Proyecto: Sistema de incendios inalámbrico

Autor: Sergio Menéndez Muñiz
"I.T.I.S."

Consultor: Jordi Becares Ferres

Fecha de entrega: 31 de Mayo de 2012

Resumen

Este proyecto final de carrera busca la creación de un sistema de detección de indencios (en adelante SDI) basado en pequeños dispositivos inalámbricos de fácil instalación y mínimo mantenimiento.

A su vez, es un sistema totalmente ampliable, ya que estos dispositivos son genéricos y usan un sistema operativo libre y pone a disposición del programador una librería Java para poder programar mediante este lenguaje las aplicaciones de PC que se necesiten para interactuar con él, por lo que es totalmente libre, no hay software propietario.

En este documento se expondrá todo el despliegue de conceptos necesario para la elaboración del SDI, así como su despliegue en el entorno de producción.

También se hará una reseña a cada posible problema que haya surgido durante el desarrollo y la solución adoptada y su por qué.

A continuación se presenta el índice de la memoria.

Índice de contenido

Índice de ilustraciones

Introducción

Justificación

Desde hace unos cuantos años, hemos visto que la tendencia de las instalación informáticas ó de domótica (extendida a más que meramente un *hogar*) va hacia la implantación de sistemas "ligeros", compuestos únicamente por pequeños dispositivos inalámbricos (denominados *motes*), cuyo coste no es tan elevado, son muy versátiles y se pueden fabricar para un fin concreto ó para varios fines a la vez (se puede fabricar una *mote* con un sensor de calor, ó una *mote* con el mismo sensor y varios más, como es el caso de las que se usan en este proyecto).

Así pues, este proyecto busca extender el uso de estos dispositivos al desarrollo de un sistema de detección de incendios, usando *motes* con sensores de luz y calor (tienen varios más, pero éstos son los que nos atañen al proyecto en sí).

Hoy en día es puede conseguir un producto totalmente competitivo debido a dicho bajo coste y a su versatilidad. Además, el soporte informático necesario es reutilizable entre distintos sistemas (un sistema de incendios en este caso, y un sistema de detección de instrusos, por ejemplo, en otro).

Como empresa de IT, ofrecer un sistema basado en motes es un punto muy fuerte, ya que colocaría a la empresa en el nivel más alto de tecnología, siendo las motes la tendencia y el futuro.
Sacar todo el partida de estas nuevas tecnologías es algo muy positivo en cuanto a diversidad de sistemas que se pueden ofrecer. El SDI sería sólo el primer paso, el punto de entrada al mundo de las motes, que como se explica en el primer párrafo de este apartado, se puede extender a detectar movimiento externo (sensores infrarrojos), movimiento propio (sensores de cinética)... También se puede usar en el sector de seguridad de telecomunicaciones, creando un nuevo sistema de comunicaciones subyacente a los actuales, con lógica para eludir espionajes de la señal; como pequeños sistemas de tunning que conectados al cuadro de mandos del coche puedan controlar puertas, luces, audio...
La versatilidad del uso de las motes es realmente enorme, por eso apostar por ellas es apostar por un futuro seguro, sólo falta acertar con el camino a elegir dentro de ese futuro.

Desde el punto de vista de la organización que adquiere el SDI de una empresa de servicios, la organización recibe un sistema totalmente moderno, versátil y dinámico. Hablamos de un sistema cuya dificultad para conseguir piezas es nula respecto a sistemas basados en dispositivos particulares diseñados por la propia empresa de servicios (ejemplos de sistemas con dispositivos creados para este único fin: www.wirelessfirealarm.com , www.ttsfire.co.uk).

Por tanto, apostar por este sistema es asegurarse una inversión tecnológica que en todo momento podrá adaptarse a ampliaciones, no sólo al actual sistema de detección de incendios, sino a un sistema más grande, del que lo que aquí se ofrece, forme parte.

Descripción

En cuanto a componentes generales, el proyecto consta de tres partes:

- Aplicación de operador -> Aplicación en un PC interactuada por una persona
- Aplicación de la mote conectada al PC del operador
- Aplicación para las motes independientes (dispuestas por el edificio en el que se aplique el

sistema)

En términos funcionales, el proyecto garantiza los siguientes puntos:

- Detección de incendios
- Notificación de alarmas remotamente
- Notificación de alarmas localmente
- Sistema manual de activación de alarma
- Notificación fiable de alarmas
- Reconocimiento de alarmas por parte del operador
- Monitorización de batería de las *motes*
- Indicar visualmente el estado de la *mote*
- Protección de caídas de las *motes*
- Consulta de los datos de sensores de cada *mote*
- Prueba de cobertura
- Activación / desactivación de una *mote*
- Aplicación de operador simple y de fácil despliegue

Con todos estos puntos cubiertos, obtendremos un sistema totalmente funcional y fiable en todo momento que se ajusta a las necesidades de un sistema de detección de incendios.

Objetivos

Como objetivos, partimos de los puntos indicados en el apartado anterior. Una vez tenemos esa base funcional cubierta para el SDI, añadimos los siguientes con una perspectiva técnica global:

1. Sistema funcional en todo su sentido: conseguir un sistema realmente capaz de detectar de forma eficiente y en tiempo real un incendio.
2. Aplicación de operador intuitiva: se busca que el operador tenga claro qué información está viendo, qué puede hacer con esa información y que pueda adelantarse a eventos.
3. Que la aplicación de las *motes* sea lo más clara posible, permitiendo así la ampliación y su mantenimiento. Esto nos permite poder definir en fases posteriores a la explotación del sistema nuevas funcionalidades, o formar un nuevo sistema por encima de éste al que se adhiera la funcionalidad de lo que en este documento se presenta.
4. Independencia del sistema sobre las motes secundarias: que una mote secundaria entre o salga del sistema ó incluso se quede bloqueada no debe suponer problemas de rendimiento ni para la aplicación de la *mote* conectada al PC de operador ni para la aplicación del operador, y esta caída o adherencia debe ser lo más “transparente” posible para el sistema.
5. Un código de luces claro y fácil de mantener: se busca que las personas que vean el dispositivo tengan claro qué está pasando en base a las luces activas y la forma en que parpadeen.
6. Comunicaciones sencillas: determinar el menor número posible de mensajes y que todos ellos tengan estructuras lo más parecidas y pequeñas posible, así conseguimos un sistema de bajo consumo y baja saturación de ancho de banda / capacidad de procesamiento necesaria.

Enfoque

El SDI, teniendo en cuenta que se basa en dispositivos independientes y una aplicación central, se plantea como una aplicación cliente-servidor común. Esto significa, en nuestro caso, que la aplicación del operador será quien aúne la información de todas las motes, y quien guardará y suministrará la configuración de trabajo para ellas.

En cuanto al código de las aplicaciones, el sistema se divide en dos puntos:

- Aplicación de las *motes*: será una aplicación **modular**, en la que se divide cada funcionalidad en un módulo, haciéndola sostenible y extensible. Cada *mote* enviará los mensajes necesarios para la adhesión / desprendimiento del sistema e información de su estado (datos de sensores y estado de alarma local) y recibirá la configuración y estado de alarma general de la aplicación central. Se desarrolla en lenguaje NesC, propio de estos dispositivos corriendo bajo el sistema operativo TinyOS.
- Aplicación del operador: será una aplicación de gestión común, con módulo funcional para la comunicación con las *motes*. Se desarrolla en lenguaje Java, haciendo uso de la DLL provista por el SO TinyOS para el lenguaje Java.

En ambos casos, hablamos de software libre, lo cual reduce los costes del desarrollo del sistema, que repercuten en el cliente indirectamente.

Se deja la puerta abierta a desarrollos expresos que puedan requerir lógica avanzada en el tratamiento de datos que excedan el sentido común del operador que los ve en pantalla. Podría ser el caso de querer detectar aumentos rápidos de temperatura en un nodo concreto, ó aumentos en varios que sepamos se encuentran próximos, lo cual podría inducir que tenemos una zona con posible riesgo de incendio ó con un incendio activo pero aún oculto.
También se puede pensar en la necesidad de publicar en la red vía web site la información de los nodos, de forma que desde cualquier PC de la organización se puedan consultar estos datos, ó se envíen para su tratamiento a otros centros de procesamiento de datos.

La versatilidad que nos da este sistema, es muy grande, pero de momento nos hemos centrado sólo en los puntos citados arriba, aunque siempre pensando en poder extenderlo si se necesitase.

Planificación

Planificación de desarrollo

Las tareas para el desarrollo son las siguientes:

Nombre	Fecha de inicio	Fecha de fin
Base Funcional	19/03/12	24/04/12
Comunicación PC-BaseStation	19/03/12	24/03/12
Lectura de sensores	22/03/12	30/03/12
Botón usuario y sensor hall	29/03/12	4/04/12
Monitorizar Batería	2/04/12	4/04/12
BaseStation-Mote remota 1/2	4/04/12	24/04/12
Versión funcional completa	24/04/12	22/05/12
BaseStation-Mote remota 2/2	24/04/12	4/05/12
Aplicación Operador	4/05/12	10/05/12
Interfaz visual LED	10/05/12	22/05/12

Ilustración 1: Planificación de desarrollo del proyecto

El desarrollo de la aplicación se divide en dos grandes bloques:

- Base funcional: Versión funcional del sistema de comunicaciones entre motes externas, mote conectada al PC (BaseStation) y aplicación sencilla de consola con funcionalidad muy limitada de interacción con el operador.
- Versión funcional completa: Versión final de la aplicación que cumple todos los objetivos marcados en el proyecto.

