pueden aplicar a la detección y colección de datos en una amplia variedad de entornos, con carácter temporal o permanente.
- Alcance de un enlace en el rango de los 100 metros. La cobertura física total puede extenderse al entorno de los kilómetros utilizando esquemas multi-salto.
- Capacidad de transmisión variable según tecnologías, desde los cientos de kbps.
- Estándar oficial IEEE 802.15.4 y certificación *ZigBee*.

2.2.3 REDES AD-HOC

Estas RSI pueden observarse como un grupo de la categoría superior denominada Redes Ad Hoc Inalámbricas (RAHI). Una RAHI es una colección de nodos autónomos que se comunican entre sí mediante enlaces radio, donde no existe una infraestructura de red fija y la administración se realiza de forma descentralizada. Los nodos pueden tener libertad de movimiento y las comunicaciones entre los mismos se producen mediante técnicas multi-salto en las que los mensajes son retransmitidos por los nodos intermedios del camino hasta llegar a su destino final. En este nuevo entorno, los nodos participan en la toma de decisiones, realizando las funciones propias del mantenimiento de la red y tomando parte en los algoritmos de encaminamiento.

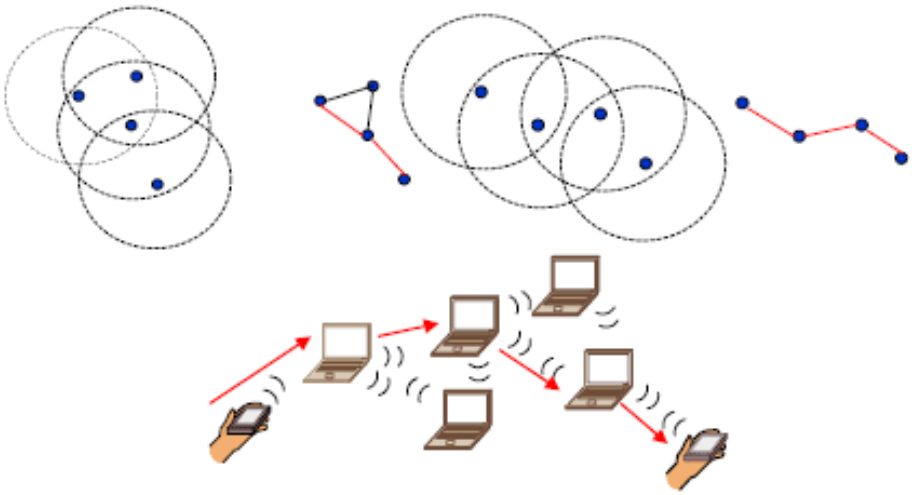


Fig 1. Redes Ad Hoc Inalámbricas: grafos de enlaces y propagación multi-salto

En general, cualquier propuesta real aplicable a una RAHI deberá tener en cuenta las restricciones impuestas por sus características inherentes como topología dinámica, enlaces de ancho de banda limitado y capacidad variable, limitaciones de energía y capacidad de procesamiento en los nodos y seguridad física limitada.

Hasta ahora, los esfuerzos de investigación se han centrado principalmente en temas relacionados con el encaminamiento. Sin embargo, existen otros aspectos no menos importantes que deben ser abordados en el diseño de una red de elevada funcionalidad y disponibilidad, como pueden ser los mecanismos de provisión de calidad de servicio (*QoS*), seguridad y descubrimiento de servicios.

Los requisitos de las actuales aplicaciones hacen necesario que la red deba proveer cierto nivel de *QoS* al usuario. Si bien este problema está prácticamente resuelto en redes fijas, las especiales características de las redes móviles ad-hoc hacen necesario un nuevo estudio para afrontar este problema. Principalmente, la topología dinámica y los escasos recursos de los nodos hacen necesario que la carga del mecanismo de provisión de *QoS* sea lo más ligera posible, en cuanto a carga de procesamiento (CPU), como de recursos de red (ancho de banda). El soporte de esta carga depende de cuatro componentes básicos: modelo, señalización, encaminamiento y acceso al medio. La cooperación de todos estos componentes determinará la capacidad y rendimiento del mecanismo de provisión de calidad de servicio.

Con respecto a la seguridad, debe destacarse que sus requisitos son especialmente difíciles de abordar en una RAHI, debido fundamentalmente a las características propias de su naturaleza. Estas características imponen una serie de restricciones que hacen que las técnicas de seguridad empleadas en redes convencionales no puedan ser aplicables directamente en este nuevo entorno. Por otro lado, la elaboración de una política de seguridad de carácter general es un problema de gran complejidad. Habitualmente, las técnicas de seguridad a introducir dependerán de la aplicación concreta para la que se realiza el despliegue de la red. Tres de los aspectos claves que deberán ser cubiertos por cualquier política de seguridad aplicable en una red ad-hoc: detección de intrusiones, encaminamiento seguro y gestión de claves.

En cuanto al descubrimiento de servicios, este aspecto es vital en una red formada por múltiples equipos cada uno con características muy específicas y debiendo cooperar y conocerse entre sí para ser de utilidad al usuario. En redes con infraestructura este problema cuenta con diversas soluciones y está tan sólo a la espera de seleccionar aquélla que finalmente cuente con un carácter universal. No obstante, en RAHI deben cubrirse múltiples aspectos. Existen dos corrientes que abordan este problema: las basadas en modelos directos y modelos mediados. Las primeras pretenden dar respuesta al problema de reducir el número de mensajes y conseguir mecanismos de difusión eficientes. Las segundas deben proponer formas para replicar los nodos servidores y distribuir la carga entre ellos. Actualmente existen distintos protocolos de descubrimiento de servicios pero, en general, enfocados a redes con infraestructura estable y poco adaptados a RAHI.

2.2.4 REDES INALÁMBRICAS MALLADAS

Las redes inalámbricas malladas consisten en RAHI multisalto en las que cada nodo puede enviar y recibir mensajes, además de tener la capacidad de funcionar como encaminador, reenviando mensajes de sus vecinos. A través del proceso de reenvío, el paquete de información encontrará su camino hacia su destino, pasando a través de nodos intermedios.

Estas redes permiten una gran escalabilidad y adaptabilidad, gracias a sus posibilidades de autoconfiguración y la facilidad para añadir o eliminar nodos.

Los parámetros más importantes en el diseño de este tipo de redes son los siguientes:

- *Bajo consumo de potencia*: las aplicaciones de redes de dispositivos inalámbricas necesitan menor consumo de potencia que las que ofrecen tecnologías actualmente disponibles como *Bluetooth*. Hay que tener en cuenta que en una red en el que exista un gran número de nodos, el cambio frecuente de baterías es impracticable, y en el futuro se tendrá que optar por fuentes de energía que la generen a partir del entorno.
- *Bajo coste*: para hacer posible el despliegue de redes de dispositivos inalámbricos con un gran número de nodos es necesario también que el coste de cada nodo sea bajo, porque de lo contrario se haría imposible cualquier aplicación basada en este tipo de redes.
- *Interoperabilidad*: para maximizar la producción, marketing, ventas y la distribución eficiente de productos basados en redes de dispositivos inalámbricos, permitir la interoperabilidad entre aplicaciones, y evitar el establecimiento de variantes regionales, es deseable producir dispositivos capaces de operar a nivel global.

- *Seguridad*: la seguridad en redes de dispositivos inalámbricos tiene dos facetas de máxima importancia: la seguridad real de la red y la percepción de la seguridad por parte de los usuarios.

2.2.5 IEEE 802.15.4

Para facilitar el volumen de producción esperado de ese tipo de sistemas y dispositivos, minimizando de esta forma el coste de los componentes de la red, es necesario el desarrollo de protocolos de comunicación estandarizados. El *IEEE 802 Local Area Network/Metropolitan Area Network Standard Commitee* (LMSC) creó el *Working Group 15* para desarrollar un conjunto de estándares para WPAN.

Con objeto de abordar las necesidades de redes inalámbricas de bajo consumo de potencia y coste, en diciembre de 2000, el *IEEE New Standard Committe* (NesCom) oficialmente instituyó un nuevo *Task Group* en el *Working Group 15* para comenzar el desarrollo de un estándar para WPAN de baja tasa de transmisión (LR-WPANs), llamado 802.15.4 [Moon]. El objetivo de este *Task Group 4* fue el de proveer de un estándar de baja complejidad, coste y potencia para la conectividad inalámbrica entre dispositivos baratos, fijos, portátiles y móviles. El alcance de este grupo, al igual que para todos los estándares inalámbricos IEEE 802, fue limitado a la creación de especificaciones de la capa física y del control de acceso al medio (*MAC* - subcapa de la capa de enlace de datos) definidas en el modelo de referencia OSI; siendo requerido que el estándar fuera compatible con el la capa del Control de Enlace Lógica 802.2. El estándar fue aprobado en mayo de 2003.

Las principales características de IEEE 802.15.4 son:

- Tasas de transmisión de 250 kbps, 40 kbps y 20 k*GPS*.
- Funcionamiento en estrella y *peer-to-peer*.
- Soporte para dispositivos de baja latencia.
- Acceso al canal mediante la técnica CSMA-CA.
- Direccionamiento de dispositivos dinámico.
- Protocolo "*Handshake*" para transferencias fiables.
- Bajo consumo de potencia.
- Bandas de frecuencia:
    - 16 canales in la banda *ISM* de 2.4 GHz.
    - 10 canales in la banda *ISM* de 915 MHz. (Sólo en US)
    - 1 canal en la banda europea de 868 MHz.

2.2.6 *ZIGBEE*

Tomando como base este estándar IEEE 802.15.4, la *ZigBee Alliance* tiene el objetivo de proveer un sistema abierto de comunicaciones global que permitiera productos de control y monitorización basados en redes inalámbricas coste-efectivas, de bajo consumo de potencia y fiables. Mientras el estándar IEEE 802.15.4 fue ratificado en 2003, la ratificación de la especificación de la versión 1.0 de *ZigBee* no llegó hasta diciembre de 2004.

La figura siguiente representa los distintos niveles de la pila de protocolos definidas por la *ZigBee Alliance*, incluyendo la capa de red, seguridad e interfaz de aplicación y su relación con IEEE 802.15.4 y las aplicaciones cliente.

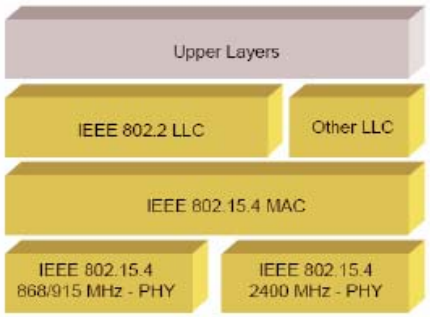


Fig 2. Niveles de la pila de protocolos según la ZigBee Alliance

Los protocolos de nivel de red están definidos para soportar topologías en estrella y malla. Un dispositivo *ZigBee* puede funcionar de tres formas: *coordinador de red*, *coordinador* o *dispositivo de red*. Un dispositivo de red puede iniciar y terminar comunicaciones, mientras que un coordinador puede también encaminar mensajes. Los dispositivos son preprogramados para su función en la red:

- Los coordinadores escanean para encontrar canales no utilizados en la red en estrella.
- El coordinador de red o encaminador (dispositivo del núcleo la red en malla) escanea para encontrar canales activos a los que unirse, y permite a otros dispositivos que se unan a él.
- El dispositivo terminal siempre intenta unirse a redes existentes.
- A este respecto, la *ZigBee Alliance* define dos tipos de dispositivos:
- *Full Function Device* (*FFD*): funciona en cualquier topología, es capaz de entenderse con otros *FFDs* y *RFDs* y puede operar en los tres modos (coordinador de red, coordinador y dispositivo de red).
- *Reduced Function Device* (*RFD*): limitado a la topología en estrella, sólo puede operar con un *FFD* coordinador, no puede convertirse en coordinador, pero es extremadamente simple y puede ser implementado usando pocos recursos y memoria.

Adicionalmente, se proponen dos modos de direccionamiento:

- Direcciones cortas de 16 bits:
    - Redes de hasta 65.000 (216) nodos.
    - Redes simples de más de 65.000 (216) nodos utilizando direccionamiento local.
- Direccionamiento IEEE de 64 bits: redes que pueden alcanzar hasta 264 nodos.

En cuanto a seguridad, la *ZigBee Alliance* define varios modos de seguridad a nivel de *MAC*:

- *Modo inseguro*: obligatorio para todos los dispositivos, pero que no proporciona ninguna seguridad. Si la seguridad no es un factor importante para la aplicación o las capas superiores proporcionan suficiente protección, un dispositivo puede seleccionar el modo inseguro para la transferencia de datos.
- *Modo ACL* (*Access Control List*)
    - Es opcional
    - No aplica métodos criptográficos
    - Se compone de dispositivos los cuales comparten un clave.

- Un dispositivo puede usar el modo *ACL* para prevenir dispositivos no autorizados para acceder a sus datos, pero no se aplican métodos criptográficos en las comunicaciones.

- *Modo seguro*
  - Es opcional
  - *Advanced Encripation Standard* (*AES*)
  - 4 servicios de seguridad: *ACL*, encriptación de datos, integridad de trama y refresco secuencial.

3 OBJETIVOS

Este PFM se encuadra dentro de un proyecto cuya meta principal era la realización de un primer prototipo de un sistema de localización en interiores mediante el uso de redes inalámbricas.

La razón subyacente para la construcción del mencionado prototipo es que sirva para cumplir con dos propósitos ulteriores: por un lado actuar como herramienta para la toma de medidas y obtención de resultados prácticos que nutrieran de experiencia las investigaciones teóricas en cuanto a métodos de cálculo para el posicionamiento en interiores; y por otro lado, hacer las veces de demostrador de las capacidades del sistema con la intención de producir la transferencia de esta tecnología al conjunto de la sociedad a través de su industrialización y explotación comercial.

Estos dos objetivos, de investigación y comercial, comparten puntos de vista comunes pero, al mismo tiempo, presentan también otros que los alejan en cuanto a requisitos. Así pues, se considera que el prototipo debe tener muy en cuenta las siguientes características globales:

Flexibilidad y Escalabilidad.

> Desde el punto de vista de la investigación, el sistema debe poder adaptarse fácilmente a los distintos experimentos: la modificación de buena parte de las variables de funcionamiento, la introducción de distintos motores de cálculo de la posición… Así mismo, desde el punto de vista comercial, al no haber aún ningún requisito concreto, debe estar lo suficientemente preparado para amoldarse a un amplio rango de situaciones y escenarios: número de nodos, distintos programas cliente del sistema, o distinto hardware con diferentes capacidades. Como se verá más adelante una consecuencia casi inmediata de todo lo mencionado es la necesidad de extraer el cálculo de la posición fuera de los propios sensores inalámbricos, así como diseñar y desarrollar el sistema con un enfoque distribuido que permita añadir recursos sin introducir ninguna modificación sustancial.

Multisistema y Multitecnlogía.

> Aunque el objetivo principal se basa en la tecnología *ZigBee*, se plantea como requerimiento del sistema su compatibilidad con otras tecnologías. De esta forma, se permite comparar prestaciones y estudiar las posibilidades de colaboración y combinación de la información procedente de cada una de ellas. Se contempla como punto de partida la compatibilidad de los componentes del sistema con *ZigBee* y *Wi-Fi*.

Transparencia al usuario y facilidad de uso.

> El programa cliente interaccionará con el sistema de manera que al usuario final le sea totalmente transparente cómo se calcula la posición de un dispositivo y la manera en que se le entrega esa información e, incluso, a qué tipo de red pertenece dicho dispositivo.

Acceso remoto.

> Se desea, además, que el usuario pueda acceder a la información de posicionamiento sin necesidad de estar conectado a la red inalámbrica empleada para tal, sino simplemente contando con una conexión IP que le ponga en contacto con el sistema.

Además de las necesidades apuntadas para determinar la localización de un dispositivo inalámbrico se requieren mecanismos de gestión de la red inalámbrica y del propio sistema. Mecanismos que por las mismas necesidades descritas arriba, debían quedar abiertos a la introducción de nuevas características de gestión y adaptarse a las capacidades de los dispositivos inalámbricos (relativamente pequeñas en el caso de *ZigBee*). Se incluyen aquí la interfaz de gestión de cada uno de los distintos bloques del sistema, los mensajes de control y comunicación entre el servidor y los dispositivos inalámbricos, pero también un simple mecanismo de monitorización (*keep-alive*) de estos últimos por el primero.

El objetivo de este PFM es construir la estructura puente entre los distintos módulos de un sistema de localización y una red de dispositivos *ZigBee* desplegada, posibilitando así

el intercambio de la información necesaria tanto para el cálculo de la localización como para la gestión remota de la red inalámbrica. Como veremos más adelante, este objetivo engloba el desarrollo tanto del módulo que permite al sistema la comunicación con redes *ZigBee*, como del programa de funcionamiento de los propios dispositivos.

Para concluir, la finalidad última de este documento es ofrecer un pequeño manual de funcionamiento del sistema a partir del cuál se pueda continuar el trabajo de desarrollo, por un lado, realizando los experimentos necesarios para proseguir con la investigación de algoritmos de localización en interiores, y, por otro lado, extrayendo de esta experiencia las ideas necesarias para mejorar el sistema realizando las modificaciones necesarias para la mejora de las prestaciones del mismo.

4 MATERIALES Y MÉTODOS

El sistema tiene dos partes diferenciadas: por un lado el sistema de localización como un sistema flexible e independiente de la tecnología, lo que se consigue gracias a una aproximación modular; y por otro lado, el programa embebido en cada una de las motas, dividido en diferentes componentes que interaccionan por medio de interfaces de manera que se pueda sustituir un componente por otro siempre y cuando se mantenga intacta dicha interfaz.

Existe un tercer aspecto relacionado con la aplicación que usará el usuario para interactuar con el sistema para gestionar la red y para visualizar los contenidos, por esta razón es conveniente hacer notar que el sistema es accesible desde cualquier tipo de programa cliente que se quiera conectar de manera remota.

En este apartado se expondrán los materiales utilizados para el presente Proyecto Fin de Máster, a saber, el sistema de localización, los propios dispositivos *ZigBee* y aquellos aspectos relacionados con la programación de ambos.

Asimismo, se detallará el plan de trabajo seguido para la consecución de los objetivos marcados que incluye un desglose de actividades y una planificación temporal de las mismas.

4.1 LOCALIZACIÓN

Cualquier dispositivo de comunicaciones inalámbricas es susceptible de ser localizado en función de diversos parámetros observables en los mensajes que intercambia con otros nodos de su red.

TOA (Time Of Arrival) y TDOA (Time Difference Of Arrival)

La localización se basa en el tiempo de llegada del mensaje debido propagación de la onda electromagnética, observando el valor absoluto (*TOA*) o la diferencia entre medidas de varios nodos (*TDOA*).

Se emplea en localización en exteriores (por ejemplo, *GPS*) pero no está muy extendido en aplicaciones en interior, donde los requisitos suelen ser más exigentes, debido a que la precisión obtenida mediante este método depende de la calidad del reloj interno así como de la sincronización de los nodos, lo que encarece considerablemente el sistema.

AOA (Angle Of Arrival)

El parámetro observado es el ángulo que ofrece una máxima potencia de recepción.

En este caso la precisión depende directamente de la directividad de la antena empleada, lo que choca con el uso justificado de antenas omnidireccionales en la mayor parte de los dispositivos inalámbricos.

RSSI (Received Signal Strength Indication)

La localización se determina en función de la potencia recibida, normalmente mediante distintas técnicas de triangulación y algoritmos de cotejo con datos provenientes de medidas previas.

En general, la precisión depende del comportamiento del canal y de su mayor o menor semejanza con el espacio libre, en el que se cumple el supuesto de que la potencia de la onda se atenúa con el cuadrado de la distancia. Por esta razón, los entornos interiores requieren algoritmos más complejos que tengan en cuenta los efectos del *multitrayecto*.

4.1.1 SISTEMA DE LOCALIZACIÓN

El sistema está dividido en módulos que tratan cubren las distintas funcionalidades del sistema de localización:

a) Gestionar la información de cada uno de los dispositivos inalámbricos.

b) Mantener una conexión permanente con los dispositivos inalámbricos para su gestión y para la obtención de información necesaria para el posicionamiento.

c) Calcular la posición de un dispositivo a partir de las medidas recogidas.

d) Ofrecer la información a los distintos clientes que así lo soliciten.

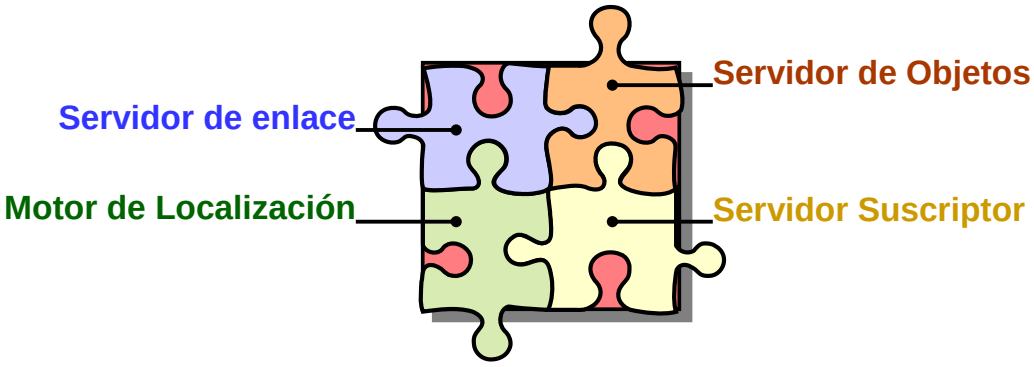


Fig 3. Figura de los módulos del sistema

Cuando el usuario acceda al sistema lo hará con intención de extraer cierta información de los distintos dispositivos de la red, bien sobre su posición o bien para tareas de gestión. En cualquiera de los casos, lo que en realidad va a obtener el usuario es una copia virtual del dispositivo inalámbrico al que desea acceder. Ese objeto virtual no es más que una abstracción del dispositivo en una serie de interfaces que proveen ciertas funcionalidades.

Esta función de proporcionar al usuario estos objetos virtuales se ha llamado ***Servidor de Objetos***, y será la principal interfaz de uso del sistema que tendrá el usuario final. Serán las distintas interfaces del *Servidor de Objetos* las que el usuario utilizará para enviar cualquier tipo de información al sistema y, a través de éste, a los distintos dispositivos de la red inalámbrica.

Para este último caso en el que la información viaje hacia los dispositivos, el sistema necesitará convertir dicha información en mensajes que puedan difundirse por la red inalámbrica y ser entendida por los dispositivos. Esa función, de puente entre las distintas redes, se implementa en el llamado ***Servidor de Enlace***.

El *Servidor de Enlace* es la función encargada también de permanecer a la escucha de toda aquella información proveniente de la red inalámbrica. En los procesos de localización, por ejemplo, el informe de medidas recopiladas por el dispositivo llegará en forma de paquetes al *Servidor de Enlace*. Éste será el encargado de preparar toda esa información contenida en dichos paquetes para el cálculo de la posición. Todo lo relacionado con dicho cálculo es tarea del ***Motor de Localización***, unidad que posee lo necesario para convertir el conjunto de medidas de potencia en coordenadas {x, y, z}.

Una vez conocidas las coordenadas de la posición del dispositivo es momento de hacérselas llegar a aquellos usuarios interesados. En este caso el modelo adoptado no es tanto el de cliente-servidor sino más bien el de suscripciones. El ***Servidor Suscriptor*** provee todos los mecanismos

necesarios para hacer posible esta relación por suscripción, desde el registro de los usuarios interesados en ciertos eventos hasta la entrega de la notificación una vez dicho evento se produce. Fiel a su compromiso con la flexibilidad, el sistema no pone límite a la clase de eventos a los que un usuario puede estar subscrito, proveyendo una interfaz abierta a cualquier nuevo evento que se desee definir. Pero dado que el objetivo principal del sistema es la localización, dicha interfaz incluye funcionalidades específicamente pensadas para eventos asociados al cambio de posición de un dispositivo.

Aunque el *Servidor de Objetos* sea el nodo fundamental en la relación del usuario con el sistema, ciertas interfaces del resto de nodos están accesibles también para los usuarios. Por ejemplo, el *Servidor Suscriptor* provee una interfaz a la que el usuario accede directamente para subscribirse a los distintos eventos.

La figura del *Servidor Suscriptor* y la relación por subscripción permite una gestión más eficiente y rápida de las notificaciones, cuestión que puede resultar clave cuando un sistema de posicionamiento instantáneo tiene que hacer frente a un número creciente de usuarios conectados y dispositivos monitorizados.

A medida que las redes inalámbricas crecen para cubrir un espacio mayor y a medida que el número de usuarios aumenta, también crece la posibilidad de que las capacidades de un sistema alojado en una única máquina se vean sobrepasadas por las necesidades. Es precisamente para hacer frente a este posible incremento de las necesidades computacionales para lo que el sistema fue pensado, diseñado e implementado para trabajar de forma totalmente distribuida: las cuatro funcionalidades básicas vistas anteriormente no sólo pueden trabajar en máquinas distintas, sino que también pueden replicarse las veces que se considere necesario.

El sistema contempla una función extra que facilite la gestión y el funcionamiento de este sistema distribuido, el ***Gestor de Red***. Su comportamiento es similar al de un *Domain Name Server* (*DNS*) en la Internet: se le indica el nombre de un nodo y él provee la dirección de red del mismo. Con el *Gestor de Red* desplegar una nueva copia de uno de los nodos consiste simplemente en dar un nuevo alta, y la búsqueda del nodo más adecuado para atender ciertas peticiones es una simple pregunta perfectamente definida en una de sus interfaces. Cualquiera de los nodos del sistema, incluidos los propios usuarios, sólo necesitará conocer la dirección de red del *Gestor de Red* y solicitarle a él el resto de direcciones. Así por ejemplo, cualquier usuario que quiera acceder a una copia virtual del dispositivo que desea gestionar sólo tendrá que solicitarle al *Gestor de Red* la dirección de red de una de las réplicas del *Servidor de Objetos*. De forma similar hará el resto de nodos en sus interconexiones: el *Servidor de Enlace* necesita encontrar el *Motor de Localización* al que enviar los informes de medidas de potencia que reciba.

Además, todos los componentes que conforman las distintas funciones del sistema poseen una interfaz de gestión propia a la cual un usuario administrador puede acceder para configurarlos y establecer cambios en su comportamiento. Entre estas tareas de carácter administrativo para las cuales los nodos tienen definidas interfaces especiales, se encuentra también la tarea de **auto-entrenamiento**. Para poder establecer la posición de un dispositivo inalámbrico móvil a partir de la potencia que recibe de sus vecinos, puede ser necesario un proceso previo de toma de medidas de potencia que sirve de calibración o entrenamiento al motor de localización. La función de auto-entrenamiento permite aprovechar la propia densidad de los dispositivos desplegados para realizar esta toma de medidas de una manera automática. Para ello, el usuario administrador sólo tiene que hacer uso de la interfaz correspondiente del *Motor de Localización* que se encargará de solicitar a cada uno de los dispositivos desplegados un informe con el número deseado de medidas de potencia, características de su posición. El auto-entrenamiento es una utilidad aún por explotar que entraña ciertas limitaciones lógicas inherentes a la propia filosofía del algoritmo, que no es otro que sustituir la medida que haría un dispositivo móvil por la que proporcionan las propios nodos *baliza* desplegados, entre ellas:

- La medida tomada a la altura de despliegue de las balizas (probablemente, el techo) trata de sustituir a la medida que un dispositivo móvil recogería a una altura distinta (habitualmente, a media altura de una persona).

- El informe de cada *baliza* no incluye su medida propia, esta es la medida que un dispositivo móvil en ese punto tomaría de esa misma *baliza*, y que, sin embargo, cabría esperar como la medida más significativa, es decir, la medida que con mayor probabilidad será la de mayor potencia, por encontrarse la *baliza* justo encima.

El siguiente cuadro muestra una sencilla representación gráfica de las distintas interfaces que ofrecen cada uno de los bloques del sistema de localización introducidos anteriormente.

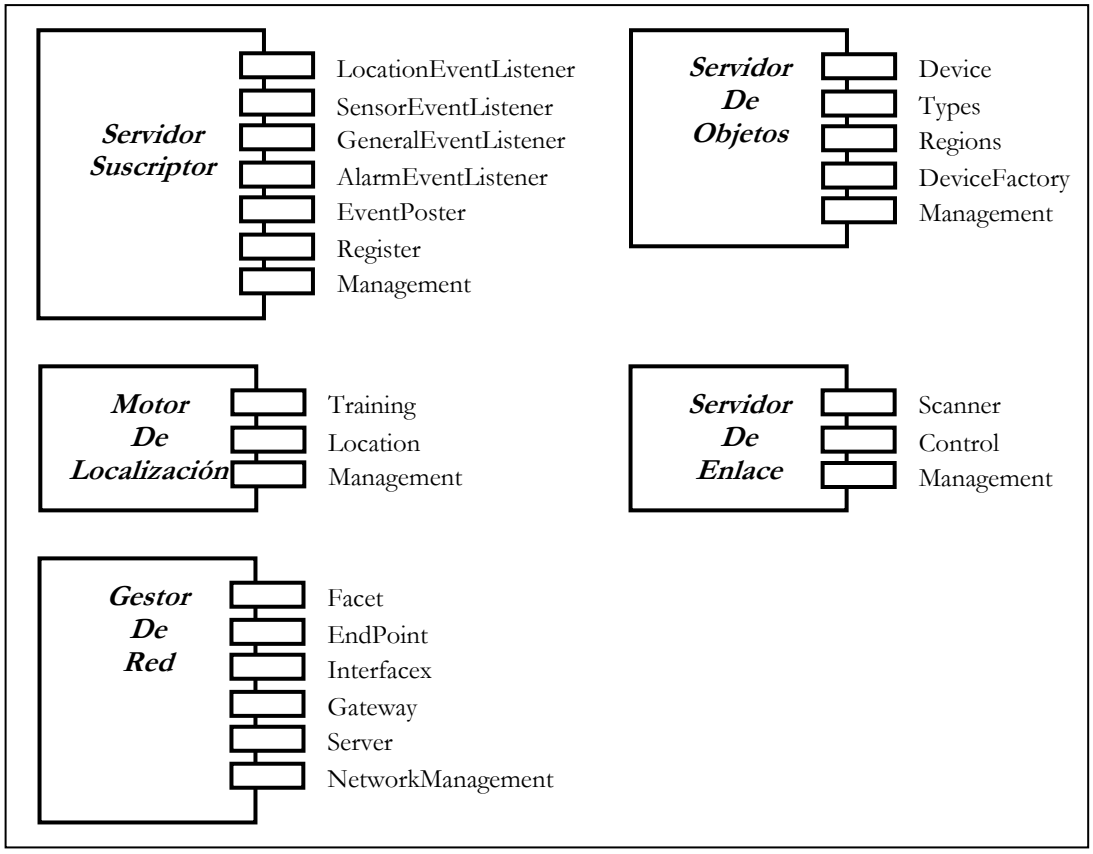


Fig 4. Interfaces publicas de los distintos módulos del sistema

4.1.2 DISPOSITIVOS *INALÁMBRICOS*

Como se ha visto anteriormente, la localización puede obtenerse de múltiples maneras. Una de ellas es aplicando distintos algoritmos de cálculo al valor *Received Signal Strength Indication* (*RSSI*) de los mensajes inalámbricos. En despliegues controlados de redes, además, se puede contar con la información de la posición de los dispositivos que conforman esa red. En estos casos se trata de establecer cierto intercambio de mensajes entre estos dispositivos (*balizas*) y aquellos que se pretenden conocer su posición (*móviles*) para aplicar el cálculo a un conjunto de pares de valores $\{RSSI_i, \text{POSICIÓN}_i\}$.

Existen muchas formas de afrontar este diálogo de mensajes orientados a recoger el suficiente número de medidas *RSSI*. Una agrupación podría ser la siguiente:

a) *Mensajes bajo demanda o "Móvil empieza"*. Cada *móvil* solicita a los dispositivos *baliza* de su entorno el envío de un mensaje del que obtener el *RSSI*.

b) *Mensajes periódicos o "Baliza empieza"*. Las balizas envían un mensaje simple periódicamente y los dispositivos móviles que lo escuchen podrán medir el valor *RSSI* de cada una de ellas.

Y estas son las estrategias para la obtención de medidas *RSSI*, pero falta la otra parte de la información necesaria para la localización: la posición fija y conocida de las *balizas* a quién pertenecen esas medidas. Así que, independientemente de la estrategia (a) o (b) adoptada para establecer el diálogo, existen, al menos, otras dos formas de ver el problema de la información de la posición de las balizas:

1- *Las balizas envían un mensaje indicando en él su posición*. Los dispositivos *móviles* sólo tiene que juntar suficiente de ellas y realizar el cálculo pues ya tendrían toda la información necesaria: medidas de *RSSI* y la posición fija y conocida de las balizas a quién pertenecen esas medidas.

2- *Las balizas envían un mensaje simple*. Los dispositivos *móviles* podrían medir el valor *RSSI* de cada una de ellas pero les faltaría la información de las posiciones fijas de las *balizas* que debería, o bien tener almacenada de antemano, o bien obtenerla de alguna fuente externa.

Sin entrar en discusiones fuera del alcance de este proyecto fin de master sobre cuál es la opción más adecuada y, antes de desvelar la estrategia adoptada, se esbozan algunas de las razones que llevaron a su elección:

- Flexibilidad.
- Escalabilidad.
- Complejidad variable en el cálculo de la localización.
- Ahorro energético en los dispositivos *móviles*.

Un aspecto importante para entender mejor ésta y otras decisiones, es que el proyecto pretendía desde un principio, y pretende ahora, servir como soporte a estudios de investigación en localización en interiores, es decir, que debía ser lo suficientemente flexible como para poder amoldarse a distintos despliegues, a distintos tamaños de red, a distintas densidades de nodos y también a distintos algoritmos de cálculo, algunos de ellos de una elevada complejidad de cómputo. Razones suficientes como para descartar el cálculo de la posición en la propia mota *móvil*, así como también el uso de *balizas* preprogramadas con el dato de su posición fija.

Por otro lado, este proyecto se diseñó con vocación de lograr un prototipo final suficientemente comercial como para servir de demostración de lo que podía llegar hacerse en localización en interiores con tecnologías inalámbricas, lo que significaba tener en cuenta escenarios en el que el número de terminales *móviles* pudiera llegar a ser realmente alto. Además, desde un principio se consideró la necesidad de contar con una red de balizas que no sólo fueran desplegadas de forma planificada sino que además tuvieran suministro eléctrico garantizado (conexión a la red eléctrica en lugar del empleo habitual de baterías) lo que resta importancia al ahorro energético en los nodos fijos de la red frente al ahorro en nodos móviles.

Por todas estas razones, la flexibilidad, la escalabilidad en número de nodos móviles y la importancia del ahorro energético en esta clase de dispositivos, llevó a la elección de un modelo en el que las *balizas* transmiten *periódicamente* mensajes *simples* (*beacons*) de forma independiente y es cada dispositivo *móvil* el que recopila el conjunto suficiente de medidas de potencia, las preprocesa y hace llegar el informe hasta la *estación base* (a través de la red que establecen las propias *balizas*). El *servidor*, una máquina mucho más potente que cada una de las motas, recibe esta colección de medidas, extrae de una base de datos la información actualizada de la posición de cada nodo fijo correspondiente, y emplea el algoritmo escogido para calcular la posición de la mota *móvil* que envió el informe.

4.1.2.1 Modo 'Baliza'

En el modo '*baliza*' la mota será la encargada de proveer soporte a la red inalámbrica, encaminando los paquetes de datos hacia y desde la *estación base*, al tiempo que envía mensajes *beacon* periódicamente a partir de cuyas medidas se calculará la posición de los nodos móviles.

Mientras que los mensajes de datos emplearán métodos de encaminamiento o diseminación, los mensajes *beacon* serán mensajes en radiodifusión por simple CSMA/CA. La escalabilidad de la red como requisito inicial, llevó a contemplar la posibilidad de independizar los canales por cada tipo de mensaje, datos y *beacons*, dejando abierta una posible planificación de frecuencias en caso de encontrar problemas de saturación del canal durante los despliegues. Teniendo en cuenta ahora este aspecto, el funcionamiento de una mota en modo '*baliza*' se resume en los siguientes puntos[1]:

1. Esperar un determinado periodo de tiempo en el canal de comunicaciones o datos durante el que se reciben y difunden paquetes de control provenientes de la *estación base* así como se encaminan hacia ésta los informes de medidas y otros paquetes de datos provenientes de las motas *móviles* y otros dispositivos de la red.
2. Cambiar de canal para enviar el mensaje de *beacon*. Y volver al canal de comunicaciones y al punto 1.

4.1.2.2 Modo 'Móvil' y 'temp-Móvil'

Cualquier mota que quiera ser localizada sólo tiene que recopilar, preparar y enviar un informe de medidas. Ese modo de proceder es el que se ha denominado '*movil*', si es el comportamiento general de la mota y los informes se envían periódicamente, y '*temp-móvil*' si es temporal y responde a una petición bajo demanda proveniente del servidor. Para hacerse una mejor idea de la diferencia, el modo '*móvil*' es el modo normal y con el que funcionan desde el principio las motas *móviles* mientras que el modo '*temp-móvil*' está pensado más para posibilitar la recopilación de medidas desde los dispositivos fijos (como parte del proceso de auto-entrenamiento) y, en general, un mayor control por parte del servidor.

Al introducir la posibilidad de usar distintos canales para datos y *beacons*, el modo *móvil* también se divide en dos partes[2]:

1. Cambiar de canal y permanecer en él un tiempo durante el cual se escuchan los mensajes *beacons* de las estaciones *baliza* y se recopilan las medidas de localización en una pila que posteriormente se preprocesará y conformará el informe de medidas de localización.
2. Volver al canal de comunicaciones y esperar un determinado periodo de tiempo durante el que se preprocesan las medidas y se prepara y envía el informe de medidas de localización así como se reciben y atienden paquetes de control provenientes de la *estación base*.

4.1.2.3 Inicialización

Existen múltiples razones por las que un dispositivo podría reiniciarse:

- reinicio manual;
- reinicio programado o bajo demanda;
- por iniciativa del sistema operativo;

---

[1] Ver *Diagrama del Modo Baliza* en **Anexo 1: Diagramas** (Pág. 65)

[2] Ver *Diagrama del Modo Móvil* en **Anexo 1: Diagramas** (Pág. 65)

- tras recobrar el suministro o la batería;

Fuera cual fuere, una vez el dispositivo completa su reinicio hardware, se aprovecha que el evento '*boot*' del componente '*Main*' se dispara para poner en marcha el procedimiento de inicialización de la mota que le permitirá, en definitiva, conectarse a la red y dar a conocer su presencia al servidor. El esquema de los pasos a seguir en el siguiente[3]:

1- Se inicializan los parámetros generales[4].