Como se aprecia, el proyecto de desarrollo dura aproximadamente 2 meses. Se puede contemplar una desviación del 20%, aumentando a 2 meses y medio. La documentación se realiza a la par que el desarrollo, por lo que cada parte incluye el tiempo dedicado a la documentación.

Planificación de la puesta en producción

La puesta en marcha no supone más que 1 día. Comprando el producto, se adquiere automáticamente un PC de operador preparado y 5 motes ya configuradas. Al ser comunicación inalámbrica, no se requiere obra alguna, por lo que sólo hace falta que el cliente indique dónde quiere colocar cada mote y el PC de operador, y listo.

El tiempo que sobrepase ese único día de puesta en marcha se deberá únicamente a las exigencias del cliente en cuanto a disposición físicas de las motes que requiera pruebas de covertura en caso de que el entorno destino presente problemas a la hora de dejar pasar las ondas de radiofrecuencia, por lo que esto es indeterminado.

Recursos Empleados

Los recursos necesarios para llevar a cabo el proyecto son los siguientes:

- Entorno de desarrollo Java Eclipse
- Árbol de código fuente de TinyOS 2.0
- Árbol de código fuente específico para mote Cou24 (el modelo concreto de hardware que usamos en este proyecto)
- 2 *motes* Cou 24
- PC bajo Linux Ubuntu 9
- Documentación oficial de TinyOS
- Site Wiki de la asignatura

Productos obtenidos

Con este proyecto, obtendremos un SDI firme para cualquier infraestructura. Tendremos tantas *motes* como necesite la infraestructura destino, que serán dispositivos inalámbricos de fácil instalación y cuyo mantenimiento es meramente cambiar las baterías cuando estén bajas, lo cual se indicará con la antelación suficiente tanto visual como informáticamente.
A parte, tendremos una aplicación de PC para que un operador pueda administrar el sistema.

Las *motes* se entregarán programadas y la aplicación de operador será de fácil despliegue, minimizando la responsabilidad que adquiere el receptor del SDI.

Descripción de los contenidos siguientes

En los apartados "antecedentes" y "estado del arte" se encuentra una descripción de la situación de mercado actual y cómo nuestro producto encaja en él de cara a ser un producto de éxito con salidas.

En el apartado "descripción detallada" se ve, en 3 aproximaciones consecutivas y cada una más profunda que la anterior, la lógica del producto desarrollado (qué es el sistema, cómo actúa, cómo interactúan las partes que lo forman y cómo funciona cada una de esas partes).

En el apartado "valoración económica" se indican costes de desarrollo del producto y de venta del producto.

En conclusiones encontramos la valoración final del producto obtenido por parte del autor así como propuestas de mejora (por un lado unas concretas y por otro puertas abiertas) y autoevaluacón.

En los anexos encontramos cómo preparar el entorno de trabajo para conseguir este producto y el manual de usuario.

Antecedentes

Estado del arte

En los últimos años se ha visto que los sistemas incrustados son bastante eficientes. Hay muchos factores que promueven su crecimiento dentro del mercado, como por ejemplo:

- bajo coste material
- bajo coste en desarrollo
- gran dinamismo en cuanto a funcionalidades ofrecidas
- amplia gama de microcontroladores
- sistema operativo libre
- amplia comunidad desarrollando con ellos, lo cual implica gran cantidad de conocimiento en la red
- soporte listo para comunicaciones inalámbricas bajo los estándares IEEE

Esto nos hace poder ver cualquier sistema de domótica (a cualquier nivel) como un sistema basado en los sistemas incrustados. En este proyecto desarrollamos un sistema de control de incendios, pero una simple búsqueda por internet sobre las *motes* existentes nos hace ver que podemos implementar, por ejemplo, sistemas de seguridad que se apoyen en sensores de movimiento (aunque su precio se dispara debido al sensor de movimiento).
La capacidad de comunicación inalámbrica es un punto más que fuerte, es un pilar fundamental del crecimiento exponencial de las posibilidades. Además, que estos dispositivos sean de bajo consumo, es muy importante, ya que hemos visto en los últimos años cómo teléfonos móviles, tabletas y ordenadores portátiles han tenido grandes problemas de autonomía por culpa del gran consumo de sus módulos de comunicación WiFi.

Estudio de mercado

Actualmente existen sistemas inalámbricos expresamente desarrollados para detección de incendios. Empresas especializadas ofrecen su producto con los Q.C.s y las certificaciones necesarias de la CE (en la justificación del proyecto se indican dos empresas que ofertan distintos productos para la detección de incendios).

La ventaja principal del sistema que en este documento explicamos es que al ser hardware con varias funcionalidades posibles, podemos de forma fácil ampliar la funcionalidad de nuestro sistemas, además de que su coste es bajo. Por tanto, a la hora de buscar posibles clientes, deberemos estudiarlos bien previamente y ver qué más podemos ofrecerles: qué necesidades, qué soluciones podemos plantearles, y cómo podemos integrar esas soluciones de forma que nuestro producto sea algo necesario en su organización.

Otra ventaja es que esta tecnología se actualiza continuamente, y como se comenta en el apartado anterior (al ser software libre), disponer de una comunidad desarrollando software para este hardware, e implementando nuevo hardware, supone un departamento de I+D gratuito.

Esta tecnología (las *motes*) comenzó en 1999, desde entonces han ido continuamente avanzando, siendo grande su variedad en los primeros años y en los últimos 8 se han mejorado las más fuertes de las primeras.
Al ser hardware cuyos esquemas son más o menos conocidos (no todos ellos), existe la posibilidad de diseñar motes específicas que se ajusten a las necesidades del sistema que se tenga que llevar a cabo, de forma que se puede minimizar aún más los costes.

Hoy en día, a pesar de que esta tecnología con ahora 13 años ya está madura, es difícil encontrar productos firmes desarrollados con motes de comercialización masiva. De hecho, búsquedas de sistemas de detección de incendios basados en motes no nos da ninguno, por lo que si existe, no está aún correctamente comercializado. Esto da un punto de entrada en el mercado muy fuerte, ya que venderíamos tecnología punta a bajo coste, pudiendo destinar parte de la inversión en el proyecto al marketing del producto.

Descripción funcional

El sistema, como se comenta en puntos anteriores, es un sistema modular desde el punto de vista del programador. Esto nos da gran capacidad de resolución de problemas, aplicación de mejoras y reutilización de código para otros proyectos.

La aplicación del operador no es tan puramente modular, porque el lenguaje es java y no se requiere gran abstracción de componentes, ha sido en la aplicación de las *motes* donde sí se ha podido llevar a cabo esto porque por un lado tenemos el sistema operativo que funciona de una forma un tanto modular, y por otro que en sí las *motes* son una agrupación de módulos funcionales que usaremos en conjuntos.

Daremos una aproximación en 3 pasos:

- visión general del sistema
- visión general del funcionamiento de cada parte
- visión en profundidad del funcionamiento de cada parte

(primera aproximación)

Visión general del sistema

Primero veremos un diagrama general de la interacción del sistema con la realidad, teniendo en cuenta la sub interacción entre las 3 partes del sistema:

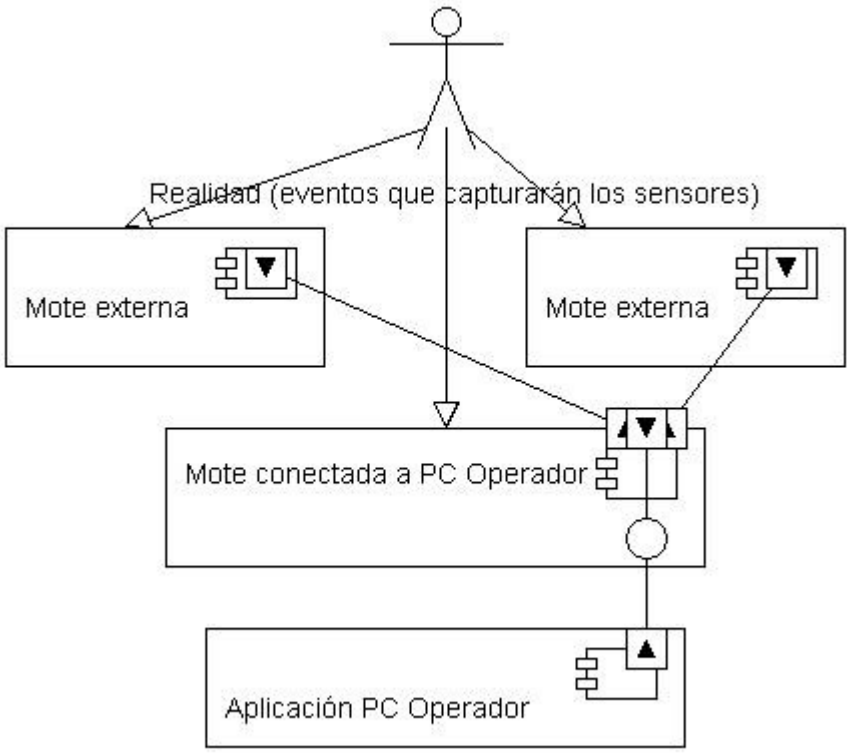


Ilustración 2: Diagrama de interacción entre la realidad que rodea al sistema y éste, indicando cómo interactúan sus 3 partes

Como se ve en la imagen anterior, los eventos de la realidad llegarán a los sensores de las *motes* externas y de la *mote* conectada al PC. Estos eventos serán: luminosidad
La comunicación entre las *motes* externas y la aplicación del PC no es directa, sino que será la *mote* conectada al PC quien reenviará la información en ambas direcciones, sólo esta *mote* tiene comunicación directa, la cual no es por radiofrecuencia sino serial (puerto USB).