2- Se habilitan las distintas interfaces radio y serie, y tanto *Receive* como *Send*.

3- Se envía un mensaje hacia la estación base (*Send Up*) indicando que la mota se quiere conectar a la red.

4- Se espera por un tiempo determinado a la respuesta del servidor. Si no llega se vuelve al paso 3 y se envía un nuevo mensaje.

5- Recibido el correspondiente mensaje de confirmación por parte del Servidor, la mota ya puede comenzar su comportamiento dentro de la red de localización: modo '*baliza*', modo '*móvil*', o modo '*estación base*' (estación base)

Este simple procedimiento de inicialización con intercambio de mensajes entre la mota y el servidor tiene varios objetivos:

- Comprobar los canales de comunicación entre la mota y el servidor.
- Disponer de una lista dinámicas de los dispositivos conectados en el árbol lógico que depende de una determinada estación base y que usará el servidor para el mecanismo de *keep-alive*.
- Dejar la puerta abierta a una posible descarga de la configuración inicial con la que la mota comenzará a funcionar: '*baliza*', '*móvil*' y '*estación base*' (estación base). Esto es posible gracias a que el código es el mismo en todas las motas, independientemente de que en un principio se planificara y programara para uno u otro modo de funcionamiento. Para esta primera versión del proyecto se optó por determinar el modo del dispositivo en función del identificador de nodo que se le daba en su programación[5].

4.2 *ZIGBEE*

4.2.1 DISPOSITIVOS *ZIGBEE*

En la realización de estos primeros prototipos del sistema de localización se han empleado los sensores inalámbricos de *Crosswbow* [XBow] tipo *MicaZ* con los que ya se contaba en el departamento de Teoría de la Señal y las Comunicaciones (TSC)

Los dispositivos *ZigBee* o 'motas' *MicaZ* [MicaZ] son módulos inalámbricos con un microcontrolador *AT-mega 128L*, que están alimentados por dos baterías clase AA y tienen un tamaño de 58x32x7 milímetros (sin incluir el habitáculo para las baterías). Las diferencias con otros

---

[3] Ver *Diagrama del Proceso de Inicialización* en **Anexo 1: Diagramas** (Pág. 65)

[4] Ver '**Inicialización de Parámetros**' en la sección del **Componente *MoteParameters***

[5] Los rangos de identificadores de nodo para cada modo de funcionamiento ('*bridge*', '*beacon*' y '*mobile*') pueden encontrarse en el archivo *./include/**LBSconfig.h***

módulos similares (como los *Mica2*) vienen dadas por el transceptor radio y su banda de trabajo en 2,4 Ghz que posibilita una tasa de transmisión algo mayor (250 kbps). En la hoja de características se indica que la vida de las baterías es de varios años, aunque no se especifica en qué condiciones.

Un aspecto importante de los módulos *MicaZ* es su conector de expansión de 51 pines. Éste conector es útil por dos motivos:

- Permite al dispositivo actuar como estación base empleándolo junto con una plataforma con interfaz serie, USB, Ethernet... Estas plataformas sirven, por tanto, de puertas de enlace entre uno de estos módulos y un ordenador personal, y, por extensión, entre una red *ZigBee* y cualquier otro tipo de red, IP, por ejemplo. En el departamento de TSC se dispone de algunos modelos de las plataformas serie [*MIB510*], USB [*MIB520*] y Ethernet [*MIB600*].
- Permite conectar distintas placas de sensores. Aunque para el sistema de localización no es necesario, es lógico pensar en un futuro sistema que aproveche tanto la información de monitorización como de posicionamiento. En el departamento de TSC se dispone de placas tipo MTS300 y MTS310 [MTS].

4.2.2 TINYOS

De la propia Web oficial [TinyOS] resaltaremos algunos conceptos que pueden ayudarnos a tener una mejor idea de lo que es y para lo que puede servir *TinyOS*:

- *TinyOS es un sistema operativo de código-libre diseñado para redes de sensores embebidos e inalámbricos*. *TinyOS* está pensado para crear códigos que optimicen en lo posible los escasos recursos de un microprocesador que controla una serie de sensores y un módem inalámbrico. Por tanto, no se empleará *TinyOS* para interaccionar directamente con un módem *ZigBee* en casos en los que se quiera, por ejemplo, usar el módem en un perfil de puerto serie.
- *Ofrece una arquitectura que se basa en componentes*. Las aplicaciones se escriben en NesC y se construyen interconectando distintos componentes a través de las interfaces que éstos proveen.
- *La librería de componentes de TinyOS incluye protocolos de red, servicios distribuidos, drivers para sensores, y herramientas de adquisición de datos*. El sistema operativo ofrece varios niveles de abstracción: desde los componentes más cercanos al lenguaje hardware, hasta componentes de más alto nivel que emplean a los primeros para lograr una mayor independencia de la plataforma[6] empleada. Y al ser libre, todos los códigos están disponibles de manera que, en principio, es posible tener acceso a cualquier variable o incluso crear nuevos componentes para nuevas plataformas.
- "*El modelo de ejecución TinyOS, orientado a eventos*…". Los programas *TinyOS* se mueven por estímulos y esperan a que los distintos eventos se produzcan para actuar de una manera u otra. Este modelo de ejecución requiere también de una manera de pensar y diseñar las aplicaciones distinta en la que, quizás, los diagramas de estado-evento se adaptan mejor que los tradicionales diagramas de flujo.

4.2.2.1 Esquema General de una Aplicación TinyOS

El siguiente esquema general intenta dar una imagen gráfica de la interacción de los distintos componentes de los que se forma el software de las motas y que puede servir de ejemplo de cómo

[6] En este caso, se emplea la palabra *plataforma* por analogía al término con el que, en *TinyOS*, se refiere a un sistema embebido concreto que combina un determinado microprocesador con un determinado módem.

los componentes *TinyOS* se interconectan para cubrir las distintas necesidades que puede requerir una aplicación como ésta:

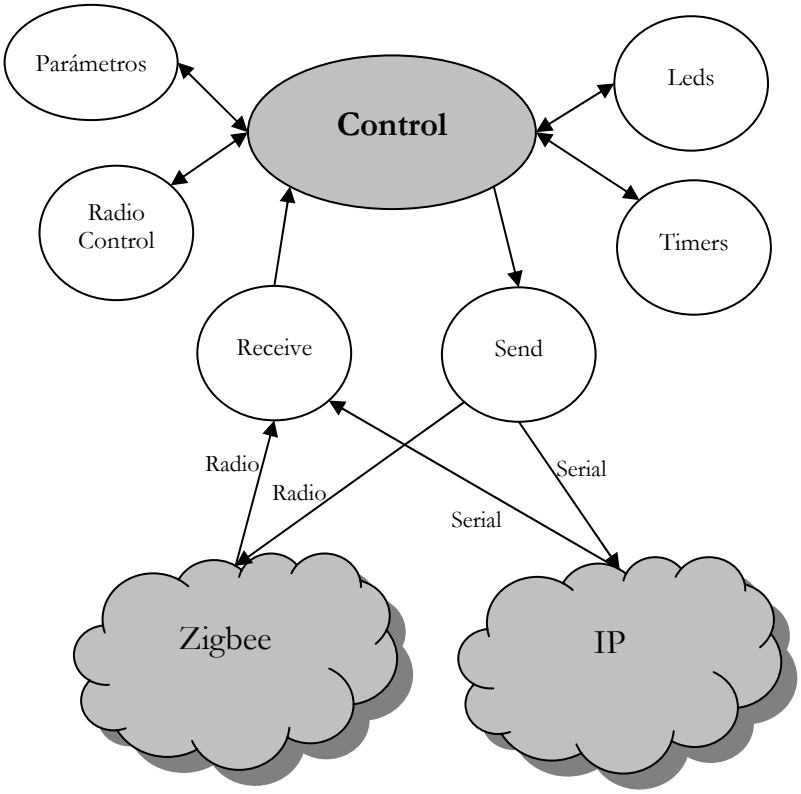


Fig 5. Esquema general del programa de la Mota y su interconexión con las distintas redes

4.2.2.2 Envío y Recepción de Mensajes en TinyOS

Existe una gran variedad de componentes que implementan el envío y recepción de mensajes a través de las interfaces radio y serie de las motas.

Las interfaces más generales y usadas por la mayor parte de estos componentes son las interfaces *Receive* y *Send*. Estas dos constituyen los principales métodos necesarios para el intercambio de mensajes entre dispositivos, pero suelen ir acompañadas de otra serie de interfaces que sirven como herramientas que facilitan el tratamiento de los mensajes a ser enviados o ya recibidos: `Packet`, `AMPacket`, `PacketAcknowledgements`, `CC2420Packet`, `CollectionPacket`, `CtpCongestion`...

Además existe una interfaz de *Control* necesaria para inicializar y habilitar las distintas interfaces hardware de envío y recepción de mensajes. La interfaz de Control tiene un funcionamiento muy simple a un nivel alto de programación, no necesita explicación y se resume en, principalmente, un comando '`start`' y, opcionalmente, un evento '`startDone`'.

Interfaces Receive

Estos componentes proporcionan los métodos necesarios para la recepción de mensajes. La interfaz de un *Receive* se forma de los siguientes métodos y eventos:

`event message_t* receive(message_t* msg, void* payload, uint8_t len);` Es el evento fundamental de la interfaz pues es el que se produce cuando un nuevo mensaje llega a la interfaz. Parámetros: • msg: puntero al nuevo mensaje recibido, que siempre tiene la estructura propia de los mensajes en *TinyOS* definida por *message_t* • payload: puntero a la carga de datos que trae el nuevo mensaje, evitando la necesidad de conocer cuáles son las cabeceras propias de la clase de mensaje para la que se suscribió la interfaz. • len: longitud del campo 'payload' en bytes.
`command void* getPayload(message_t* msg, uint8_t* len);` Muy útil para acceder a la posición del campo de datos del mensaje sin necesidad de conocer cuáles son las cabeceras propias de la clase de mensaje para la que se suscribió la interfaz.
`command uint8_t payloadLength(message_t* msg);` Devuelve la longitud en bytes del campo 'payload' de un mensaje.

Tabla 6. Descripción de la interfaz Receive

Un componente que use una interfaz *Receive* deberá implementar el evento ***receive***, y a partir de entonces queda suscrito a la recepción de una determinada clase de mensajes. Esto significa que el componente que implemente la interfaz *Receive* deberá contemplar un parámetro en su constructor que le indique a qué clase de mensajes deberá atender. Esto último se puede hacer de distintas formas tal y como muestran los siguientes ejemplos de interfaces:

```
generic configuration AMReceiverC(am_id_t amId) {
  provides {
    ...
    interface Receive;
    ...
  }
```

```
generic configuration DisseminationC(typedef t, uint16_t key) {
  provides {
    ...
    interface Receive;
    ...
  }
```

```
configuration SerialActiveMessageC {
  provides {
    ...
    interface Receive[am_id_t id];
    ...
  }
```

```
configuration CollectionC {
  provides {
    ...
    interface Receive[collection_id_t id];
    interface Receive as Snoop[collection_id_t];
    ...
  }
```

Tabla 7. Distintos componentes que implementan la interfaz Receive

Como se puede adivinar por su nombre, existen distintos componentes que implementan la interfaz *receive* pero con objetivos muy particulares:

- Para el medio radio: `AMReceiverC`, `ActiveMessageC`.
- Para el puerto serie/usb: `SerialActiveMessageC`.
- Para mensajes de difusión: `DisseminationC`, `MoteDisseminationC`[7].
- Para mensajes hacia un nodo raíz o sumidero: `CollectionC`.

Para este proyecto, se hace uso de los cuatro tipos de componentes:

- *Receive 'Beacon'*: para cualquier paquete *broadcast* que emitan los nodos *baliza*.
- *Receive 'Serial'*: para mensajes desde el servidor hacia la red *ZigBee*.
- *Receive 'Down'*: el paquete recibido tiene como origen la raíz del árbol o un nodo intermedio de posición superior en la estructura lógica.
- *Receive 'Up'*: el paquete recibido tiene como origen algún nodo hoja o algún nodo intermedio de posición inferior en la estructura lógica. En un principio, sólo los nodos sumideros (*'estación base'*) contarán con el evento de mensaje recibido en sentido Up.

Interfaces Send

Estos componentes proporcionan los métodos necesarios para el envío de mensajes desde la mota. La interfaz de un *Send* se forma de los siguientes métodos y eventos:

`command error_t` **`send(message_t* msg, uint8_t len);`** Es el método fundamental de la interfaz pues es el que posibilita el envío del mensaje a través de la interfaz. Parámetros: • msg: puntero al nuevo mensaje a ser enviado, que siempre tiene la estructura propia de los mensajes en *TinyOS* definida por *message_t* • len: longitud del campo 'payload' en bytes.
`command error_t` **`cancel(message_t* msg);`** Método para cancelar el envío del un mensaje.
`event void` **`sendDone(message_t* msg, error_t error);`** Este evento se produce cuando el mensaje ha sido enviado por completo y por tanto la interfaz está libre ya para poder enviar uno nuevo. Parámetros: • msg: puntero al mensaje recientemente enviado, que siempre tiene la estructura propia de los mensajes en *TinyOS* definida por *message_t* • error: señala si el envío concluyó con éxito.
`command uint8_t` **`maxPayloadLength();`** Este método devuelve la longitud máxima en bytes que puede tener el campo de carga de datos en los mensajes de la clase para la que se suscribió la interfaz.
`command void*` **`getPayload(message_t* msg);`**

[7] Componente no genérico de *TinyOS* sino de creación propia de este PFM.

Muy útil para acceder a la posición del campo de datos del mensaje sin necesidad de conocer cuáles son las cabeceras propias de la clase de mensaje para la que se suscribió la interfaz.

Tabla 8. Descripción de la interfaz Send

Un componente que use una interfaz *Send* para el envío de mensajes deberá implementar el evento ***sendDone*** para cada clase de mensaje para el que se haya suscrito una interfaz *Send*. Éstos son unos ejemplos:

```
generic configuration AMSenderC(am_id_t AMId) {
  provides {
    ...
    interface AMSend;
    ...
  }
```

```
generic configuration DisseminationC(typedef t, uint16_t key) {
  provides {
    ...
    interface Send;
    ...
  }
```

```
configuration SerialActiveMessageC {
  provides {
    ...
    interface AMSend[am_id_t id];
    ...
  }
```

```
configuration CollectionC {
  provides {
    ...
    interface Send[uint8_t client];
    ...
  }
```

Tabla 9. Distintos componentes que implementan la interfaz Send

Se distinguen dos clases de *Send*: para el medio radio; y para la interfaz de puerto serie/USB.

Además haciendo uso de la estructura lógica de la red *ZigBee* en forma de árbol, los *Send* para radio se dividen en dos:

- *Send 'Up'*: el paquete es encaminado hacia la raíz del árbol, es decir, hacia el nodo sumidero que sirve de puerta hacia otras redes (*'estación base'*)
- *Send 'Down'*: el paquete tiene como destino un nodo hoja o intermedio y de posición inferior en la estructura lógica.

4.3 PLAN DE TRABAJO

A continuación se muestra el plan de trabajo seguido para la confección de este PFM. En él pueden observarse las distintas etapas descritas a continuación y las características y particularidades propias de cada una de ellas, así como las tareas concretas a completar, los resultados de cada una y la duración final de todas ellas.

1. Definición del proyecto
2. Planificación del proyecto

3. Diseño Conceptual
4. Codificación de procesos
5. Elaboración de la memoria del PFM

4.3.1 DEFINICIÓN DEL PROYECTO

Temporalización:

Semana 1.

Objetivos:

Definir las necesidades del sistema de localización y los objetivos del PFM

Tareas:

- **T1.1 Identificación de necesidades para la elaboración de un PFM**: describir las funcionalidades que requiere el sistema de localización en cuanto a interacción con una red *Zigbee* y que el PFM debe cubrir.
- **T1.2 Estudio del estado del arte en el desarrollo de aplicaciones Java y tecnología ICE**: revisión tanto de tutoriales, librerías y APIs, así como de su integración en los distintos entornos de desarrollo (Eclipse, Netbeans).
- **T1.3 Estudio del estado del arte en el desarrollo de programas *TinyOS***: revisión tanto de tutoriales y librerías de componentes, así como de su integración en los distintos entornos de desarrollo (Eclipse, Netbeans).
- **T1.4 Definición de objetivos iniciales en la elaboración del PFM**.

Entregables (semana en que tendrá lugar):

- **E1.1 Introducción y Necesidades (s1)**: documento que introduce las necesidades del sistema de localización y que debe cubrir el PFM.
- **E1.2 Objetivos (s1)**: documentos con los objetivos iniciales del PFM.

4.3.2 PLANIFICACIÓN DEL PROYECTO

Temporalización:

Semanas 2 a 3.

Objetivos:

Realizar una planificación detallada del PFM. Concretar las acciones y tareas a desarrollar y seleccionar las alternativas más adecuadas en cada caso.

Tareas:

- **T2.1 Descripción del comportamiento general de los dispositivos y los parámetros de diseño a tener en cuenta en su funcionamiento**.
- **T2.2 Definición de distintas alternativas para el diseño de un programa *TinyOS* que instalar en los dispositivos *ZigBee***.
- **T2.3 Descripción del funcionamiento de un módulo que actúe como servidor de enlace entre el sistema y la red *ZigBee***.

- **T2.4 Definición de distintas alternativas para del servidor de enlace**.
- **T2.5 Selección de un diseño**: escoger una alternativa entre todas las presentadas, que será aquella que se adapté mejor a las necesidades definidas en la etapa anterior.
- **T2.6 División del PFM en tareas y fases**: modularización del proyecto y establecimiento de intervalos temporales de ejecución de cada una de las fases

Entregables (semana en que tendrá lugar):

- **E2.1 Diagrama de Gannt del PFM (s2)**: documento que identifique las diferentes fases, las dimensione temporalmente y señale la relación temporal entre todas ellas.
- **E2.2 Análisis de alternativas (s3)**: documentos con las diferentes alternativas a seguir y la finalmente escogida.

4.3.3 DISEÑO CONCEPTUAL

Temporalización:

Semana 4.

Objetivos:

Definición de los distintos procesos y funciones en los que se dividen tanto el servidor de enlace como el programa de las motas. Diseño de los procesos identificados y establecimiento de relaciones y llamadas entre ellos. Integración y revisión del diseño.

Tareas:

- **T3.1 Definición de los procesos que conforman el servidor de enlace**: dividir el diseño del módulo en procesos o funcionalidades más concretas y clasificar cada uno en distintos objetos lógicos.
- **T3.2 Definición de los procesos que conforman el programa de las motas**: dividir el diseño del funcionamiento del dispositivo en procesos o funcionalidades más concretas y clasificar cada uno en distintos componentes *TinyOS*.
- **T3.3 Integración de los procesos**: asegurar la coherencia entre las relaciones y llamadas entre los distintos procesos.
- **T3.4 Revisión de los procesos definidos**.

Entregables (semana en que tendrá lugar):

- **E3.1 Diagramas de Flujo de todos los procesos definidos (s4)**: documento que describe la funcionalidad concreta de cada proceso definido.
- **E3.2 Documentación adicional (s4)**: información que explica, aclara o documenta algún proceso, aspecto o función concreta del diseño

4.3.4 CODIFICACIÓN DE PROCESOS

Temporalización:

Semana 5 a 7

Objetivos:

Realizar la programación de los procesos implementados y su integración dentro del sistema de localización.

Tareas:

- **T4.1 Definición de las interfaces y componentes *TinyOS* necesarios para implementar el comportamiento de la mota:** definición de los distintos procesos que incluye cada parte o rutina del diseño completo y que llegan a formar un programa completo de comportamiento.
- **T4.2 Programación de los componentes *TinyOS*:** implementar el código completo de algunos procesos que completen cada una de las rutinas del programa.
- **T4.3 Definición de las interfaces de los distintos objetos java que compondrán el servidor de enlace:** definición de las distintas funciones que incluye cada interfaz.
- **T4.4 Programación de los objetos java:** implementar el código completo de cada uno de los objetos que componen del módulo.
- **T4.5 Verificación de la coherencia entre los diagramas de flujo y la programación implementada.**
- **T4.5 Integración del servidor de enlace como un módulo más del sistema de localización.**

Entregables (semana en que tendrá lugar):

- E4.1 Ficheros con el código de las rutinas seleccionadas (s8): documento que incorpora la programación a alto nivel de los procesos seleccionados

4.3.5 ELABORACIÓN DE LA MEMORIA DEL PFM

Temporalización:

Semana 1 a 8.

Objetivos:

Realizar un documento entregable como memoria del PFM de acuerdo a los requisitos del máster y describiendo los distintos códigos implementados que permita un mayor entendimiento de los procesos diseñados y facilite su optimización y programación definitiva.

Tareas:

- **T5.1 Definición de la estructura de la memoria del PFM**: identificar los requisitos incluidos en el Reglamento PFM ResMovil y creación de la estructura de la memoria.
- **T5.2 Redacción de la memoria**: redactar los distintos apartados que la componen e incluir diagramas de flujo, gráficos y códigos generados en las distintas etapas del proyecto.
- **T5.3 Revisión final de la memoria del PFC**: revisión final y correcciones de errores identificados por parte del tutor.

Entregables (semana en que tendrá lugar):

- **E5.1 Memoria encuadernada del PFM (s8)**: 1 copia en papel y otra en formato electrónico.

Fases y Tareas del PFM	Semana 1							Semana 2							Semana 3							Semana 4							Semana 5							Semana 6							Semana 7							Semana 8						
	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7
1. Definición del Proyecto																																																							
T1.1 Identificación de necesidades para la elaboración de un PFM																																																						
T1.2 Estudio del estado del arte en el desarrollo de aplicaciones Java y tecnología ICE																																																						
T1.3 Estudio del estado del arte en el desarrollo de programas TinyOS																																																						
T1.4 Definición de objetivos iniciales en la elaboración del PFM																																																						
2. Planificación del Proyecto																																																						
T2.1 Descripción del comportamiento general de los dispositivos y los parámetros de diseño																																																						
T2.2 Definición de distintas alternativas para el diseño de un programa TinyOS																																																						
T2.3 Descripción del funcionamiento de un módulo que actúe como servidor de enlace																																																						
T2.4 Definición de distintas alternativas para del servidor de enlace																																																						
T2.5 Selección de un diseño																																																						
T2.6 División del PFM en tareas y fases																																																						
3. Diseño Conceptual																																																						
T3.1 Definición de los procesos que conforman el servidor de enlace																																																						
T3.2 Definición de los procesos que conforman el programa de las motas																																																						
T3.3 Integración de los procesos																																																						
T3.4 Revisión de los procesos definidos																																																						
4. Codificación de Procesos																																																						
T4.1 Definición de las interfaces y componentes TinyOS necesarios																																																						
T4.2 Programación de los componentes TinyOS																																																						
T4.3 Definición de las interfaces de los distintos objetos Java																																																						
T4.4 Programación de los objetos Java																																																						
T4.5 Verificación de la coherencia entre los diagramas de flujo y la programación implementada																																																						
T4.5 Integración del servidor de enlace como un módulo más del sistema de localización																																																						
5. Elaboración de la Memoria del PFM																																																						
T5.1 Definición de la estructura de la memoria del PFM																																																						
T5.2 Redacción de la memoria																																																						
T5.3 Revisión final de la memoria del PFC																																																					

Tabla 10. Diagrama de Gannt

5 RESULTADOS

Esta sección de resultados pretende reflejar cómo se aborda la doble faceta del objetivo general del Proyecto Fin de Máster, es decir, describir la parte del sistema basada en la tecnología *ZigBee*, es decir, el *Servidor de Enlace,* y el código que define el comportamiento de las propias motas.

5.1 *SERVIDOR DE ENLACE*

El *Servidor de Enlace* es el encargado de gestionar la comunicación entre el servidor y una o varias subredes inalámbricas *ZigBee* que se definen bajo su ámbito de control, para lo que contará con una conexión por cada estación base de cada una de estas subredes.

El bloque *Servidor de Enlace* fue creado para hacer frente a los siguientes objetivos y/o necesidades del sistema:

- Ofrecer una serie de funciones que permitan al resto de bloques del sistema enviar mensajes de control hacia las subredes inalámbricas.
- Interpretar los mensajes (*ZigBee*) que llegan a través de las estaciones base.
- Hacer llegar la información originada en los dispositivos inalámbricos al correspondiente bloque del sistema.
- Llevar un control del estado de los dispositivos inalámbricos que forman sus distintas subredes ('*keep-alive*').

Este bloque muestra, de forma natural, dos caras: una que mira hacia el resto de bloques del sistema y está construida en Java sobre tecnología Ice[8] ("cara Ice") y que, por tanto, permite asegurar los requisitos de escalabilidad y despliegue distribuido del sistema; y otra que atiende a las subredes *ZigBee* y que está programada en Java ("cara *ZigBee*").

Dado que se desconoce el momento de llegada de un mensaje desde un dispositivo, la *cara ZigBee* del *Servidor de Enlace* es, obviamente, asíncrona. Sin embargo, algunas veces el sistema necesitará atender peticiones provenientes de la *cara Ice* que requieran una respuesta originada en la red *ZigBee*. Debido a esto, una de las tareas del *Servidor de Enlace* será implementar un mecanismo de parada y espera, de manera que la *cara Ice* tenga un carácter sincrónico y las llamadas queden bloqueadas hasta que, o bien llega la respuesta, o bien se cumple un tiempo determinado.

[8] Ice (Internet Communication Engine) es un moderno middleware orientado a objeto para desarrollo en entornos distribuidos heredero de Soap y Corba. www.zeroc.com

5.1.1 LAS INTERFACES ICE

Tres son las interfaces definidas en la *cara Ice* y que permiten el acceso remoto a las funcionalidades que provee el *Servidor de Enlace*[9]: la interfaz 'Management' similar a las de otros bloques del sistema y que permiten ciertas tareas de gestión; la interfaz 'Scanner' con el propósito concreto de permitir el auto-entrenamiento de la red, del que ya se habló con anterioridad; y la interfaz 'Control' detallada a continuación.

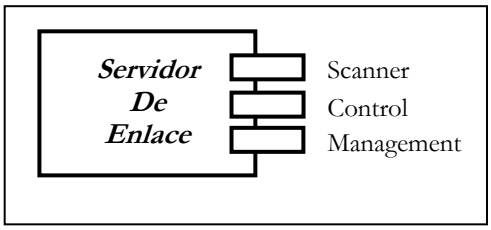


Fig 6. Interfaces públicas del Servidor de Enlace

5.1.1.1 La Interfaz Control

Cualquiera que necesite interactuar con la red *ZigBee* puede establecer conexión con el *Servidor de Enlace* y usar la interfaz 'Control', una interfaz muy simple con únicamente cuatro opciones. Estas opciones vienen descritas en la siguiente tabla y se clasifican claramente en dos grupos: funciones de gestión, para consulta y modificación de los valores de ciertos parámetros de los dispositivos *ZigBee*; y funciones de interacción, para el envío de mensajes que desencadenen cierto comportamiento en los dispositivos.

getParameter	Solicitar a un determinado dispositivo *ZigBee* el valor actual de uno o más parámetros. El *Servidor de Enlace* proveerá las funciones necesarias para realizar esta función sincrónica, es decir: - Interpretar los parámetros de entrada para conformar y enviar los mensajes necesarios hacia las subredes inalámbricas. - Esperar la llegada de cada una de las respuestas a través de las distintas conexiones con las estaciones base. - Devolver la lista de valores.
	Parámetros de entrada: - Identificador del dispositivo. - Lista de parámetros a consultar[10].
	Parámetros de salida: - Lista de valores.
setParameter	Establecer cierto valor de uno o más parámetros de un determinado dispositivo *ZigBee*.

[9] El **Anexo 2: Interfaces ICE del Servidor del Enlace** (Pág. 74) recoge la definición de las interfaces del *Servidor de Enlace* en el formato que ICE emplea para generar automáticamente el resto de códigos necesarios para su integración en el correspondiente entorno de programación (Java, C,…). Se incluye también la definición de la interfaz *básica y general de gestión* (*es::urjc::ceci::management::interfaces::**BasicManagement***)

[10] La lista de parámetros de un dispositivo ZigBee que pueden ser consultados y modificados se encuentra definida en el archivo *es.urjc.ceci.znet.brdigeserver.lib.tools.**ZBParameter.java***.

	El *Servidor de Enlace* implementará las funciones necesarias para proveer esta función sincrónica, es decir: - Interpretar los parámetros de entrada para conformar y enviar los mensajes necesarios hacia las subredes inalámbricas - Esperar la llegada de un mensaje de confirmación por cada uno de los mensajes enviados. - Informar del éxito o fracaso de la operación.
	Parámetros de entrada: - Identificador del dispositivo - Lista de parámetros a cambiar[11]. - Lista de valores
	Parámetros de salida: - Valor de éxito en la operación
CommandSync	Solicitar a los dispositivos tareas bajo demanda y esperar por cierta confirmación o resultado de la operación.
	Parámetros de entrada: - Identificador del dispositivo - Lista de comandos solicitados[12]: o *Blink*: provoca un parpadeo temporal. o *Check*: comprobar el estado del dispositivo o *Get Report*: solicita la recopilación y envío de un informe con medidas de potencia usando el tipo de preprocesado actualmente vigente en el dispositivo. o *Get Report FM*: solicita la recopilación y envío de un informe con medidas de potencia usando el tipo de preprocesado indicado. o *Get MReport*: múltiples informes de medidas. o *Get MReport FM*: múltiples informes de medidas con preprocesado explícito. - Lista de argumentos para los comandos
	Parámetros de salida: - Lista con los distintos valores de éxito en cada operación.
CommandAsync	Solicitar a los dispositivos tareas bajo demanda sin esperar ninguna confirmación o resultado de la operación.
	Parámetros de entrada:

[11] La lista de parámetros de un dispositivo ZigBee que pueden ser consultados y modificados se encuentra definida en el archivo *es.urjc.ceci.znet.brdigeserver.lib.tools.**ZBParameter.java***.

[12] La lista de comandos que un dispositivo ZigBee reconoce se encuentra definida en el archivo *es.urjc.ceci.znet.brdigeserver.lib.tools.**ZBCommand.java***.

	- Identificador del dispositivo - Lista de comandos solicitados[11]: o *Blink*: provoca un parpadeo temporal. - Lista de argumentos para los comandos
	Parámetros de salida: (no hay)

Tabla 11. Descripción de la interfaz 'Control' del Servidor de Enlace

5.1.1.2 La Interfaz Scanner

La siguiente tabla describe la interfaz "Scanner", especialmente creada para atender peticiones de medidas de potencia bajo demanda y que es aprovechada para proveer auto-entrenamiento.

refresh	Solicitar a un determinado dispositivo *ZigBee* la recopilación y envío de un informe de medidas de potencia. El *Servidor de Enlace* proveerá las funciones necesarias para realizar esta función sincrónica, es decir: - Interpretar los parámetros de entrada para conformar y enviar los mensajes necesarios hacia las subredes inalámbricas. - Esperar la llegada de cada una de las respuestas a través de las distintas conexiones con las estaciones base. - Conformar el informe de medidas.
	Parámetros de entrada: - Identificador del dispositivo.
	Parámetros de salida: - Valor dicótomo que indica el éxito de la llamada.
getTime	Consulta el momento en el que se logró satisfactoriamente y por última vez una medida de potencia.
	Parámetros de entrada: (no hay)
	Parámetros de salida: - Número equivalente a la fecha/hora del momento de la medida.
getMeasureSeq	Solicitar el informe de potencias una vez conformado a partir de las colección de medidas que el dispositivo envío.
	- Parámetros de entrada: (no hay)
	Parámetros de salida: - Lista de medidas de potencia.

Tabla 12. Descripción de la interfaz 'Scanner' del Servidor de Enlace

5.1.1.3 La Interfaz Management

Todo bloque del sistema tiene una interfaz 'Management' que permite ciertas tareas básicas de gestión remota. La siguiente tabla describe esta interfaz básica de gestión:

commit	Muchos de los parámetros de funcionamiento de los distintos bloques son modificables de forma remota. 'Commit' solicita que la configuración actual se almacene de forma local de manera que no se pierda en caso de que se de un reinicio del servidor.
	Parámetros de entrada: (no hay)
	Parámetros de salida: (no hay)
restart	Solicitar al sistema que termine su actividad y reinicie de nuevo.
	Parámetros de entrada: (no hay)
	Parámetros de salida: (no hay)
shutdown	Solicitar al sistema que de por concluida su actividad, inhabilite las interfaces y cierre las conexiones.
	Parámetros de entrada: (no hay)
	Parámetros de salida: (no hay)

Tabla 13. Descripción de la interfaz básica 'Management' del Servidor de Enlace

El *Servidor de Enlace* (como el resto de bloques) extiende esta interfaz básica de gestión, dando la posibilidad de consultar la lista de nodos que actualmente están en conexión directa, tanto estaciones base, como nodos *baliza* y móviles:

getGatewayList	Solicitar la lista de nodos pasarela que actualmente están conectados al *Servidor de Enlace*.
	Parámetros de entrada: (no hay)
	Parámetros de salida: - Lista de identificadores de dispositivo.
getBeaconList	Solicitar la lista de nodos *baliza* que actualmente están accesibles a través de una determinada pasarela conectada al *Servidor de Enlace*.
	Parámetros de entrada: - Identificadores de dispositivo de la pasarela en cuestión.
	Parámetros de salida: - Lista de identificadores de dispositivo.
getMobileList	Solicitar la lista de nodos móviles que actualmente están accesibles a través de una determinada pasarela conectada al *Servidor de Enlace*.

	Parámetros de entrada: - Identificadores de dispositivo de la pasarela en cuestión.
	Parámetros de salida: - Lista de identificadores de dispositivo.

Tabla 14. Descripción de la extensión de la interfaz básica 'Management' en el Servidor de Enlace

5.1.2 COMPONENTES DEL *SERVIDOR DE ENLACE*

5.1.2.1 ZigBee-Manager[13]

El *Servidor de Enlace* está dividido en un subconjunto de clases que estructuran sus distintas funcionalidades y que proveen sus propias interfaces para su interrelación y para su relación con otros bloques del sistema:

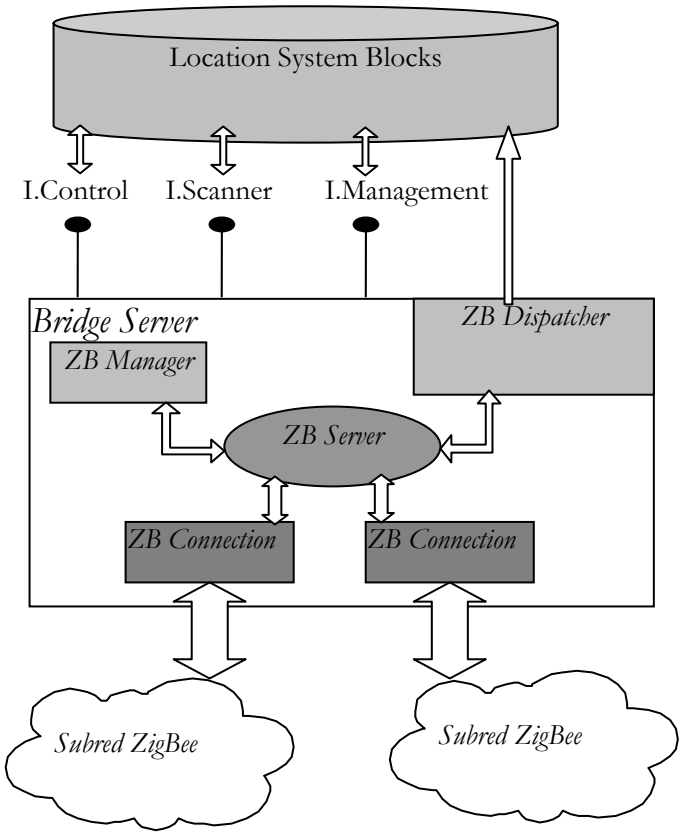


Fig 7. Esquema interno del Servidor de Enlace y sus interrelaciones con el resto del sistema.

[13] es.urjc.ceci.znet.*Servidor de Enlace*.zbmanager.zigbControl.**ZBManager.java**

Cada vez que una petición entrante por alguna de las interfaces Ice requiere del envío de mensajes a las subredes *ZigBee*, es atendida por un objeto *ZigBee-Manager* que implementa la siguiente interfaz:

```
public interface IZBManager {
        public Integer[] getParameter( int pDeviceId, ZBParameter[] pParamName );
        public ZBResult setParameter( int pDeviceId, ZBParameter[] pParamName, int[] psValue );

        public ZBResult syncBlink( int pDeviceId, int psRepetitions, int psPeriod );
        public ZBResult asyncBlink( int pDeviceId, int psRepetitions, int psPeriod );
        public ZBResult check( int pDeviceId);


        public Vector<ZBMeasurementObj> getReport( int pDeviceId );
        public Vector<ZBMeasurementObj> getReportWithFilterMode( int pDeviceId, int pFilterMode );
        public Vector<ZBMeasurementObj> getMReport( int pDeviceId, int psRepetitions, int psPeriod );
        public Vector<ZBMeasurementObj> getMReportWithFilterMode( int pDeviceId, int pFilterMode,
                                                                  int psRepetitions, int psPeriod );


        public int syncCallFunction(String psFunctionName, int[] piArgs);
        public int asyncCallFunction(String psFunctionName, int[] piArgs);
}
```

Tabla 15. Interfaz pública del ZigBee-Manager

La operación del *ZigBee-Manager* parece evidente si se piensa en el proceso desde que se llama a una de las funciones de la anterior interfaz hasta que se devuelve, o no, resultados. Antes de nada, el *Servidor de Enlace* identifica el tipo de llamada entrante por una de sus interfaces y crea un *ZigBee-Manager* para atenderla invocando una de sus funciones con los parámetros de entrada adecuados. Entonces el *ZigBee-Manager* debe construir y preparar el paquete *ZigBee* correspondiente, incluyendo cabeceras (tipo de paquete, dirección de destino…) y parámetros. Después debe enviar el mensaje y, en el supuesto que sea necesario, esperar por la respuesta, implementando el mecanismo de parada y espera, que más adelante se explicará con más detalle. La última tarea es liberar los recursos empleados.

5.1.2.2 ZigBee-Dispatcher[14]

Cuando llega un mensaje originado en la red *ZigBee* (informes de medidas, alarmas…) y no es la respuesta a ningún mensaje de control previo (ningún *ZigBee-Manager* espera la respuesta) entonces el *Servidor de Enlace* debe decidir a qué bloque del sistema debe encaminar la información contenida en dicho paquete. El objeto *ZigBee-Dispatcher* cumple este papel de repartidor. Cualquier evento que se añada a su cola de entrada, el *ZigBee-Dispatcher* lo identificará, extraerá la información y la enviará al bloque correspondiente del sistema.