- Topología de la red:

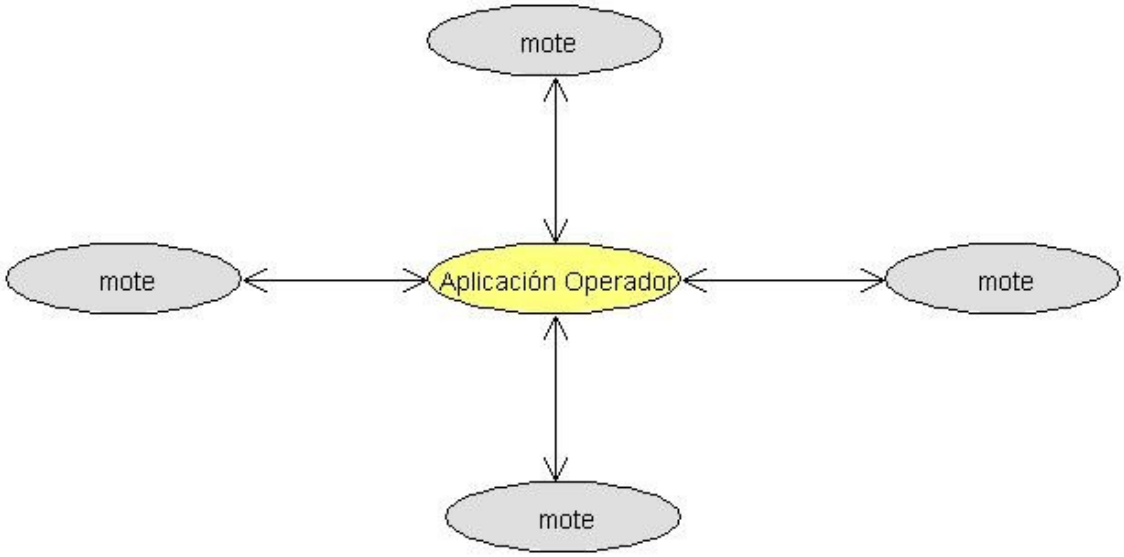


Ilustración 3: Topología de red

El sistema usará una topología de estrella, siendo la aplicación del operador el centro de la red. No tiene sentido otro tipo de red para este sistema (la única alternativa es una comunicación con paso de testigo, pero esto no es del todo viable tan a alto nivel de encapsulación). A continuación se muestra el gráfico indicando la interacción en la red de los componentes. Aunque realmente los paquetes enviados por las *motes* externas son reenviados (pasando de RF a serial) por la mote conectada al PC de operador, este aspecto se ha obviado a la hora de hacer el gráfico ya que a efectos funcionales, es una topología en red tradicional:

(segunda aproximación)

Interacciones entre componentes

A continuación se muestra la interacción entre los 3 componentes (Mote externa, mote conectada a PC y aplicación de operador). Como se ve, la interacción entre la Mote conectada al PC con la aplicación de operador será mucho más intensa que la que mantenga con una *Mote* externa, ya que por ella pasarán todos los paquetes entre los otros dos componentes. Esto hace que la aplicación desarrollada para las motes sea lo más rápida posible, ya que si no se produciría un cuello de botella en la mote conectada al PC.

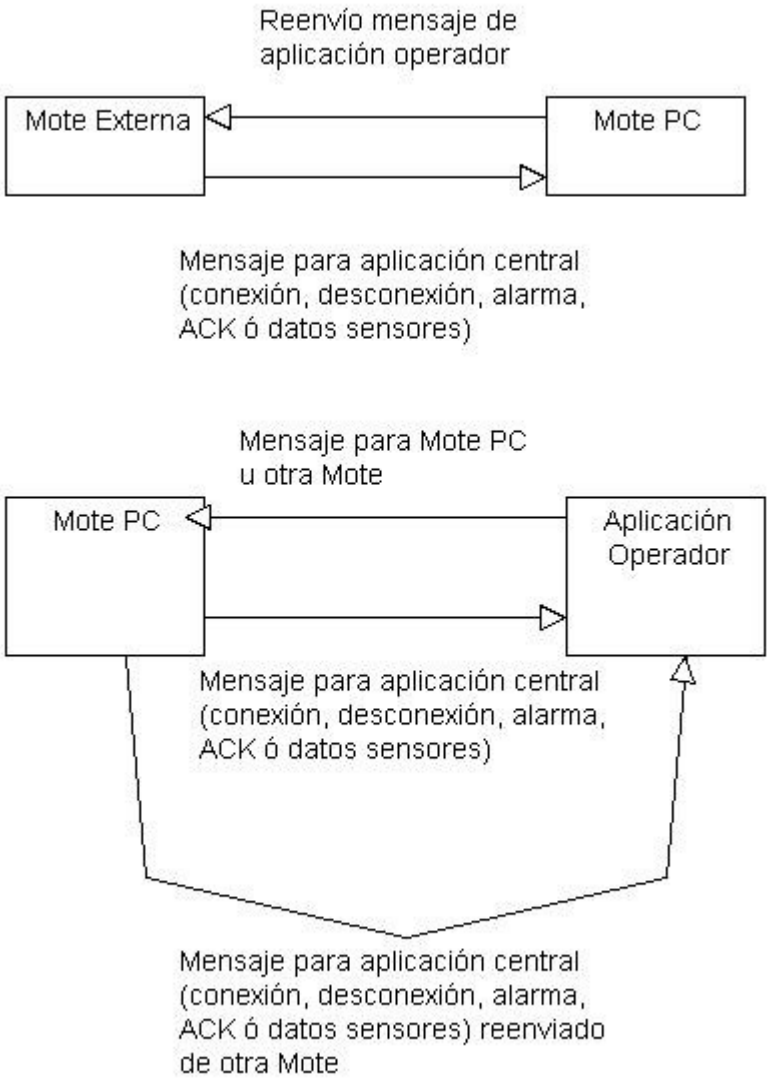


Ilustración 4: Diagrama interacción componentes

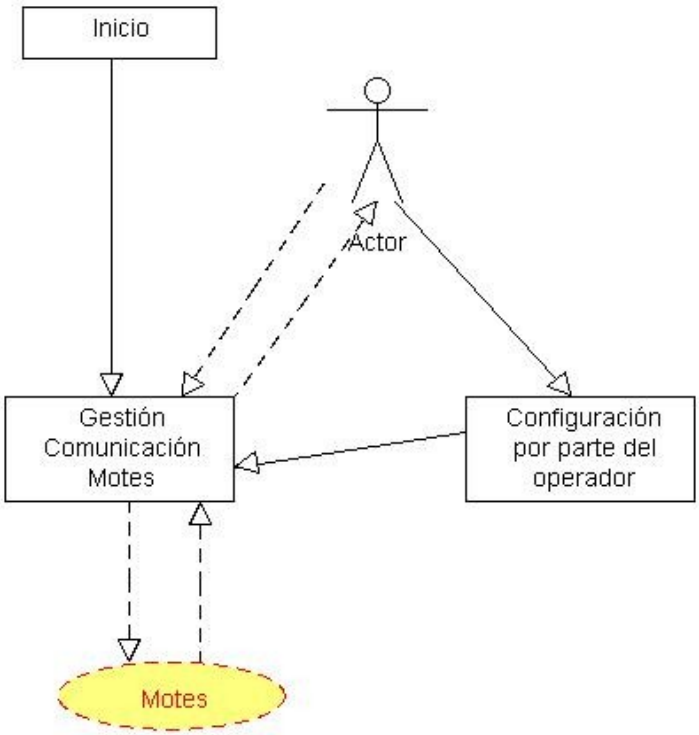
En el gráfico se indica la conexión *mote* externa – PC aunque realmente sea virtual por estar puenteada por la *mote* conectada al PC, pero así queda claro que hay una interacción.