A modo de ejemplo, podemos ver qué sucede con un informe de medidas de potencia. Éste debe hacerse llegar al bloque de *Motor de Localización* para que sea él quien calcule la posición geográfica del dispositivo que lo envió. El mensaje con las medidas junto con la estación base por la

[14] es.urjc.ceci.znet.*Servidor de Enlace*.zbmanager.zigbControl.**ZBDispatcher**

que llegó conforman el evento que será incluido en la cola de entrada del *ZigBee-Dispatcher*. Éste se encargará de:

- esperar por todos los fragmentos en el caso de que el informe esté compuesto por un número de medidas que excedan la capacidad de un mensaje;
- extraer la colección completa de medidas de potencia;
- establecer comunicación con la interfaz Ice adecuada del *Motor de Localización*;
- invocar la llamada que desencadenará el cálculo de la posición a partir de dicha colección de medidas (pasada como parámetro).

Dado que el sistema tiene un enfoque distribuido y puede emplearse la replicación de bloques como solución de escalabilidad, el uso de tipo de objetos, como el *ZigBee-Dispatcher*, tiene la ventaja de concentrar en un único componente la decisión de a cuál de todos los objetos distribuidos recurrir en cada momento, centralizando y facilitando la configuración del sistema. Continuando con el ejemplo anterior respecto al informe de medidas, podría darse el caso de que hubiera distintos bloques *Motor de Localización* por los que optar a la hora de enviar la colección de medidas. Queda así abierta la posibilidad de implementar un procedimiento de selección que contribuya a la optimización de ciertos parámetros del sistema, por ejemplo, balanceando la carga de trabajo computacional entre los distintos *Motor de Localización* disponibles.

5.1.2.3 ZigBee-Connection[15]

Tradicionalmente las redes *ZigBee* fueron pensadas para auto-organizarse en árboles lógicos en los que la información originada en los sensores es encaminada hacia una estación base que sirve de interconexión con la red donde se procesa y almacena dicha información. Y de igual manera, la información en el otro sentido, hacia los dispositivos, llega primero a la estación base que se encarga de difundirla por la subred. Nuestro sistema de localización también sigue este modelo con la salvedad de que los mensajes difundidos por la red *ZigBee* no son de información general para todos los dispositivos sino que suelen ser información de control específica para una determinada mota.

Los *ZigBee-Connection* son los objetos encargados de permanecer a la escucha de cualquier mensaje proveniente de los dispositivos *ZigBee* así como de hacer llegar los mensajes a la estación base a la que está permanentemente conectado para que ésta los difunda por la subred asociada. Su comportamiento es completamente asíncrono y orientado a evento, es decir, envía los mensajes hacia la subred *ZigBee* cuando así se solicita, y recoge los mensajes salientes de la subred cuando éstos llegan.

Envío y recepción de mensajes

El envío de mensajes es transparente: *ZigBee-Connection* transmite automáticamente el mensaje a la estación base tal y como se lo entregaron a él. Sin embargo en recepción es necesario cierto preprocesado por parte del *ZigBee-Connection*. Desde su punto de vista, los mensajes que llegan desde una subred sólo pueden ser de dos tipos, así que dos son las actuaciones posibles:

a) Originados en uno de los dispostivos *ZigBee* (informes de medidas, alarmas…)

   Estos mensajes deben remitirse a los distintos bloques del servidor, así que el *ZigBee-Connection* sólo debe incluir el mensaje como un nuevo evento a ser repartido por el *ZigBee-Dispatcher*. Ese evento incluye el mensaje entrante pero también el identificador de la correspondiente estación base por la que llegó.

[15] es.urjc.ceci.znet.*Servidor de Enlace*.zbmanager.zigbControl.**ZBConnection**

b) Respuestas a mensajes de control enviados previamente

Cuando el *ZigBee-Connection* observa que el bit de ACK viene puesto a '1' entiende que el mensaje es una respuesta a un mensaje de control previo. Si es así, el dispositivo *ZigBee* copió el identificador de mensaje para construir su respuesta, el mismo que el propio *ZigBee-Connection* generó en el momento del envío del mensaje de control. Por lo tanto, extrae dicho identificador de mensaje y busca en su lista algún *ZigBee-Manager* que pudiera estar esperando por esta repuesta.

Monitorización de dispositivos

Obviamente tanto las funciones de transmisión como de recepción de mensajes requieren que el *ZigBee-Connection* gestione los identificadores de mensaje enviados hacia la subred *ZigBee* y conozca cuál es el identificador de su estación base. Esto enlaza con la tercera de las funciones fundamentales de esta clase de objetos. Y es que, además de enviar y recibir mensajes, el *ZigBee-Connection* tiene la tarea de monitorizar el estado de los dispositivos asociados a su subred. Esta monitorización es, actualmente, un simple mecanismo de '*keep-alive*' que se divide en dos fases: una primera de iniciación de dispositivos y otra de monitorización periódica.

Cada vez que un dispositivo *ZigBee* se reinicia debe enviar un mensaje especial en el que incluye sus parámetros fundamentales de estado. Con este mensaje de INIT el dispositivo solicita permiso expreso para comenzar a funcionar dentro del sistema. Las estaciones base hacen lo mismo, pero distinguiendo su mensaje como ROOT_INIT y sin esperar la confirmación para comenzar a reenviar los mensajes desde y hacia el *ZigBee-Connection*. Cuando mensajes de INIT llegan al *ZigBee-Connection*, éste inscribe la dirección fuente del mensaje en la lista de dispositivos dentro de su subred y, en el caso de los ROOT_INIT, lo almacena como el identificador de la estación base a la que está conectado. Cada registro es acompañado de una marca de tiempo. Una vez hecho esto envía un mensaje de confirmación que, al llegar al dispositivo en cuestión, le permite integrarse en la subred y funcionar normalmente.

Hasta aquí el proceso de iniciación de los dispositivos. El *ZigBee-Connection* ya tiene su lista de dispositivos y su tarea se limita ahora a revisar la marca de tiempo para decidir si el comportamiento de alguno de ellos puede haberse alterado y si es necesario o no enviar un mensaje de reinicio (REBOOT)[16]. Lógicamente, la marca de tiempo debe ser actualizada y para ello se usa otro tipo de mensaje, muy similar al INIT, y que se denomina ALIVE. Este mensaje también incluye los parámetros de estado del dispositivo, que lo envía de forma periódica siempre y cuando no haya transmitido antes alguna otra información hacia el *ZigBee-Connection*. Así por ejemplo, aquella mota que continuadamente recoge y envía informes de medidas de potencia no necesita además transmitir un mensaje ALIVE porque todos esos informes, obligatoriamente, llegarán al *ZigBee-Connection* que podrá saber que, efectivamente, ese dispositivo está trabajando correcta y normalmente. Y exactamente lo mismo se aplica a la estación base que sin su correcto funcionamiento ningún mensaje alcanzaría al *ZigBee-Connection*.

5.1.2.4 ZigBee-Server[17]

En el momento de arrancar el *Servidor de Enlace* y crear sus interfaces Ice para la conexión con el resto del sistema, se crea un objeto *ZigBee-Server* que se convertirá en el núcleo del bloque y hará de lazo de unión centralizado entre los distintos elementos vistos anteriormente.

[16] De revisar los sellos de tiempo y enviar el mensaje de inicio se encarga una clase definida dentro del propio es.urjc.ceci.znet.bridgeserver.zbmanager.zigbControl.ZBConnection y que se le ha llamado ***MoteDaemon***

[17] es.urjc.ceci.znet.*Servidor de Enlace*.zbmanager.zigbControl.**ZBServer**

El *ZigBee-Server* será el encargado de crear y gestionar cada una de las conexiones con las estaciones base bajo el dominio del *Servidor de Enlace*. Su principal función es facilitar y administrar el flujo de mensajes en ambos sentidos, ampliando las posibilidades de optimización y balanceo de carga:

- *Sentido hacia la red inalámbrica* (*downlink*): permitir el envío de mensajes por parte de cualquier *ZigBee-Manager* de forma transparente al número de estaciones base existentes. El *ZigBee-Manager* simplemente usa la interfaz del *ZigBee-Server* que será el que administre la forma de encaminar ese mensaje por las distintas subredes a través de las estaciones base.
- *Sentido desde la red inalámbrica* (*uplink*): permitir la entrega al *ZigBee-Dispatcher* o correspondiente *ZigBee-Manager* de los eventos detectados por cada *ZigBee-Connection*.

5.1.3 EL MECANISMO DE PARADA Y ESPERA

Como se mencionó anteriormente, se hace necesario un mecanismo de parada y espera para las llamadas bloqueantes de la interfaz 'Control'. La forma de implementarlo es relativamente simple, hace uso de colas y ejecuta los siguiente pasos:

1- El *ZigBee-Manager* cuando necesita enviar un mensaje que requiere una respuesta o confirmación, prepara el mensaje y se lo pasa al *ZigBee-Server* junto con una cola donde espera sea guardada la respuesta una vez llegue.

2- El *ZigBee-Server* será el encargado de, antes de enviar el mensaje por cada una de las estaciones base, pasar dicha cola a cada *ZigBee-Connection*.

3- Éstos indexarán la cola asociándole un identificador de llamada que el *ZigBee-Server* colocará en el campo correspondiente de la cabecera del mensaje.

4- En el momento en que la respuesta llega por alguna de las estaciones base, el *ZigBee-Connection* que las atiende extrae la cola de recepción mediante el identificador de llamada que viene en el mensaje.

5- Una vez almacenado el mensaje en la cola, el *ZigBee-Manager* sólo tiene que recogerlo y gestionarlo, esperando por más fragmentos si son necesarios o recuperando la información y liberando la llamada bloqueada que esperaba una respuesta.

6- Concluida la comunicación, el *ZigBee-Manager* solicitará al *ZigBee-Server* que se liberen todos aquellos recursos empleados, de manera que su cola desaparecerá de todas los *ZigBee-Connection* y los identificadores de llamada podrán ser reutilizado.

5.2 MOTAS

5.2.1 COMPONENTES *TINYOS*

5.2.1.1 MoteManager

El *MoteManager* es el componente principal de la mota, y el que sirve de lazo de unión entre el resto de los componentes.

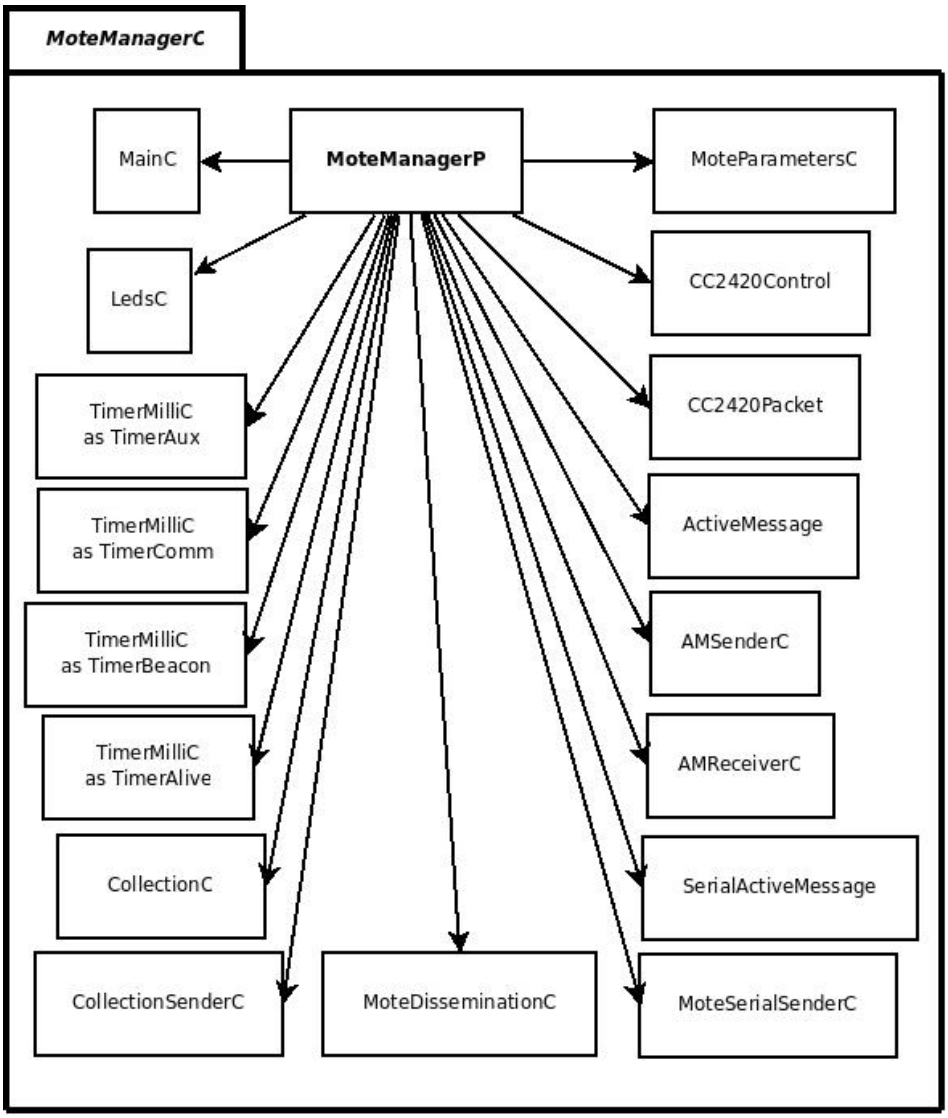


Fig 8. Diagrama con los distintos componentes que usa el MoteManager.

Este componente es el encargado de realizar las tareas propias al modo de comportamiento activo en cada momento, y atender a los mensajes de control que lleguen a la mota, realizando la acción correspondiente.

5.2.1.2 Parámetros 'Dinámicos': MoteParameters

El funcionamiento normal de la mota depende de ciertos parámetros generales: unos permanecerán con el mismo valor constante a lo largo de la vida del *programa*; otros, aquí llamados "dinámicos", por el contrario, serán susceptibles de ser consultados y modificados dinámicamente por el gestor desde el *servidor*.

Fiel al objetivo general de flexibilidad de este proyecto, la idea principal no es tanto dejar una serie cerrada de parámetros configurables sino, más bien, proveer cierta herramienta que permita consultar y modificar un conjunto abierto y ampliable de ellos. Así, para esta versión inicial del proyecto, se ha considerado suficiente definir un subconjunto de los posibles parámetros *dinámicos* (subconjunto que se verá con más detalle a continuación) que incluye la potencia de transmisión del mensaje de *beacon*, el tiempo entre *beacon*, el tiempo que un mota *móvil* emplea en recopilar medidas, el preprocesado que la mota hará a las medidas antes de preparar el informe e incluso el identificador de la mota. Sin embargo no están en esta lista, aunque quizás podría ser interesante incluirlas como variables en un futuro, parámetros como los canales para comunicaciones y para *beacon*, o la potencia de transmisión en el canal de comunicaciones, por ejemplo.

El contemplar la posibilidad de modificar los parámetros generales entraña dos aspectos que deben ser considerados:

- El acceso controlado a las variables
- El almacenamiento no volátil de la configuración para evitar pérdidas de cambios dinámicos.

Ninguna de estos dos aspectos necesitan mucha justificación y ambos son provistos por el componente *MoteParameters* que se explica a continuación, así que ahora sólo se esbozará el procedimiento para resolver la inicialización de parámetros con almacenamiento no volátil[18]:

0. Como paso previo a cualquier acción a realizar por el dispositivo, y tal y como se explica en la documentación de *TinyOS*, se define el tamaño de la memoria que se reserva para almacenamiento no volátil mediante un fichero ***volumes-XXX.xml***[19], usado durante la compilación del programa.

1. Ahora sí, el primer paso una vez el dispositivo se pone en funcionamiento es *montar* la zona de memoria no volátil. Si se detecta algún problema en este paso se dará por imposible el almacenamiento volátil y, como en el resto de los pasos, se irá al *punto 5* para establecer la configuración por defecto.

2. Una vez la zona está montada con éxito es necesario validar los datos que ésta guarda, pues podría ser la primera vez que recurrimos a esta memoria y que ésta estuviera vacía. Si esta comprobación del sistema operativo resulta en fallo se irá al *paso 5*, si bien antes es recomendable hacer un intento previo de *commit* para conocer si el almacenamiento volátil es realmente posible.

3. Si los datos pasan la comprobación, es momento de lanzar una lectura completa de los parámetros almacenados.

4. Si la lectura termina sin problemas, el programa puede ya manejar normalmente los valores de la configuración almacenada. Entre los parámetros guardados debe estar el número de versión del *programa.*

   Si la versión del programa actual concuerda con la versión salvada se establecerá la configuración recuperada y este cuarto paso será el último.

   Si, por lo contrario, ambas versiones difieren, significaría que se ha reprogramado el dispositivo y se considerará que los parámetros antiguos pueden estar obsoletos así que la nueva configuración por defecto tiene preponderancia, pasando al *punto 5*.

[18] Ver *Diagramas del Proceso de Inicialización de Parámetros* en **Anexo 1: Diagramas** (Pág. 65)

[19] Fichero ***volumes-at45db.xml*** para MicaZ, y ***volumes-stm25p.xml*** para TelosB

5. Llegado a este punto, no se estableció la configuración salvada y es necesario dar unos valores por defecto a los parámetros *dinámicos*.[20]

Para salvar la configuración actual es necesario saber si tras el proceso anterior se concluyó que el almacenamiento volátil es posible. Si es así, los pasos son sólo dos[21]:

1. Solicitar la escritura de la configuración en la zona no volátil.
2. Validar los datos escritos mediante un *commit*.

En realidad los parámetros no se leen ni se escriben de forma individual sino que la configuración se maneja en bloque con la siguiente estructura[22]:

```
typedef struct mote_config_t {
      uint8_t version;
      uint16_t tos_node_id;
      uint8_t state;
      uint8_t power;
      uint32_t tiempoME;  // Tiempo ModoEspera
      uint32_t tiempoTick;      // Tiempo Tick
      uint32_t tiempoRB;  // Tiempo Recibir Beacon
      uint32_t tiempoAlive;     // Tiempo Alive
      uint32_t tiempoInit;      // Tiempo Init
      uint8_t filter;     // Preprocesado
} mote_config_t;
```

Tabla 16. Estructura con los parámetros de configuración de la mota

Componente MoteParameters

Este componente ofrece una interfaz para la gestión de los parámetros básicos del funcionamiento de la mota, controlando el acceso asíncrono a los mismos y posibilitando una sencilla forma de manejar valores siempre actualizados. Además se encarga de guardar automáticamente la configuración actual en una zona de memoria no volátil para evitar que modificaciones dinámicas de los valores de los parámetros generales de funcionamiento se pierdan en cualquier reinicio del dispositivo.

La interfaz de este componente dependerá, fundamentalmente, de la lista de parámetros *dinámicos* de la mota que son los que éste gestionará. Estos parámetros pueden ser consultados y modificados bajo demanda, por lo que esta interfaz estará en general compuesta por pares de funciones `Get` y `Set`, en correspondencia con los paquetes de control `Get` y `Set` definidos para la comunicación entre la mota y el *servidor*. Es para esta comunicación por lo que se definen unos códigos que identifican el tipo de parámetro en ambos extremos[7]:

Parámetro	Código	Descripción / Interfaz
Power	0x00	`PARAM_POWER` es el valor de la potencia a la que se deben transmitir los mensajes *baliza* vía radio. En las motas *MicaZ*, los valores posibles están comprendidos entre 3 y 31: `#TXPOWER_0DBM 0x1f //0dBm`

[20] Los valores por defecto de los parámetros y sus códigos se definen en el fichero *../include/**LBSconfig.h***

[21] Ver *Diagramas del Proceso de Almacenamiento No Volátil de la Configuración* en **Anexo 1: Diagramas** (Pág. 65)

[22] Esta estructura para la configuración está definida en el fichero *../MoteParameters/**MoteParameters.h***

		`#TXPOWER_M3DBM 0x23 //-3dBm`
		`#TXPOWER_M5DBM 0x19 //-5dBm`
		`#TXPOWER_M10DBM 0x0B //-10dBm`
		`#TXPOWER_M15DBM 0x07 //-15dBm`
		`#TXPOWER_M25DBM 0x03 //-25dBm`

		`command uint8_t` **`getPower();`** `command void` **`setPower(uint8_t piPower);`**
Identificador	0x01	`PARAM_ID` es el identificador único que cada dispositivo debe tener y que, en un principio, le será dado durante la programación de su programa.
		`command uint16_t` **`getDeviceId();`** `command void` **`setDeviceId(uint16_t piDevId);`**
Estado/Modo	0x02	`PARAM_STATE` es una variable que permite distinguir entre modos de funcionamiento para el mismo dispositivo. Consiste en un registro de 1 byte cuyos bits tienen un significado booleano independiente: • Started (0x01): un dispositivo establecerá este bit a '1' cuando detecte que está conectado a la red de sensores. • Blink (0x04): este bit estará a '1' cuando el dispositivo esté ejecutando una orden de 'blink', en la cual debe encender y apagar un número de veces todos los *leds* y, por lo tanto, se inhabilitan los informes habituales mediante *leds*. • Bridge (0x80): un dispositivo en modo 'Bridge' acturará como pasarela entre la red *ZigBee* y otra red de otra tecnología (generalmente IP) • Beacon (0x40): cuando un dispositivo actúa como *baliza* enviará periódicamente un mensaje de *beacon* que servirá a las motas móviles para formar su informe de medidas. • Mobile (0x20): un dispositivo *móvil* recolectará mensajes de *beacon* para enviar un informe de medidas de *RSSI* que permita calcular su posición geográfica. • TempMobile (0x20): cualquier dispositivo estará en este estado una vez recibe una solicitud de informe bajo demanda (`GET_REPORT_MSG`) y pasa a recolectar mensajes de *beacon* para generar el informe de medidas de *RSSI* solicitado.
		`command bool` **`isBridgeMode();`** `command bool` **`isTempMobileMode();`** `command bool` **`isMobileMode();`** `command bool` **`isBeaconMode();`** `command bool` **`isBlinkMode();`** `command bool` **`isStartedMode();`** `command void` **`setBridgeMode(bool pbFlag);`** `command void` **`setTempMobileMode(bool pbFlag);`** `command void` **`setMobileMode(bool pbFlag);`** `command void` **`setBeaconMode(bool pbFlag);`** `command void` **`setBlinkMode(bool pbFlag);`** `command void` **`setStartedMode(bool pbFlag);`** `command uint8_t` **`getState();`** `command void` **`setState(uint8_t pState);`** `command void` **`initState();`**
Batería	0x03	`PARAM_BATTERY` guarda el valor más actual de la cantidad de batería que aún conserva el dispositivo. Se puede decidir si el dispositivo sondeará periódicamente o bajo demanda el valor actual de su batería. Y también si se incluye dicho valor

		en los mensajes que el dispositivo envíe hacia el servidor.
		`command uint16_t` **`getBattery();`**
Filter	0x0A	`PARAM_FILTER` contiene el tipo de filtro. Cuando una mota conforma su informe con las medidas recolectadas puede hacer o no un preprocesado. De nuevo se usará un único byte en el que algunos de los bits tendrán valores independientes del resto: • Last One (0x01): la medida final que se envía es la última medida recibida de cada estación *baliza* (incompatible con filtros '*average*' y '*mobile*'), MFi[n] = LOi[n] = Xi[n] • Average (0x02): la medida final es la media aritmética de las medidas recibidas de cada estación *baliza* (incompatible con filtros '*last one*' y '*mobile*'), MFi[n] = AVi[n] = SUM(Xi[n]) / N • Mobile (0x03): la medida final es una media ponderada de las medidas recibidas de cada estación *baliza*, de manera que la última medida tiene un peso algo mayor (incompatible con filtros '*average*' y '*last one*'), MFi[n] = MOBi[n] = (Xi[n]+MOBi[n-1]) / 2 • Every Neighbour (0x40): independientemente del filtro escogido ('*last one*', '*average*', o '*mobile*'), la mota envía en el informe la medida de cada una de las estaciones *baliza* que haya podido recolectar (incompatible con el filtro '*Strongest*'). ENB[n] = MFi[n] • Strongest (0x80): independientemente del filtro escogido ('*last one*', '*average*', o '*mobile*'), la mota únicamente envía el informe de la estación *baliza* cuya medida final resultó máxima (incompatible con el filtro '*Every Neighbour*'), STR[n] = MAX[MFi[n]]
		`command uint8_t` **`getFilter();`** `command bool` **`isStrongestFilter();`** `command bool` **`isEveryNeighbourFilter();`** `command bool` **`isLastOneFilter();`** `command bool` **`isAverageFilter();`** `command bool` **`isMobileFilter();`** `command void` **`setFilter(uint8_t piFilter);`** `command void` **`setFilterMode(uint8_t piFilterMode);`** `command void` **`setStrongestFilter(bool pbFlag);`** `command void` **`setEveryNeighbourFilter(bool pbFlag);`**
Tiempo en Modo de Espera	0x0B	`PARAM_TIEMPO_ME` indica al dispositivo cuánto tiempo debe estar en modo de espera, es decir, tiempo entre intentos de establecer conexión con la red, o, si ya está conectado, recibiendo paquetes de control o enviando informes de medidas si es necesario.
		`command uint32_t` **`getTiempoME();`** `command void` **`setTiempoME(uint32_t piTiempo);`**
Tiempo entre	0x0C	`PARAM_TIEMPO_TICK` indica al dispositivo cuánto tiempo debe guardar

Medidas		entre medidas, y que aprovechará para permanecer conectado a la red recibiendo paquetes de control o enviando informes de medidas si es necesario.
		`command uint32_t` **`getTiempoTick();`** `command void` **`setTiempoTick(uint32_t piTiempo);`**
Tiempo para Recolectar Balizas	0x0D	`PARAM_TIEMPO_RB` indica el tiempo destinado a la recolección de paquetes de *beacon* provenientes de las estaciones *baliza*.
		`command uint32_t` **`getTiempoRB();`** `command void` **`setTiempoRB(uint32_t piTiempo);`**
Tiempo entre mensajes *Alive*	0x0E	`PARAM_TIEMPO_ALIVE` indica al dispositivo cuanto tiempo debe guardar antes de enviar un mensaje '*alive*' para dar a conocer al servidor su correcta conexión a la red y su estado de funcionamiento.
		`command uint32_t` **`getTiempoAlive();`** `command void` **`setTiempoAlive(uint32_t piTiempo);`**
Tiempo entre mensajes *Init*	0x0F	`PARAM_TIEMPO_INIT` indica al dispositivo cuanto tiempo debe guardar antes de enviar un mensaje '*init*' que sirve tanto para dar a conocer al servidor la correcta conexión a la red del dispositivo como para solicitar el mensaje de consentimiento para empezar a funcionar en el modo correspondiente ('*mobile*', '*beacon*', o '*bridge*').
		`command uint32_t` **`getTiempoInit();`** `command void` **`setTiempoInit(uint32_t piTiempo);`**

Tabla 17. Descripción de los distintos parámetros y su código asociado

Además de las funcionalidades anteriores, el componente de *MoteParameters* gestiona otros parámetros *no dinámicos* de carácter general y de funcionamiento interno de la mota para los que incluye algunos métodos en su interfaz:

`command uint8_t` **`getProgVersion();`**	Proporciona la versión del programa actualmente instalado en el dispositivo (`LBS_MOTE_PROG_VERSION`).
`command uint16_t` **`getLastCallId();`** `command uint16_t` **`getNewCallId();`**	Cuando los mensajes originados en la mota requieren de un identificador univoco, el componente *Paremeters* proporciona una implementación general que asegure el acceso asíncrono a dicha variable.
`command void` **`initParameters();`**	Inicializa los valores de los parámetros según la configuración guardada o a sus valores por defecto, tal y como se establezca en el fichero de configuración general (***./include/LBSconfig.h***)

Tabla 18. Otras funciones de la interfaz de MoteParameters

5.2.1.3 Componente MoteDissemination

El concepto general y más básico de la red de sensores es de una red auto-organizada de dispositivos que se comportan como fuentes de información (la recogida del medio que les rodea) que es encaminada hacia un nodo sumidero de características probablemente especiales y que suele servir de puerta de enlace con alguna otra clase de red que lo conecta con algún equipo monitor.

La mayor parte de componentes que se encuentran en *TinyOS* están creados de acuerdo a este concepto y ésta es la razón por la que no existe ningún tipo de protocolo implementado que permita llevar información de manera eficiente no desde, sino hacia, los sensores.

Sí existe la difusión de información por inundación, y una implementación se encuentra en el componente *Dissemination* de *TinyOS*.

MoteDissemination no es más que una sencilla transformación de las interfaces de este componente de difusión de *TinyOS* a interfaces *Receive* y *Send*, que, como ya se he indicado anteriormente, son las más habituales en el envío y recepción de mensajes.

El objetivo de este cambio es facilitar en la medida de lo posible una futura incorporación de algún componente que, proveyendo las típicas interfaces *Receive* y *Send*, implemente algún tipo de mecanismo más eficiente y escalable que la inundación para hacer llegar información hasta los nodos más alejados de la raíz del árbol lógico.

5.2.2 MENSAJES

Como se ha visto más arriba, las interfaces *Receive* y *Send*, manejan mensajes de una determinada clase a la que han sido suscritas. Eso quiere decir, por ejemplo, que el evento **receive** de la interfaz *Receive* no se producirá mientras no llegue un mensaje con ciertas características, ignorando el resto.

Para poder enviar distintos tipos de mensaje y no tener que definir una interfaz *Receive* para cada uno de ellos, se ha creado una clase general de mensaje especialmente definida para la aplicación de localización. Se llama ***LBSmessage_t*** y su definición es la siguiente:

```
typedef nx_struct LBS_header_t {
      nx_uint8_t type;
      nx_uint16_t callId;
      nx_uint16_t device_id;
} LBS_header_t;


enum {
      LBSMESSAGE_DATA_LEN = MAX_LBSMESSAGE_DATA_LEN - sizeof(LBS_header_t),
};

typedef nx_struct LBSmessage_t {
  LBS_header_t header;
  nx_uint8_t data[LBSMESSAGE_DATA_LEN];
} LBSmessage_t;
```

Tabla 19. Definición de la estructura general de los mensajes

El resto de mensajes heredan este tipo general de mensaje, conservando en primer lugar los bytes asociados a la cabecera y variando la estructura lógica del campo de datos[23]. En realidad, la posición del campo '*type*' es la más crucial, pues permite conocer el tipo de mensaje y, por tanto toda su estructura, únicamente leyendo el primer byte del mensaje. De esta manera se deja abierta la posibilidad de variar libremente la estructura del mensaje siempre y cuando el primer byte lo identifique sin error.

Campo 'Type' (1 byte)							
7	6	5	4	3	2	1	0
Frag.	Ack	D / C	Data / Control type				

D / C	
1	Data
0	Control

D / C	Data type					
1	1	0	0	0	0	REPORT_MSG
1	1	0	0	0	1	GET_REPORT_MSG

D / C	Control type					
0	0	0	0	0	1	GET_MSG
0	0	0	0	1	0	SET_MSG
0	0	0	0	1	1	ALARM_MSG
0	0	0	1	x	x	(4 disponibles)
0	0	1	0	0	x	(2 disponibles)
0	0	1	0	1	0	BEACON_MSG
0	0	1	0	1	1	(1 disponibles)
0	0	1	1	x	x	(3 disponibles)
0	0	1	1	1	1	BLINK_MSG
0	1	0	0	x	x	(4 disponibles)
0	1	0	1	0	0	INIT_MSG
0	1	0	1	0	1	ROOT_INIT_MSG
0	1	0	1	1	0	ALIVE_MSG
0	1	0	1	1	1	ROOT_ALIVE_MSG
0	1	1	x	x	x	(7 disponibles)
0	1	1	1	1	1	REBOOT_MSG

23 La definición de los mensajes pueden encontrarse en el archivo *./include/**LBSmsg.h***

Fig 9. Descripción del uso del campo 'Type' de la cabecera general de los mensajes

5.2.2.1 Atendiendo mensajes de control

Cuando se recibe un paquete, en función del primer byte se atiende de tres formas posibles: como mensaje con el bit ACK activado, como mensaje de control o como mensaje de datos. Lo habitual será que la mayor parte de mensajes recibidos por una mota (a excepción de la estación base) sean de control:

```
typedef nx_struct LBScontrolMsg_t {
        LBS_header_t header;
        nx_uint8_t dataLen; // numero de Parametros
        LBSparameter_t parameter[NUM_MAX_PARAMETROS_POR_CONTROL];
} LBScontrolMsg_t;
```

Tabla 20. Definición de la estructura de los mensajes de control

Por lo general, los paquetes de control son mensajes de consulta (`Get`) o de modificación (`Set`) de uno o varios (según indique el campo `dataLen`) parámetros y que se especifican en el campo `parameter`, mediante un conjunto de estructuras (`LBSparameter_t`) que en realidad no son más que pares de nombre-valor. Una vez identificado el mensaje como paquete de control, la mota copiará el valor actual del parámetro correspondiente en caso de tratarse de un mensaje `Get` o cambiando la configuración actual asignando el valor incluido el mensaje `Set`.

En cualquier caso, y recorridos los `dataLen` parámetros, se activa el bit de ACK, dejando el resto de la cabecera igual (el mismo `callId`) y se reenvía el paquete al servidor, que lo considerará bien como mensaje de confirmación (en el caso de los `Set`), bien como respuesta con los valores consultados (en el caso de los `Get`) [24].

5.2.3 PROGRAMACIÓN DE DISPOSITIVOS YA DESPLEGADOS: COMPONENTE DELUGE

Un componente importante y que en un futuro es completamente necesario incluir es el *Deluge*[25]. Su función es proveer reprogramación de los dispositivos una vez ya han sido desplegados y sin necesidad de conectarlos uno por a uno a la plataforma programadora. Esta posibilidad se hace completamente necesaria en despliegues con pretensiones de larga duración en, los que, muy probablemente, se deban introducir cambios y/o mejoras en el programa de los dispositivos.

Desde un punto de vista de más alto nivel, el *Deluge* sigue unos pasos bien definidos a la hora de llevar a las distintas motas un nuevo programa compilado:

---

[24] Ver *Diagrama de la Gestión de los Mensajes de Control* en **Anexo 1: Diagramas** (Pág. 65)

[25] En *TinyOS 2.x* este componente se llamó *Deluge T2* y un manual de uso rápido se puede encontrar en la página http://docs.tinyos.net/index.php/Deluge_T2. Para información algo más detallada también se puede consultar "**Deluge 2.0 - TinyOS Network Programming Manual**" en http://www.cs.berkeley.edu/~jwhui/deluge/documentation.html

0- El paso previo a cualquier reprogramación es proporcionar a todas los dispositivos desplegados (incluidas las estaciones base) el mínimo código *Deluge* que sirva para:

   a) Dividir la memoria y reservar cierta partición para el almacenamiento de los nuevos programas.

   b) Proveer de los mecanismos y protocolos básicos para la recepción de nuevos programas.

1- Enviar el nuevo programa compilado a la estación base para que la almacene en la partición indicada.

2- Dar la orden a la estación base para que difunda el código almacenado por todos los dispositivos desplegados.

3- Difundir a través de la estación base un mensaje de reinicio a todas motas que haga que los dispositivos comiencen a funcionar según lo dispuesto en el nuevo código.

Actualmente este componente no está implementado en las motas porque se detectó un problema de incompatibilidad entre la batería empleada y la energía necesaria para grabar de forma totalmente fiable el nuevo código en la memoria no volátil. Sin embargo, se pudo comprobar que no existía ningún problema si los dispositivos contaban con un suministro de energía externo.

6 CONCLUSIONES

A la conclusión de este proyecto, se ha conseguido desarrollar el software necesario para poner en marcha un sistema para la experimentación en localización en redes inalámbricas *ZigBee*, incluidos los procedimientos para su gestión mediante una colección abierta de parámetros y mensajes entre servidor y dispositivos. Un sistema suficientemente flexible, escalable y fácil de usar y desplegar que posibilita la realización inmediata de experimentos que proporcionen resultados prácticos para el análisis de variables que pueden influir en la precisión del cálculo de posición, como pueden ser la densidad de nodos, la orientación de las antenas, la potencia de transmisión, el nivel instantáneo de batería disponible, el número de medidas tomadas, etc. Además, se deja la puerta abierta a la introducción de diferentes motores localización (redes neuronales, máquinas de vector soporte…), el uso de la herramienta de auto-entrenamiento, así como extender la investigación a posicionamiento en tres dimensiones incluyendo la coordenada Z en los cálculos.

En concreto, se ha creado un *Servidor de Enlace* capaz de unir eficaz y distribuidamente el sistema de localización a una red *ZigBee,* y se ha programado los dispositivos que la conforman. Programación lo suficientemente abierta como para posibilitar a las motas atender todas las peticiones que el sistema pudiera tener en relación a las medidas necesarias para la localización pero también para la gestión de la propia red.

Con todo ello se ha contribuido a la construcción de un sistema de localización en interiores capaz de dar servicio tanto a tareas de investigación en este campo como a posibles aplicaciones comerciales.

7 TRABAJO FUTUROS

A lo largo de las diferentes pruebas realizadas, se han observado en el comportamiento normal de las motas ciertas inconsistencias que no siempre se veían solventadas por el mecanismo de monitorización (*keep-alive*) del sistema. Antes de concluir cualquier problema hardware, se hace necesario un análisis detallado del sistema, en especial del código que gobierna los sensores inalámbricos. Ya se ha comenzado dicho análisis y se ha podido constatar la dificultad que entraña encontrar puntos críticos en el código *TinyOS* donde pudieran darse conflictos entre procesos. Los diagramas de flujo tradicionales no se adaptan bien al modelo orientado a eventos de los sistemas hardware embebidos, por lo que ya existen intentos de adoptar el enfoque SDL (*Specification and Description Language*) tratando de buscar equivalencias entre los distintos conceptos ambos mundos [Dietterle, 2004] [Dietterle, 2008]. Así por ejemplo, cada proceso SDL tendría su correspondencia en un componente *TinyOS*, y las señales que comunican distintos procesos en los distintos commands y events de sus interfaces.

Sea como fuese el análisis del sistema no sólo debería proporcionar ideas claras de cómo lograr códigos *TinyOS* fiables, robustos y reutilizables, sino también una valoración del aprovechamiento de los recursos del dispositivo inalámbrico así como de la eficiencia energética del modelo adoptado como sistema de localización.