Diagramas generales de las 3 aplicaciones

- Aplicación de operador

  En el siguiente gráfico, se ve el diagrama que nos indica a grandes rasgos cómo funciona la aplicación del operador. Veremos que en un primer momento, la aplicación pide al operador una configuración inicial y espera que le dé estos datos. Posteriormente, pasa al estado de gestión de comunicaciones con las motes, durante el cual se pueden producir interacciones no bloqueantes con el operador.

Ilustración 5: Diagrama de bloques aplicación operador



Vemos en el gráfico, que es una aplicación poco compleja. Como se viene comentando, la aplicación que se ha desarrollado para este proyecto, solamente mostrará los datos recibidos y nos permitirá enviar configuraciones y alarmas a las *motes*.

Si, como se mencionó en puntos anteriores, quisiéramos extender la funcionalidad del sistema (a nivel general), ésta sería, seguramente, la aplicación cuya complejidad aumentaría de forma exponencial, mientras que las de las *motes* simplemente aumentarían linealmente por el desarrollo de nuevos módulos.

– Aplicación de mote externa

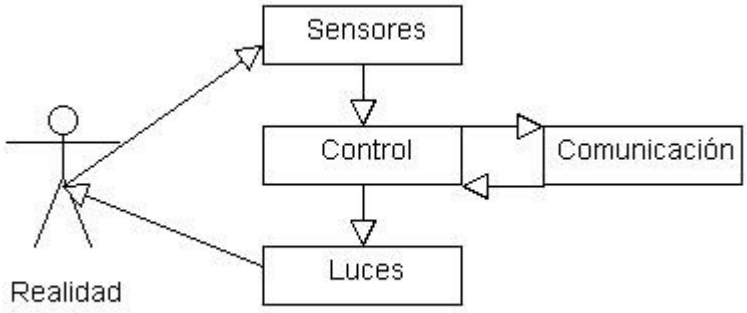


Ilustración 6: Esquema general funcional mote externa

En la imagen anterior, vemos los 4 grandes módulos de la aplicación de la *mote* externa. Éstos son:

- Sensores: el módulo que se encarga de recibir los datos de la tabla de sensores integrados.
- Luces: el módulo encargado de indicar el estado de la *mote* de forma luminosa a través de los 3 LEDs.
- Comunicación: módulo encargado de gestionar la comunicación inalámbrica (envío y recepción).
- Control: el módulo más importante, encargado de gestionar la información recibida por los demás y controlar el estado de la *mote.*

La interacción con la realidad será bidireccional: por un lado la mote recibirá a través de los sensores los datos luminosos y de temperatura, y por otro, plasmará luminosamente su estado.

La comunicación también será bidireccional ya que el módulo de comunicaciones recibirá y enviará a la mote los mensajes de la aplicación de operador así como enviará los datos recogidos y las posibles alarmas.

– Aplicación de mote conectada a PC de operador

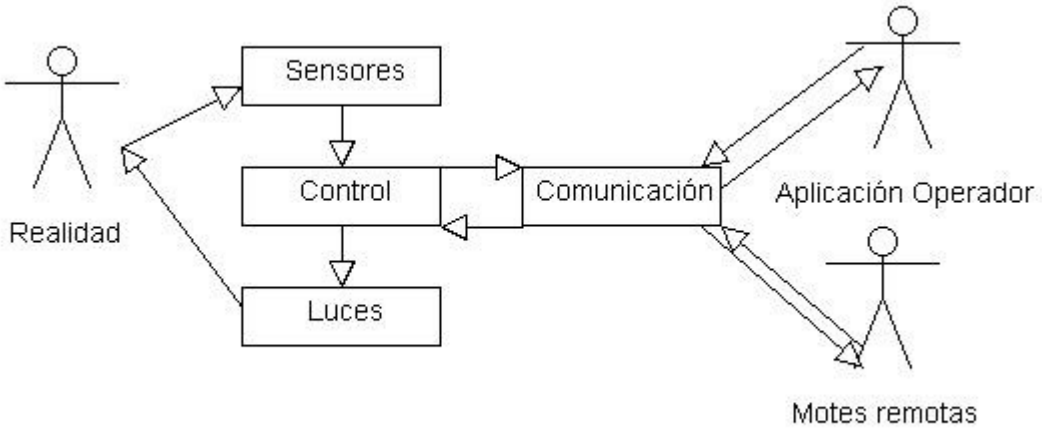


Ilustración 7: Esquema funcional general mote PC

Como se ve, se comparten los 4 grandes módulos. Las diferencias estriban en que el módulo de comunicación gestionará comunicaciones inalámbricas y seriales. Las primeras se encargarán de la recepción y envío de paquetes de datos a las motes remotas; las segundas de la comunicación con la aplicación del operador.

(tercera aproximación)

Descripción Detallada

Descripción de mensajes usados en el sistema

Los mensajes usados en el sistema son mensajes de un alto nivel de abstracción. Esto significa, que las capas subyacentes no se tienen en cuenta, ya que el sistema operativo TinyOS se encarga de gestionar la comunicación.

Por ello, en esta descripción sólo se verán los campos propios de cada mensaje, sin entrar en las cabeceras que realmente tendrían los mensajes de protocolo de comunicaciones TCP de las capas subyacentes, porque para el SDI esto es totalmente transparente:

		longitud (bytes)	
mensaje	**MACMsg**		*mensaje de registro en el sistema*
campos	moteId	1	*identificador de la mote emisora*
	macAddr	2	*dirección física de la mote*
	seqno	2	*número de secuencia de mensaje*

mensaje	**AckMsg**		*Mensaje de confirmación de recepción*
campos	moteId	1	*identificador de la mote emisora*
	tomoteId	1	*mote destino del mensaje*
	seqno	2	*número de secuencia de mensaje*
	NotType	2	*tipo de mensaje que se confirma*

mensaje	**HelloMsg**		*Mensaje de registro en el sistema*
campos	moteId	1	*identificador de la mote emisora*
	tomoteId	1	*mote destino del mensaje*
	seqno	2	*número de secuencia de mensaje*

mensaje	**SensorsMsg**		*Mensaje de muestreo de la mote*
campos	moteId	1	*identificador de la mote emisora*
	countsTemp	2	*valor de sensor de temperatura*
	countsPhoto	2	*valor de sensor de luz*

	countsBat	2	*valor de sensor de batería*
	vrefmilivolts	2	*referencia de voltaje del sistema*
	counter	2	*contador de muestras*
	seqno	2	*número de secuencia de mensaje*

mensaje	**ConfigMsg**		*Mensaje de configuración de mote*
campos	moteId	1	*identificador de la mote emisora*
	tomoteId	1	*mote destino del mensaje*
	TimerStep	2	*tiempo de muestreo de la mote*
	countsTemp	2	*valor máximo de temperatura*
	countsPhoto	2	*valor máximo de luz*
	countsBat	2	*valor mínimo de batería*
	vrefmilivolts	2	*valor mínimo de voltaje*
	seqno	2	*número de secuencia de mensaje*

mensaje	**AlarmMsg**		*Mensaje de notificación de alarma*
campos	moteId	1	*identificador de la mote emisora*
	value	1	*valor de la alarma*
	seqno	2	*número de secuencia de mensaje*
	tomoteId	1	*mote destino del mensaje*

mensaje	**ErrorMsg**		*mensaje de error en mote*
campos	moteId	1	*identificador de la mote emisora*
	value	1	*valor del error*
	seqno	2	*número de secuencia de mensaje*

mensaje	**CovMsg**		*mensaje de prueba de covertura*
campos	moteId	1	*identificador de la mote emisora*
	seqno	2	*número de secuencia de mensaje*
	reintentos	1	*número de reintentos de comunicación*

Aplicación de operador en profundidad

La aplicación de operador, necesita que se haya lanzado previamente la aplicación SerialForwarder, incluída en el árbol de instalación de TinyOS. Esta aplicación está desarrollada en java y será quien pase los paquetes de datos de las motes de la red inalámbrica a la red del ordenador.

Para poder capturar los mensajes que la aplicación SerialForwarder "publica", la aplicación de operador usar los objetos Phoenix, propios del interfaz TinyOS-Java de comunicaciones: la

aplicación será un subscriptor del SerialForwarder.

Se usarán 2 objetos de este tipo, uno para la escucha y otro para la escritura.

A coninuación se detallan las distintas partes, peculiaridades y funcionalidaes de la aplicación para posteriormente mostrar gráficamente los casos de uso y los flujos de la aplicación. Teniendo una visión clara de qué es cada cosa, se podrán entender fácilmente.

- Inicialización de la aplicación de operador

  Una vez instanciados, se activan las escuchas para los distintos tipos de mensajes que hemos definido en la aplicación (se irán viendo en detalle a lo largo de esta tercera aproximación).
  Se activa un temporizador encargado de monitorizar pérdidas de comunicación con las *motes*.
  Se envía mensaje Hello a todas las *motes*. Esto hará que las *motes* vuelvan a estado de inicialización y pidan la configuración.
  Para que esta inicialización funcione, es imprescindible que el SerialForwarder esté activo. Si no lo está, recibiremos una excepción de java.

- Descripción visual de la aplicación:

  La aplicación consta de 2 áreas, 2 menús de ventana y un menú contextual.

  - Área superior: el área superior contiene un listado con las *motes* del sistema, donde se indica número, estado de alarma, hora y tipo y últimos datos de sensores recibidos.
  - Área inferior: este área contiene el cuadro de texto de log de la aplicación. Aquí veremos qué está pasando en el sistema: mensajes de *motes*, mensajes enviados a las *motes* y posibles problemas de comunicación.
  - Menú "Motes": este menú permite interactuar con las *motes*. Nos muestra 3 opciones:
    - Listar Motes: lista las *motes* activas en el sistema, con sus últimos datos de sensores.
    - Alarma general: permite enviar una alarma general a todas las *motes*, pidiendo su valor (1= Alarma, 0= No alarma).
    - Configurar... : este apartado permite enviar una nueva configuración a 1 o todas las *motes*. Tras elegir si a todas o a una sola, pedirá cada parámetro de la nueva configuración.
  - Menú "Ayuda": este menú simplemente muestra en el cuadro de texto del área inferior el texto de ayuda creado para la aplicación (el cual es un fichero de texto que se incluye en la carpeta de la aplicación, que puede ser modificado a gusto).
  - Menú contextual: el único menú contextual existente en la aplicación pertenece al listado de *motes*. Permite un acceso rápido a enviar configuración, obtener últimos datos de los sensores y enviar mensaje Hello, que reiniciará el estado de la mote.

- Mensaje recibido: Mensaje MacMsg:

  Este mensaje es enviado por la *mote* cuando está en estado MS_INI_INI, es decir, en estado de inicialización. Contiene solamente el identificador numérico de la *mote* (que corresponde

al ID que se especifica durante la compilación y subida de código a la *mote*) y su dirección MAC.

- Mensaje recibido: Mensaje ConfigMsg:

  La *mote* envía un mensaje de configuración vacío a la aplicación de operador una vez se ha inicializado correctamente (capas de comunicación y sensores). La aplicación de operador contestará con un mensaje de configuración rellenado.

- Mensaje recibido: Mensaje CovMsg:

  Durante el modo de prueba de covertura de la *mote*, ésta enviará periódicamente a la aplicación de operador un mensaje de este tipo, en el que se indica solamente su ID, el número de secuencia de paquete y las veces que ha tenido que reenviar dicho paquete. Este último campo es muy importante, ya que es el indicador de mensajes perdidos. Nuestra aplicación contestará con un mensaje de reconocimiento (AckMsg, que más adelante se detalla).

- Mensaje recibido: Mensaje SensorsMsg:

  Este es el mensaje más importante junto con el de alarmas (más adelante). Lo envía la *mote* periódicamente, indicando su ID de *mote* y los datos recogidos por los sensores (luz, temperatura, cantidad de batería restante y voltaje).
  La recepción de este mensaje significa que la *mote* NO está en estado de alarma local. La explicación es la siguiente: cuando la *mote* entre en estado de alarma, será por haber sobrepasado (ya sea superior o inferiormente, según el tipo) uno de los valores establecidos por configuración para los datos recogidos por sus sensores. En dicho momento, enviará un mensaje de alarma, y el sistema lo recibirá y reconocerá. Desde este momento, no interesa saber los valores de los sensores de la *mote*, ya que se tiene constancia en el sistema de que existe un problema con ella que se debe atender y resolver. Una vez dicho problema se resuelva, saldrá de estado de alarma y enviará de nuevo los valores de sus sensores para continuar monitorizando.

  Tampoco se recibirán durante la prueba de covertura y la inicialización de la *mote* o cuando se haya parado manualmente la detección de sensores (funcionalidad manual pasando un imán 4 veces por el sensor de efecto hall de la *mote*).
  Durante el estado de alarma remota (alarma general enviada desde la aplicación de operador), la *mote* SÍ que enviará los datos de sus sensores, ya que interesa continuar monitorizando su estado y la evolución de sus indicadores.
  Una ventaja es que se reduce el número de mensajes en la red durante la ocurrencia de alarmas.

- Mensaje recibido: Mensaje AlarmMsg

  Mensaje de alarma. La *mote* lo enviará cuando entre o salga de un estado de alarma. Se indica id de *mote*, estado de la alarma (si existe o no alarma), tipo de la alarma (temperatura, luz, manual ó batería) y número de secuencia del mensaje. La aplicación de operador contestará con un mensaje de reconocimiento y alertará visualmente, en las dos áreas de la ventana, de este evento.

- Mensaje recibido: Mensaje ErrorMsg

  Este mensaje tiene 2 finalidades. Por un lado, avisar de que la *mote* ha entrado en modo de no lectura de sensores (por pase de imán junto al sensor de efecto hall). Por otro lado, informa de un error que se haya producido en el dispositivo. Se contemplan los siguientes errores:

  - Tabla de sensores: se ha producido un error con el interfaz de lectura de los sensores.
  - Cambio de estado sin envío de mensaje: se ha producido un cambio de estado en la *mote*, pero no se ha enviado ningún mensaje a la aplicación de operador para notificar este cambio. Este es un suceso que, si se repite, provocará el reinicio de la *mote* mediante la funcionalidad de guardián WatchDog.
  - Sin lectura de sensores: por algún motivo desconocido, no se han producido lecturas en los sensores. Puede ser indicio de futuros problemas en la tabla de sensores ó de los temporizadores.

- Mensaje recibido: Mensaje AckMsg:

  Este mensaje es el mensaje de confirmación de recepción. En la aplicación de operador es menos importante que en las *motes*, aquí sólo es relevante para controlar las motes que han recibido la alarma general. De todos modos, las *motes* lo envían cuando reciben una nueva configuración, y se muestra en el área inferior de la ventana, por lo que se puede saber si la *mote* ha recibido la configuración nueva.

- Reconocimiento manual de una alarma:

  Las alarmas recibidas de las *motes*, aparecerán notificadas en el área superior dejando la línea perteneciente a su mote con el texto en color rojo además de indicando qué tipo de alarma se ha producido y la hora. Si el operador desea dejar constancia de que hay tratado la alarma aunque aún no se haya recuperado, puede hacerlo mediante el menú contextual, con la opción "Reconocer alarma". Esto dejará el texto con el color normal y además enviará dicho reconocimiento a la *mote*, la cual lo hará patente en su representación luminosa de su estado.

Diagramas de flujo de Aplicación de operador

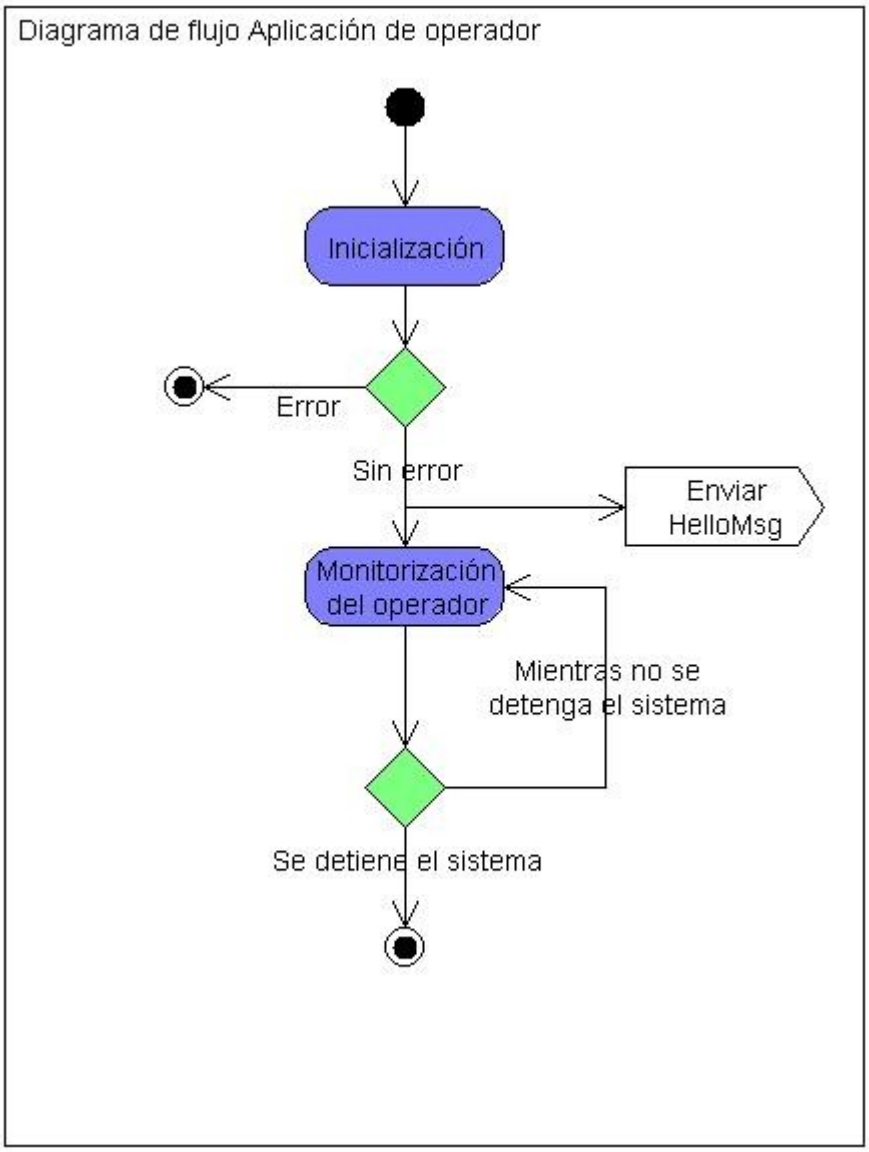


Ilustración 8: Aplicación de operador, diagrama de flujo 1

En este diagrama de flujo, se ve la parte de arranque de la aplicación y de forma general la operación humana sobre la aplicación, es decir, meramente la monitorización del operador.