Como última idea a resaltar, incidir en la experiencia que este proyecto ha proporcionado al grupo de trabajo, tanto en el desarrollo de aplicaciones *TinyOS*, para la parte de sensores inalámbricos integrados, como en IP, Java y ICE, para el núcleo del sistema modular y distribuido. Esto permite plantear objetivos futuros más complejos, flexibles y eficientes, y con un coste en tiempo más favorable. Algunos ejemplos podrían ser el diseño y creación de un protocolo eficiente para el encaminamiento en ambos sentidos entre los dispositivos y la estación base, que sustituiría a la difusión indiscriminada de mensajes en la red, o la introducción de políticas de balanceo de carga computacional entre los distintos bloques del sistema para hacer frente a posibles situaciones de gran número de dispositivo a localizar y/o peticiones a atender.

8 REFERENCIAS

LOCALIZACIÓN

- [Place] *Place Lab - www.placelab.org*
- [Bahl] *RADAR: An In-Building RF-based User Location and Tracking System.* ***Paramvir Bahl and Venkata N. Padmanabhan****. 2000.*
- [Rosum] - *www.rosum.com/rosum_tv-gps_indoor_location_technology.html*

ZIGBEE

- [Moon] *System Design Considerations for a ZigBee RF Receiver with regard to Coexistence with Wireless Devices in the 2.4GHz ISM-ban.* **Hae-Moon Seo**, **Yong-Kuk Park**, **Woo-Chool Park**, **Dongsu Kim**, **Myung-Soo Lee**, **Hyeong-Seok Kim** and **Pyung Choi**. KSII Transactions On Internet And Information Systems. Vol. 2, Nº1. Feb 2008.

TINYOS

- [TinyOS] *TinyOS* - www.tinyos.net, y *Tutorial* - http://es.wikipedia.org/wiki/TinyOS
- [Dietterle, 2004] *Mapping of High-Level SDL Models to Efficient Implementations for TinyOS*. **Daniel Dietterle, Jerzy Ryman, Kai Dombrowski, Rolf Kraemer**. IHP, Im Technologiepark 25, D-15236 Frankfurt(Oder), Germany. 2004
- [Dietterle, 2008] *Embedded system protocol design flow based on SDL: from specification to hardware/software implementation.* **Daniel Dietterle**. IHP microelectronics GmbH, Wireless Communication Systems. 2008

CROSSBOW TECHNOLOGY

- [XBow] *Crossbow* - www.xbow.com
- [MicaZ] Módulos *MicaZ* (hoja de características) –

www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf

- [MIB510] *Plataforma Serie* (hoja de características) –

www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB510CA_Datasheet.pdf

- [MIB520] *Plataforma USB* (hoja de características) –

www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB520_Datasheet.pdf

- [MIB600] *Plataforma Ethernet* (hoja de características) –

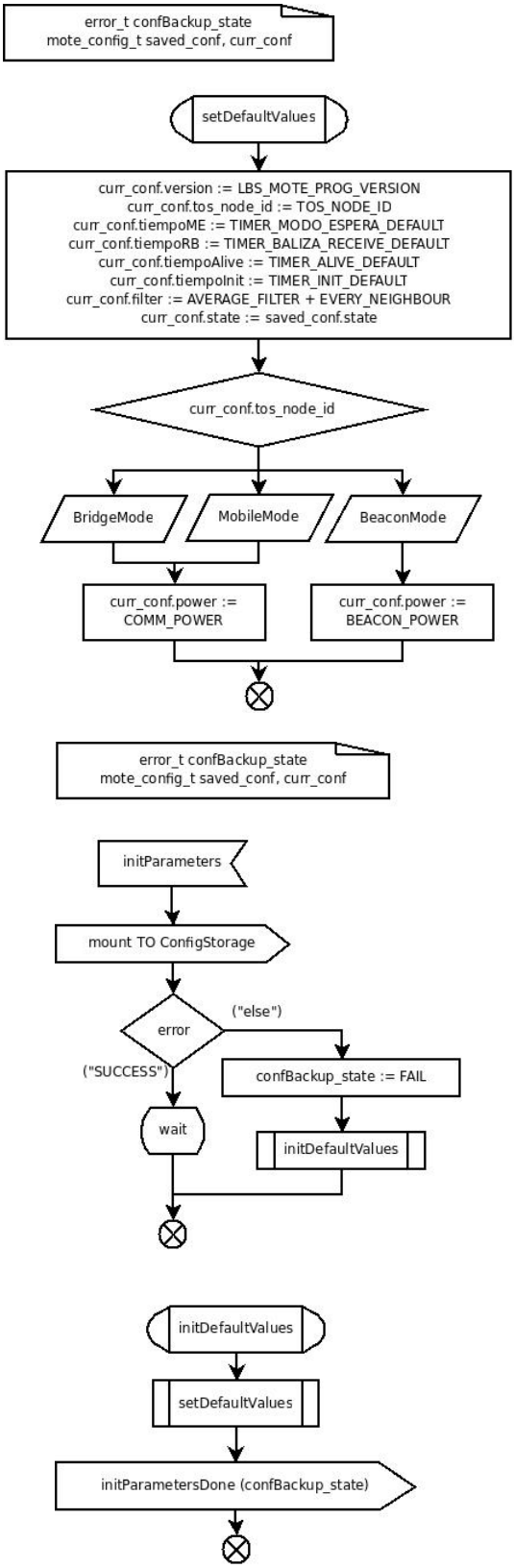
www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB600CA_Datasheet.pdf

- [MTS] *Placa de sensores de la familia* MTS *(*MTS300 *y* MTS310*)* (hoja de características) –

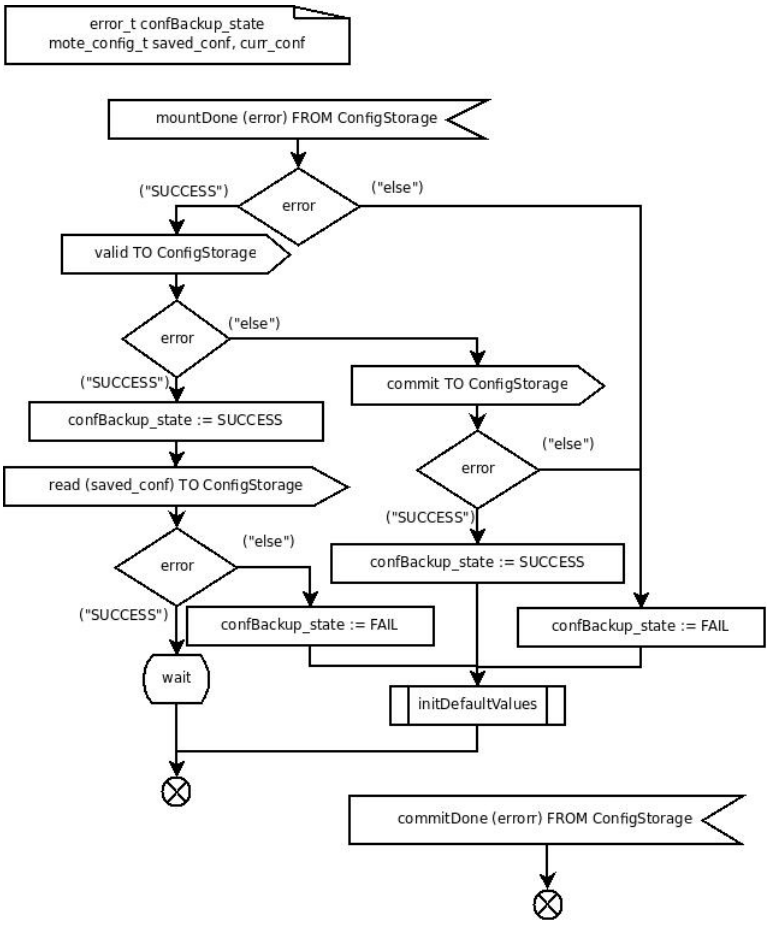
www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MTS_MDA_Datasheet.pdf

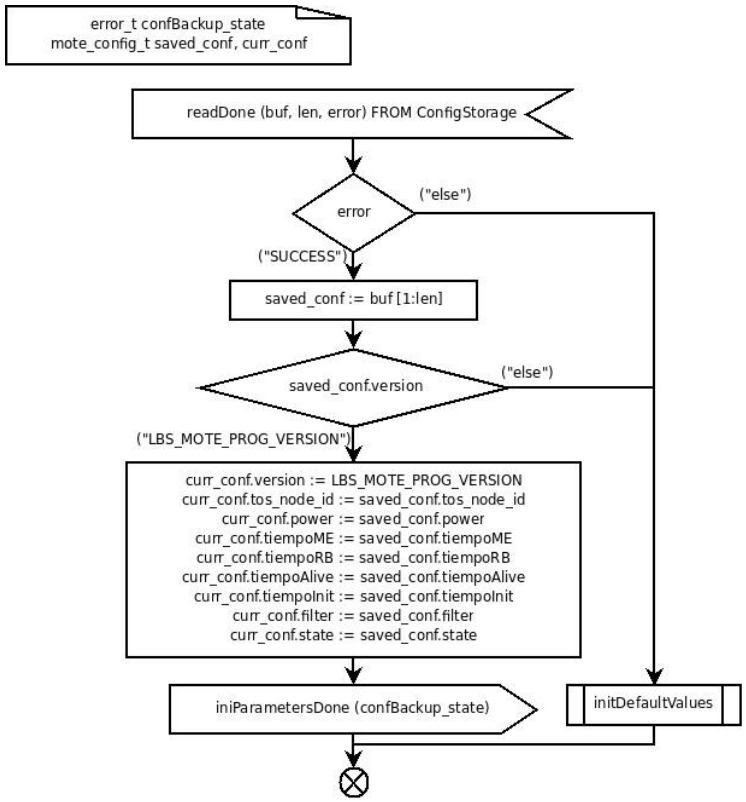
9 ANEXOS

9.1 ANEXO 1: DIAGRAMAS

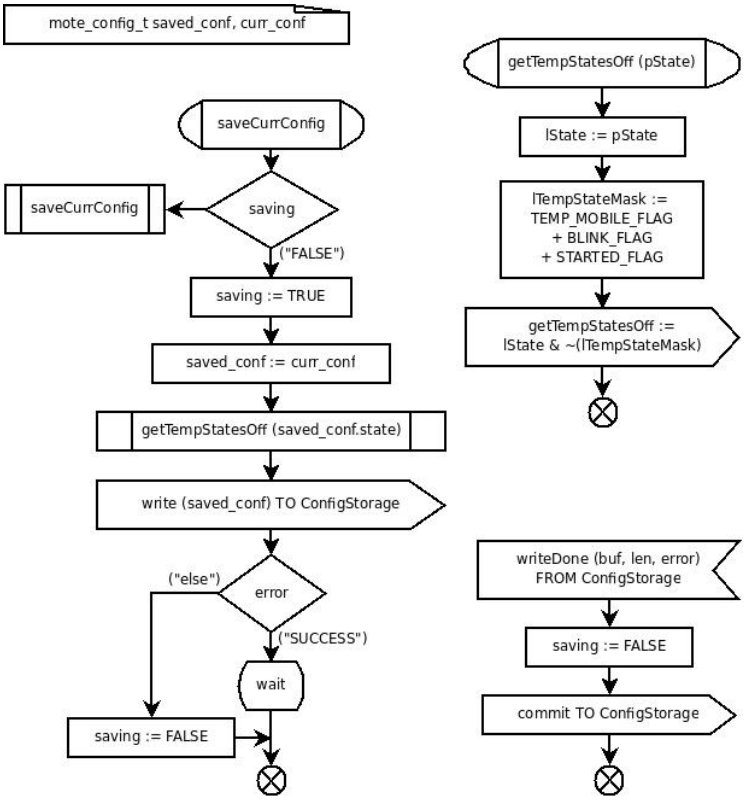


Diagramas del Proceso de Inicialización de Parámetros (*figuras 1 y 2 de 4*)





Diagramas del Proceso de Inicialización de Parámetros (*figuras 3 y 4 de 4*)



Diagramas del Proceso de Almacenamiento No Volátil de la Configuración

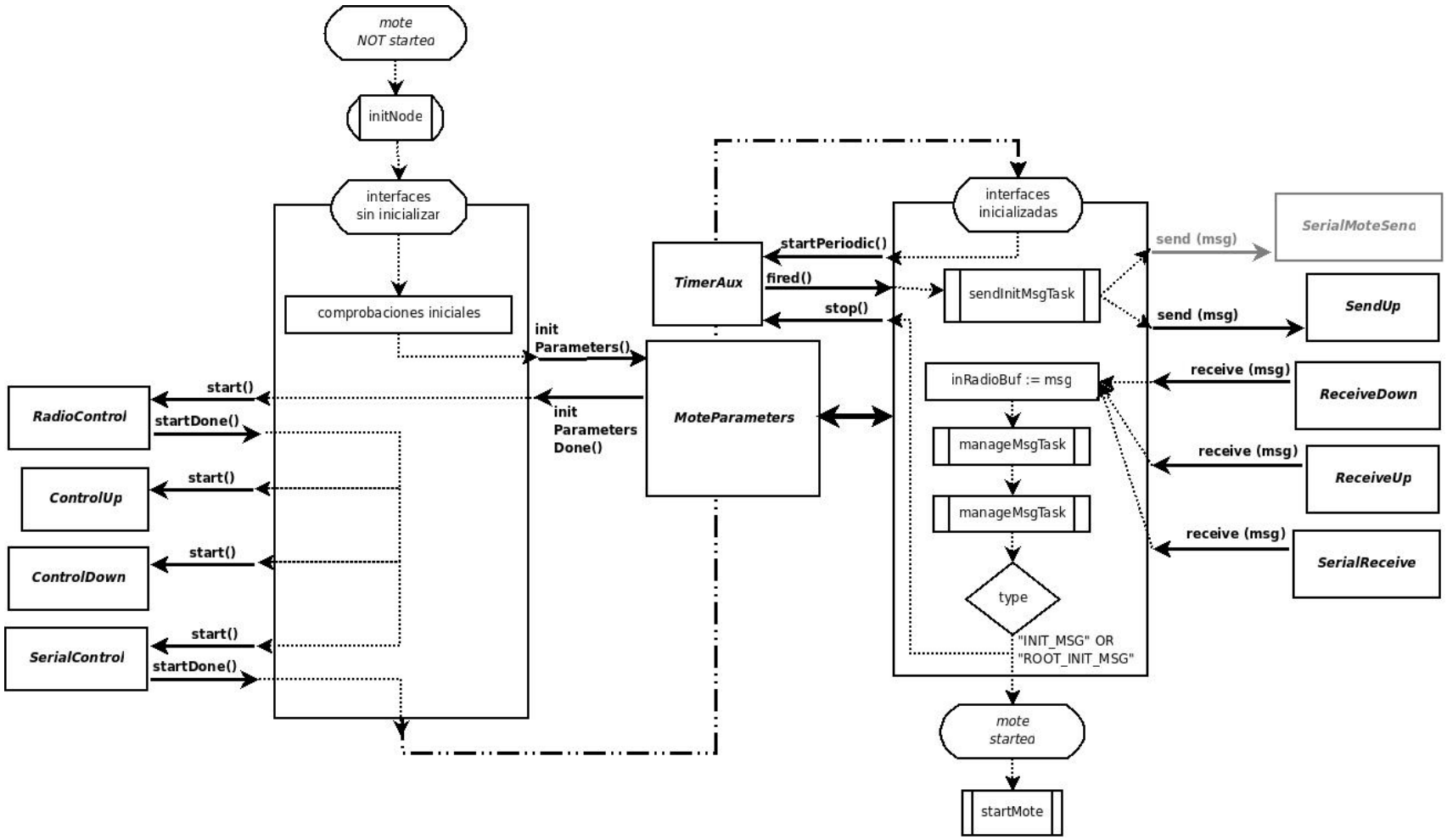


Diagrama del Proceso de Inicialización

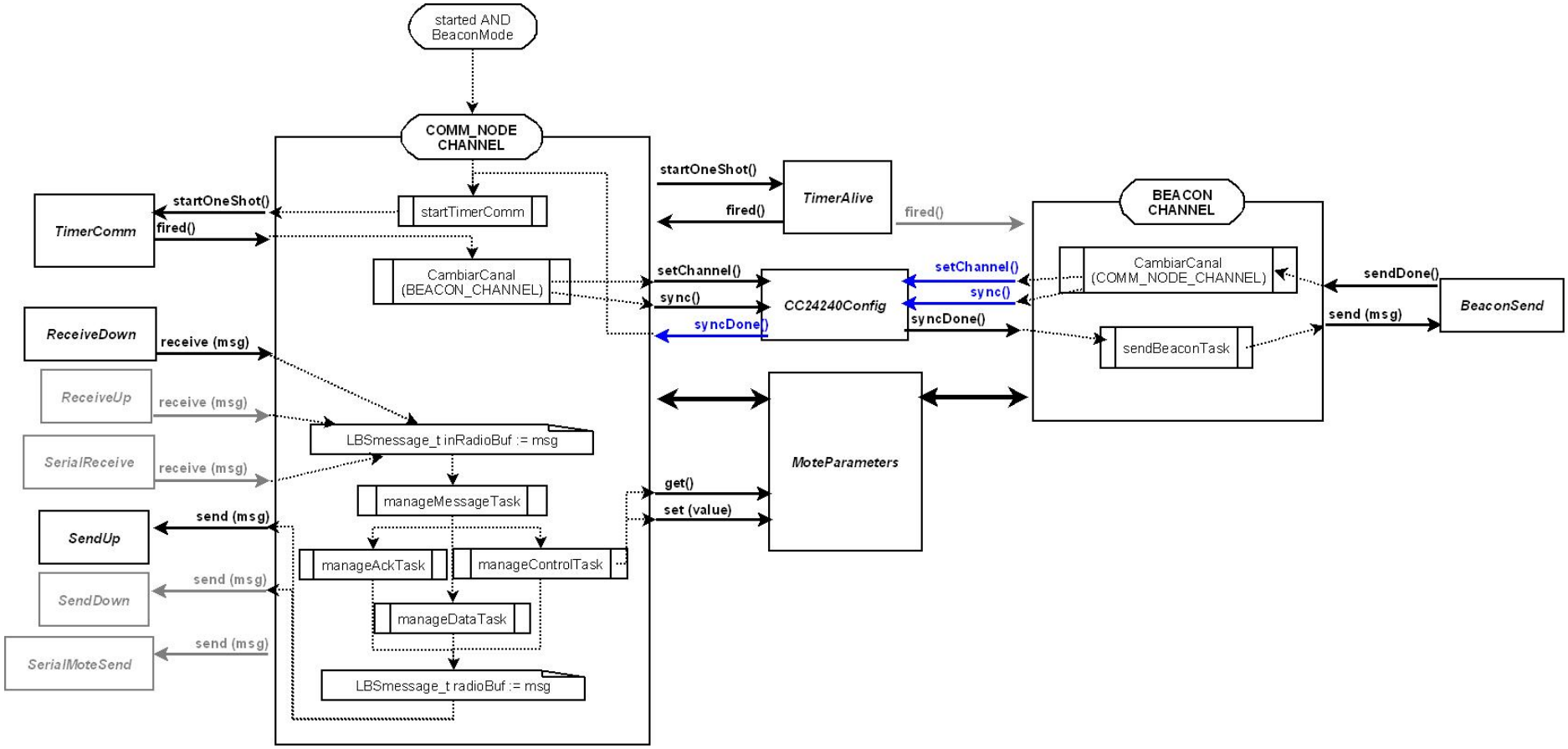


Diagrama del Modo '*Baliza*'