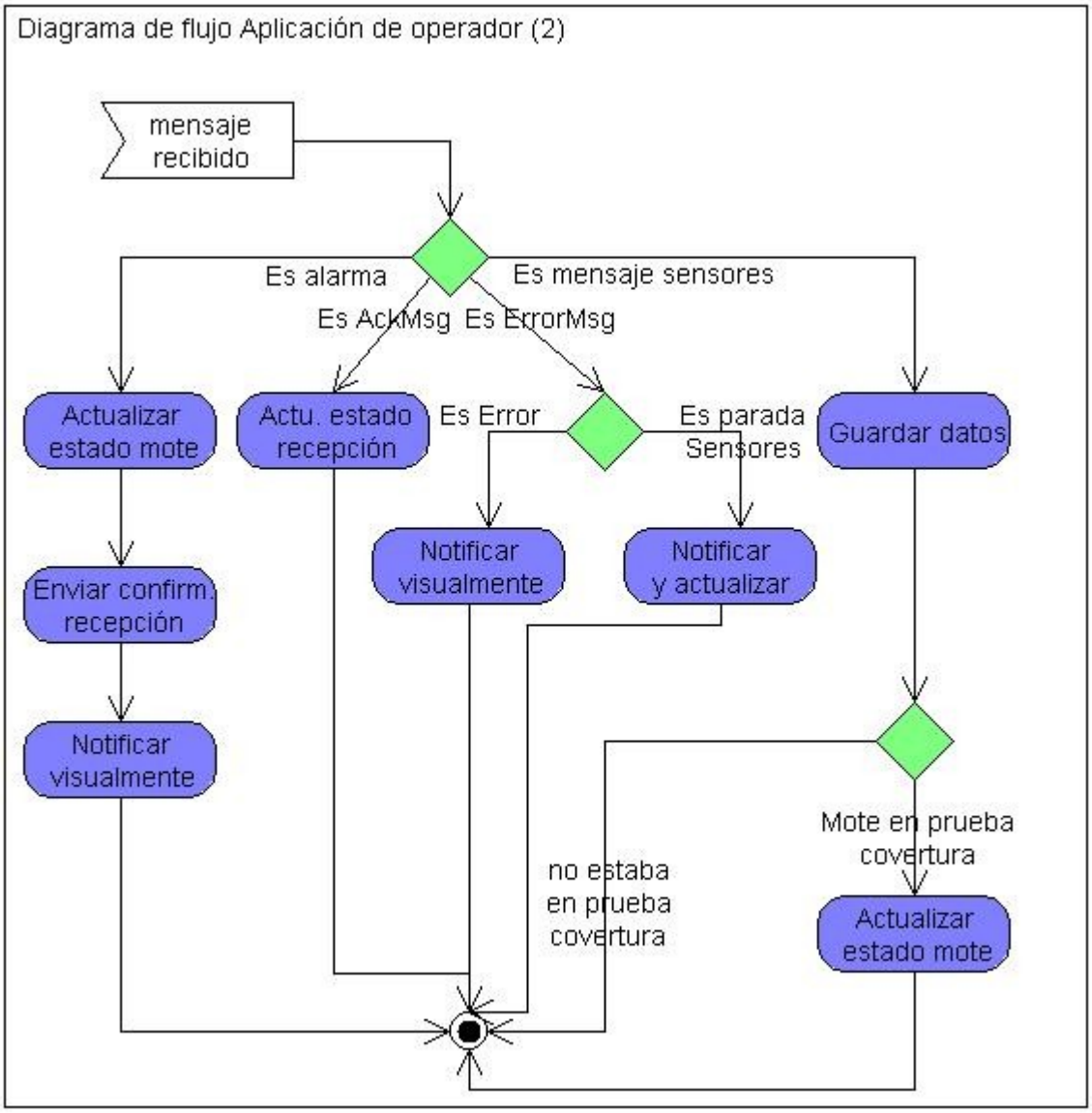


Ilustración 9: Aplicación de operador, diagrama de flujo 2

En el segundo diagrama de flujo, se muestra qué sucede en la aplicación de forma secuencial al recibir uno mensaje. Para que sea claro, se han dejado unos tipos de mensaje para el siguiente gráfico.

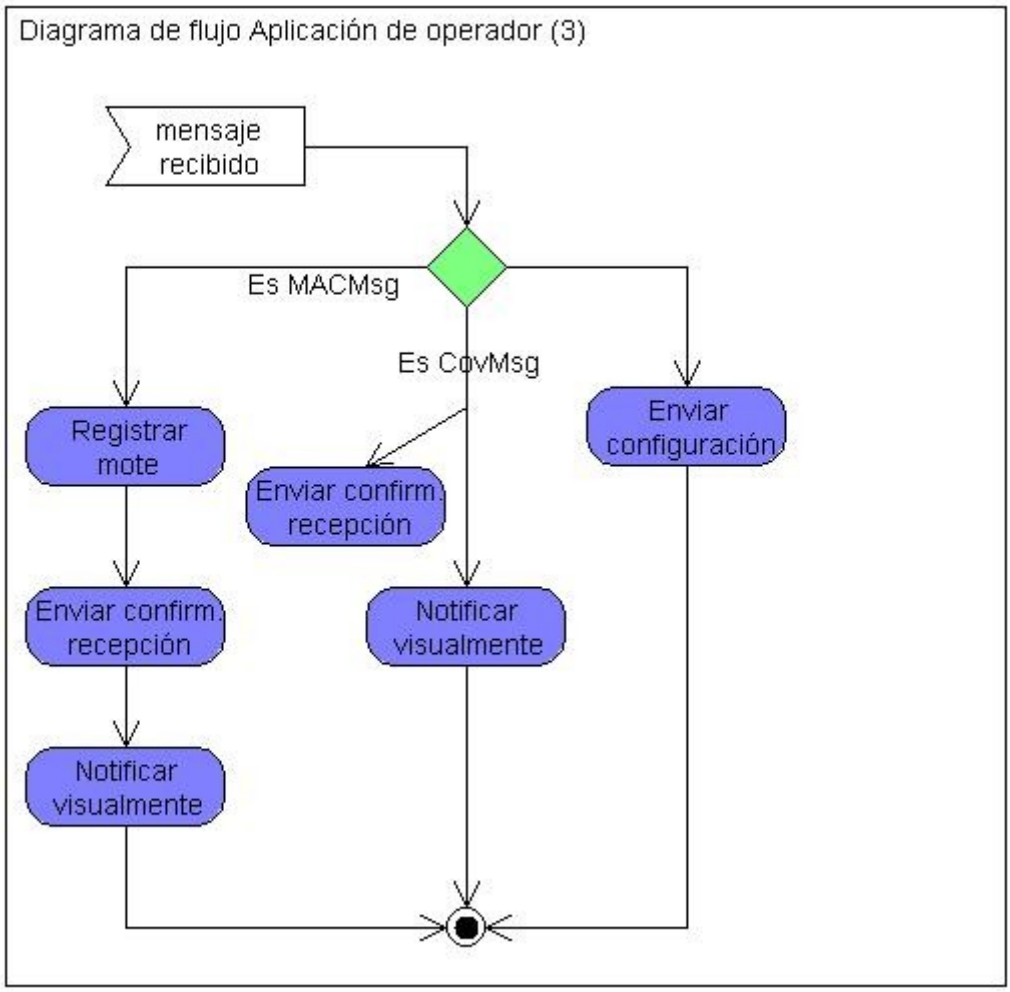


Ilustración 10: Aplicación de operador, diagrama de flujo 3

En este último gráfico sobre el flujo de la aplicación de operador, es muestran los otros tres mensajes que podría recibir la aplicación.

En los 3 gráficos, se omiten pequeños detalles que son complementados con los diagramas de casos de uso.

Casos de uso de aplicación de operador:

Caso de uso #1:

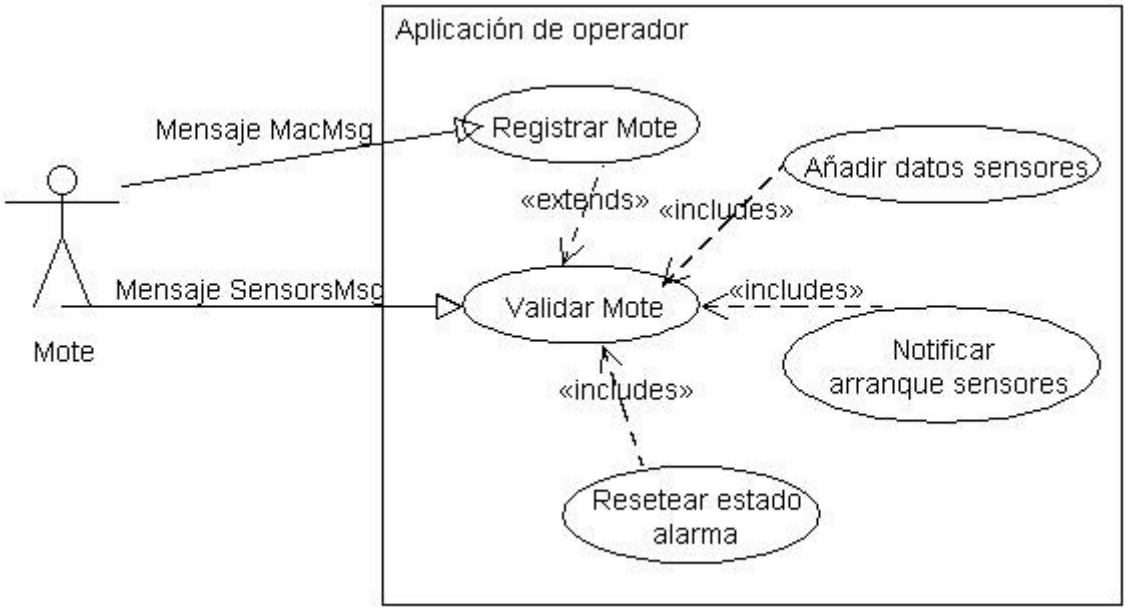


Ilustración 11: Aplicación operador, caso de uso 1: MacMsg y SensorsMsg

En este primer caso de uso se describe la recepción de 2 mensajes:

- MacMsg: En la recepción del mensaje de registro de la mote, la aplicación de operador registra la mote en el sistema, sobrescribiendo cualquier valor anterior que tuviera (siendo el identificador el ID de mote enviado en el mensaje).
- SensorsMsg: En la recepción de este mensaje, la aplicación guardará los valores recibidos de los sensores y restablecerá el estado de alarma local de la mote, poniéndolo a "no alarma", siempre se establece a dicho valor porque como se explicó en la descripción de los mensajes, el hecho de recibir un mensaje de sensor, significa que la mote NO está en estado de alarma local; además, no evaluando el estado anterior, evitamos una comprobación que a efectos de código, es superflua.
  Además, la aplicación decide si acceder a los siguientes métodos:
  - Notificar arranque sensores: como se explicaba en puntos anteriores, el hecho de recibir un mensaje de datos de sensor, significa, entre otras cosas, que la mote NO está en estado de parada de sensores. Por tanto, si en la aplicación hay constancia de que la mote esté parada, se notificará textualmente al operador que esta mote ahora está activa.
  - Registrar mote: en caso de que la aplicación no tenga registrada previamente la mote, se procederá a crear el objeto interno. La contrapatida de recibir mensajes de sensor sin haber recibido antes uno de tipo MacMsg, es que el sistema no podrá registrar la MAC de la mote. No obstante en este sistema no se ha necesitado este dato para nada relevante.

Caso de uso #2:

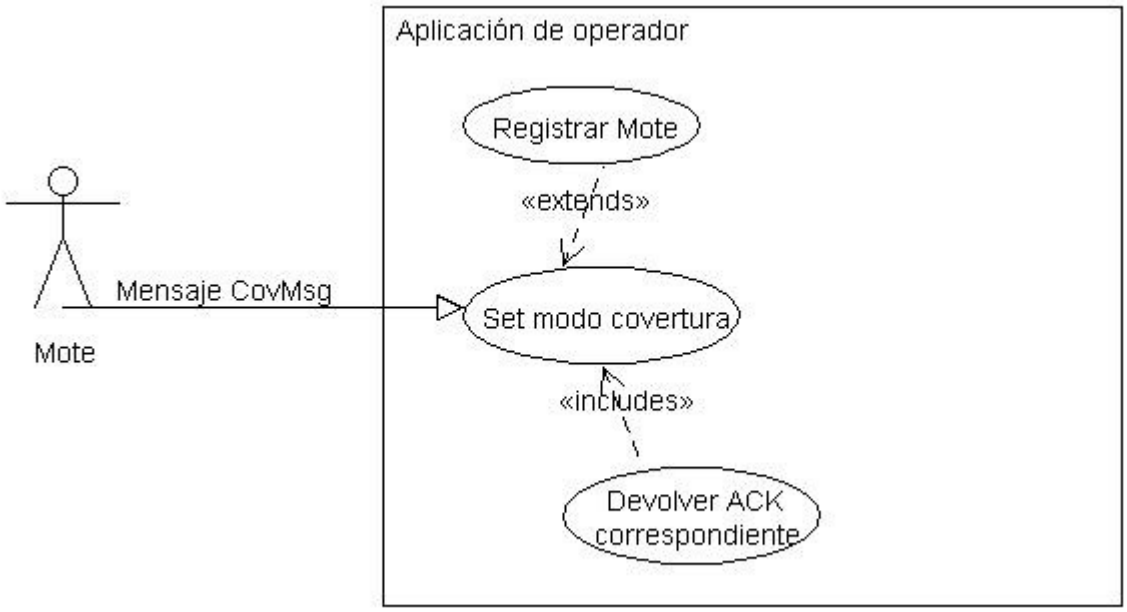


Ilustración 12: Aplicación operador, caso de uso 2: Recepción de CovMsg

En la recepción de un mensaje de covertura, la aplicación de operador registrará la mote, en caso de no estar ya registrada, al igual que se indicó en el caso anterior.

Además, enviará el mensaje de confirmación de recepción a la mote, para que quede constancia de que está dentro de rango de covertura.

Se mostrará en el área inferior de la ventana del operador el evento de recepción de este mensaje así como el número de repetición de mensaje asociado (número de veces que la mote ha reenviado el mensaje hasta recibir la confirmación de recepción).

Caso de uso #3:

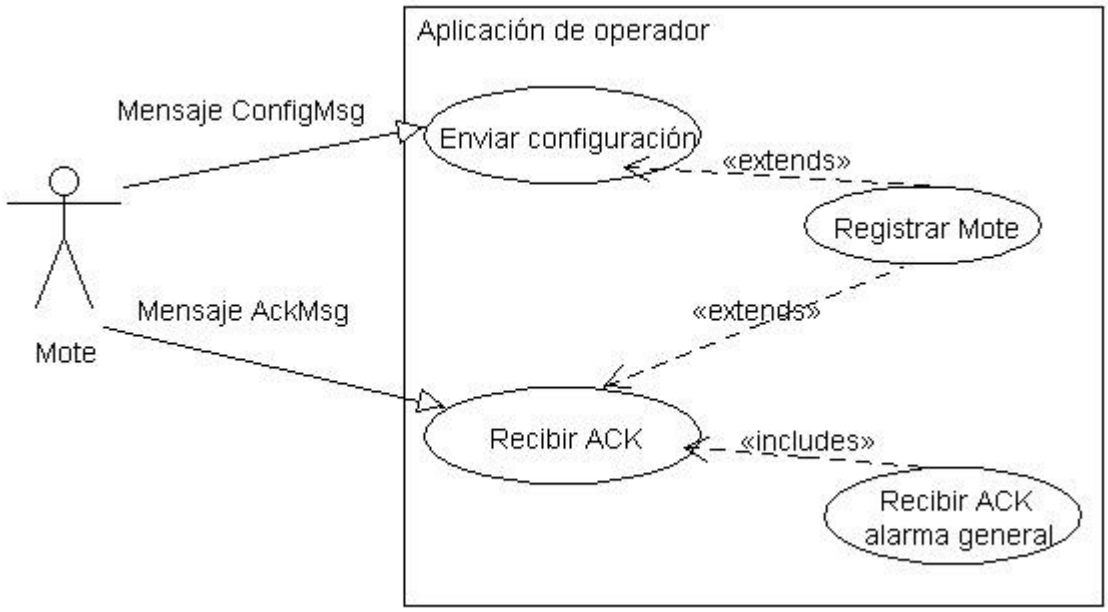


Ilustración 13: Aplicación operador, caso de uso 3: Recepción ConfigMsg y AckMsg

En la recepción del mensaje de configuración, simplemente se contestará con la última configuración establecida para las motes por el operador. En caso de no haberse establecido ninguna, se enviará la configuración por defecto de la aplicación.