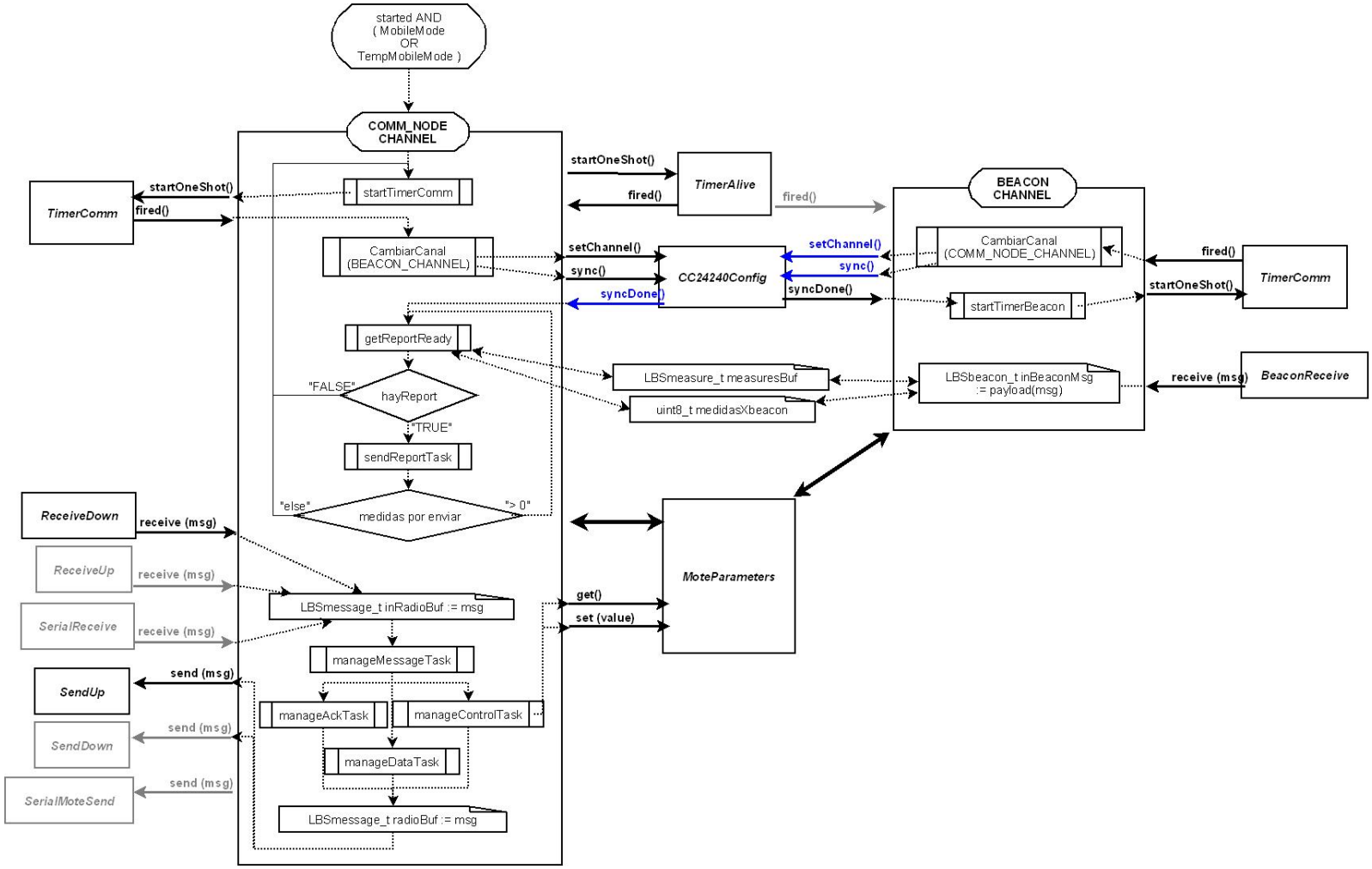


Diagrama del Modo '*Móvil*'

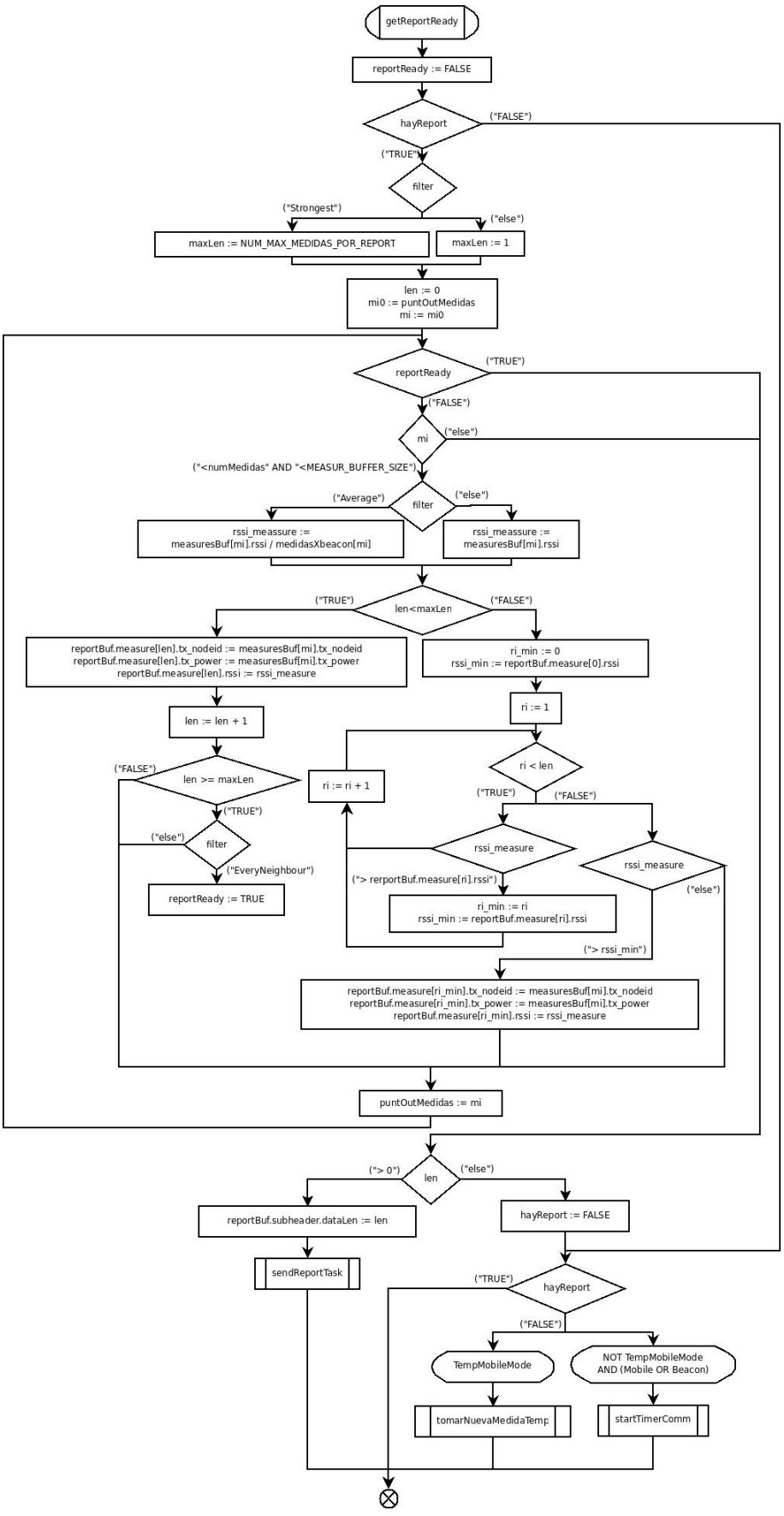


Diagrama del Preprocesado de las Medidas y Preparación del Informe de Localización

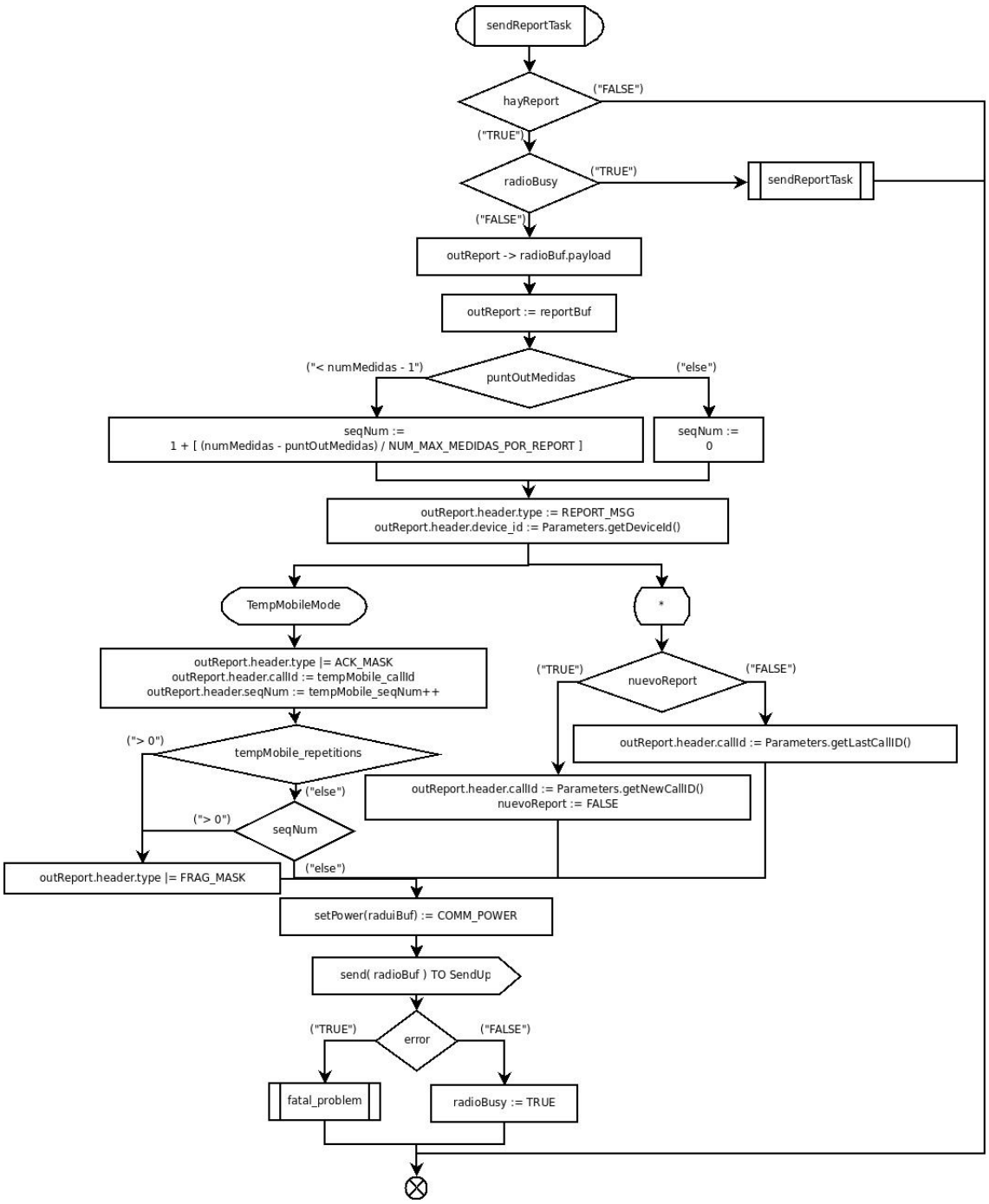


Diagrama del Proceso de Envío de del Informe de Localización

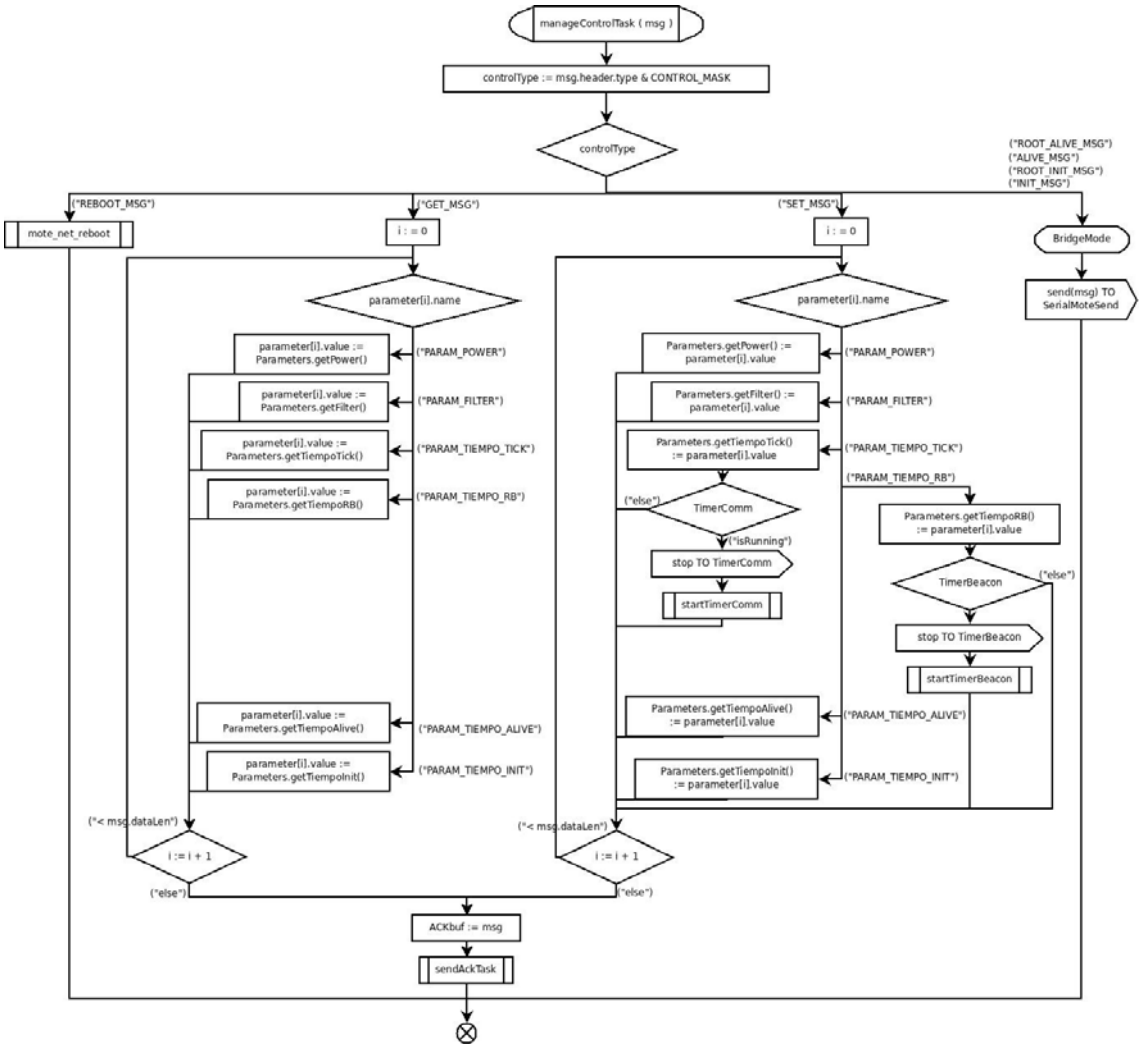


Diagrama de la Gestión de los Mensajes de Control

9.2 ANEXO 2: INTERFACES ICE DEL SERVIDOR DEL ENLACE

9.2.1 BRIDGE SERVER

```
// **********************************************************************
// Copyright (c) 2008 URJC. All rights reserved.
// **********************************************************************

#ifndef CECI_ZIGBBRIDGESERVER_ICE
#define CECI_ZIGBBRIDGESERVER_ICE

#include <ServerProperties.ice>
#include <GeneralManagement.ice>

module es
{
module urjc
{
module ceci
{
module znet
{
module interfaces
{
   module bridgeserver
   {
```

```
/**
 * This exception is raised if the action request to the
 * mote timed out and failed.
 **/
exception RequestTimedOutException
{
    string message;
};
sequence<string> AddressSeq;
module zigbscanner
{
      /**
       * Data structure to contain measured signal strength
       * infomation.
       */
      struct Measure
      {
          string objectID;
          string objectNet;
          int         rssi;
      };
      sequence<Measure> MeasureSeq;
      /**
       * Data structure to contain measured signal strength
       * infomation.
       */
```

```
sequence<double> SensorValueSeq;
struct Sensor
{
    string          objectID;
    string          objectNet;
    long            type;
    SensorValueSeq  val;
};
sequence<Sensor> SensorSeq;

/**
 * The scanner interface allows signal stringth measurments
 * to be taken for a specific mote.
 */
interface Scanner
{
       /**
        * Refreshes measurment list
        */
       bool refresh(string id) throws RequestTimedOutException;
       /**
        * Retreives time of sucessful last measurment
        */
       long getTime();

       /**
```

```
             * Retreives the measurment list
             */
            MeasureSeq getMeasureSeq();
      };
};

/**
 * The Control interface allows message and commands to be sent
 * to a targeted mote.
 */
sequence<string> ParameterTypeSeq;
sequence<string> CommandSeq;
sequence<int> ResultSeq;
sequence<int> ArgSeq;

interface Control
{
      /**
       * Gets a specified parameter
       */
      ResultSeq getParameter(string id, ParameterTypeSeq parameterNames)
            throws RequestTimedOutException;
      /**
       * Sets a specified parameter
       */
      int setParameter(string id, ParameterTypeSeq parameterNames, ArgSeq values)
```

```
                throws RequestTimedOutException;
        /**
         * Exicutes a specified command synchronously
         */
        ResultSeq CommandSync(string id, CommandSeq commands, ArgSeq values)
                throws RequestTimedOutException;
        /**
         * Exicutes a specified command asynchronously
         */
        void CommandAsync(string id, CommandSeq commands, ArgSeq values);
};

interface Management extends ::es::urjc::ceci::management::interfaces::basetools::BasicManagement
{
        // *******************************************************
        // * Model management functions
        // *******************************************************/
        /**
         * Get list of gatway address attached to bridge.
         */
        AddressSeq getGatewayList();
        /**
         * Get list of beacons attached to a gatway.
         */
        AddressSeq getBeaconList(string gateway)
              throws ::es::urjc::ceci::management
```

```
                                ::interfaces::serverProperties::ElementMissingException;
            /**
             * Get list of mobiles attached to a gatway.
             */
            AddressSeq getMobileList(string gateway)
                  throws ::es::urjc::ceci::management
                                ::interfaces::serverProperties::ElementMissingException;
        };
    };
};
};
};
};
};

#endif
```

9.2.2 GENERAL MANAGEMENT

```
// ************************************************************************
// Copyright (c) 2008 URJC. All rights reserved.
// ************************************************************************
#ifndef CECI_GENERALMANAGEMENT_ICE
#define CECI_GENERALMANAGEMENT_ICE

#include <ServerProperties.ice>

module es
{
module urjc
{
module ceci
{
module management
{
module interfaces
{
	module basetools
	{
		interface BasicManagement extends ::es::urjc::ceci::management::interfaces::serverProperties::ServerList
		{
			// *******************************************************
			// * General management functions
```

```
                // ******************************************************/
                /**
                 * Commits the set of changes made to the locally stored repository.
                 * If not called any changes made will only be tempory.
                 */
                void commit();
                /**
                 * Restarts the server. Any changes mad in the managment interface will
                 * have no effect until the server is restarted.
                 *
                 * NOTE: This call cannot be used to restart the server on a different host.
                 * In that case, shut down the server and restart on the requied host.
                 */
                void restart();
                /**
                 * Shuts down the server
                 */
                void shutdown();
            };
        };
};
};
};
};
};
#endif
```