En la recepción de un mensaje de tipo ACK, es decir, un mensaje de confirmación de recepción, se evaluará si es una confirmación de un mensaje de alarma general, y si coincide con la última alarma general enviada (se verifica el número de secuencia enviado desde la aplicación de operador y el que el AckMsg recibido confirma), se marca a la mote emisora como que ha recibido la alarma.

En ambos casos, se registra la mote en caso de no haber sido registrada aún.

Caso de uso #4:

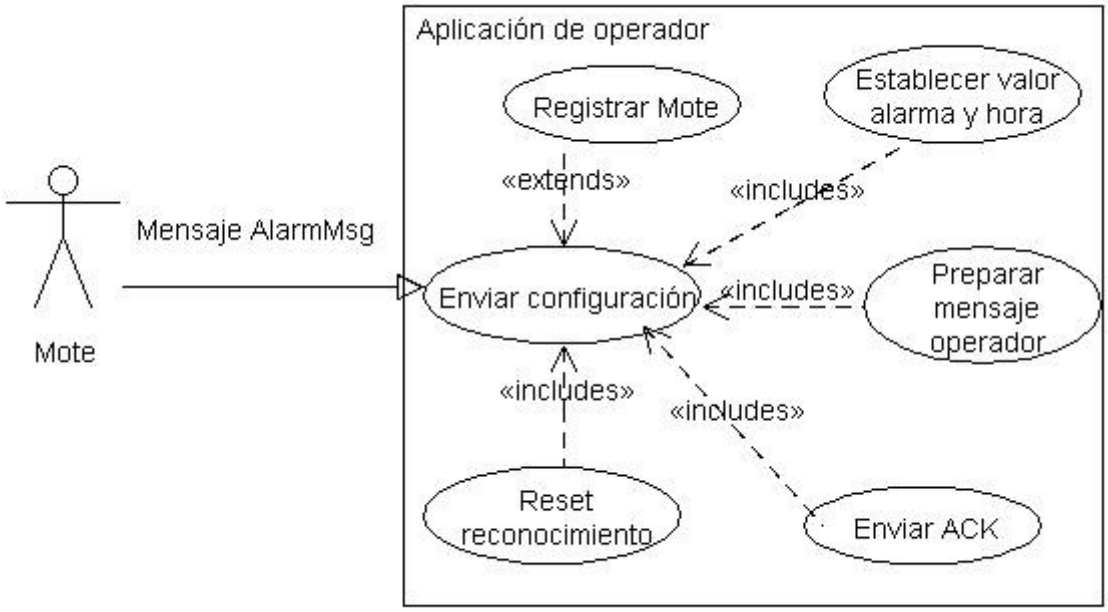


Ilustración 14: Aplicación operador, caso de uso 4: Recepción de AlarmMsg

El caso de uso de la recepción de un mensaje de alarma deja ver que es un poco más complejo.

Por un lado, se registra la mote en caso de no haberse registrado aún como con el resto de mensajes vistos hasta ahora.

Por otro lado, en proceso pinricpal de recepción de alarma se divide en los siguientes:

- Establecer valor de alarma y hora: se establece internamente el valor de alarma local de la mote recibido; también se guarda la hora de recepción.
- Preparar mensaje operador: se prepara un mensaje de texto significativo y relativo al tipo de alarma recibido para mostrar al operador en el área inferior de la ventana.
- Reset reconocimiento: se resetea el estado de "reconocimiento de alarma" sobre la mote emisora del mensaje, de forma que la alarma sea visualmente clara para el operador en el listado de motes del área superior de la ventana.
- Enviar ACK: la aplicación de operador manda automáticamente un mensaje de confirmación de recepción del mensaje. Si no lo hiciera, la mote continuaría notificando su estado de alarma.

Caso de uso #5:

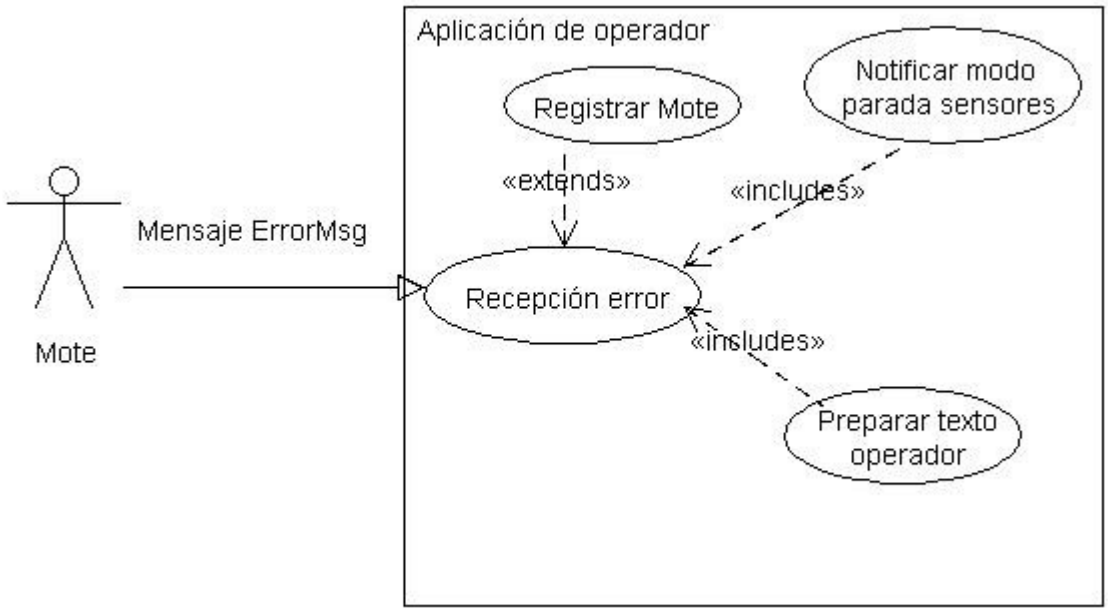


Ilustración 15: Aplicación operador, caso de uso 5: Recepción de ErrorMsg

En la recepción de un mensaje de error, se registra la mote en caso de no estarlo y:

- Se notifica parada de sensores si el valor del error así lo indica. Se notifica textualmente al operador.
- Se preparar un mensaje de texto que explique el error ocurrido de forma clara al operador (en caso de no ser una parada de sensores manual).

Caso de uso #6:

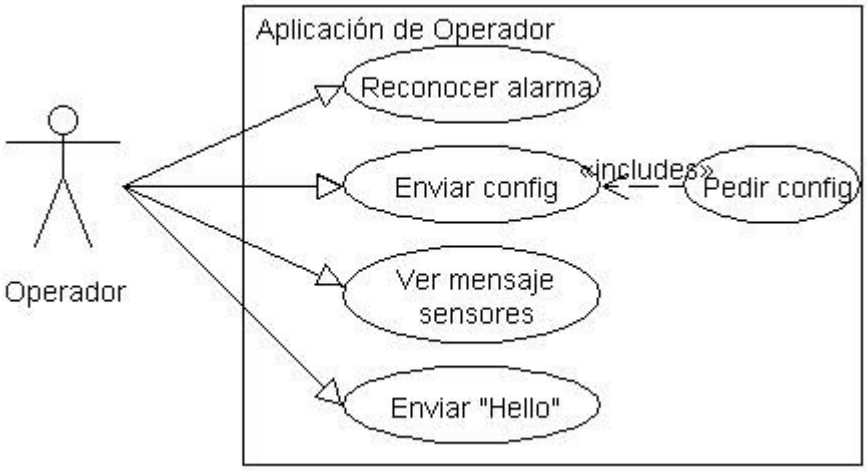


Ilustración 16: Aplicación de operador, caso de uso 6: acciones del menú contextual

En esta ocasión, el gráfico indica la interacción humana por parte del operador con la aplicación mediante el menú contextual del listado de motes (área superior de la ventana de operador). Se obvia que las acciones se realizarán sobre la mote sobre cuya línea hace click el operador.

Se indican los 4 posibles casos:

- Reconocer alarma: El operador quiere marcar una alarma activa como reconocida. Esto marca la alarma del objeto mote interno como reconocida, por lo que el texto de la línea toma su color normal.
- Enviar Config: El operador selecciona una mote a la que enviar una nueva configuración. Una vez hace click, la aplicación le pedirá los distintos valores necesarios para determinar una configuración para la mote. Se muestran los valores por defecto de la aplicación la primera vez, las siguientes se muestra el último valor configurado. Para la temperatura siempre se muestra la temperatura de la mote más 5 grados celsius por defecto. Una vez establecidos todos los valores, se envía un mensaje ConfigMsg a la mote objeto de la acción.
- Ver mensaje sensores: Al seleccionar esta opción, el operador verá en el área inferior de la ventana los datos del último mensaje SensorsMsg recibido de la mote.
- Enviar "Hello": envía un mensaje tipo HelloMsg a la mote, lo cual la reinicializa.

Caso de uso #7:

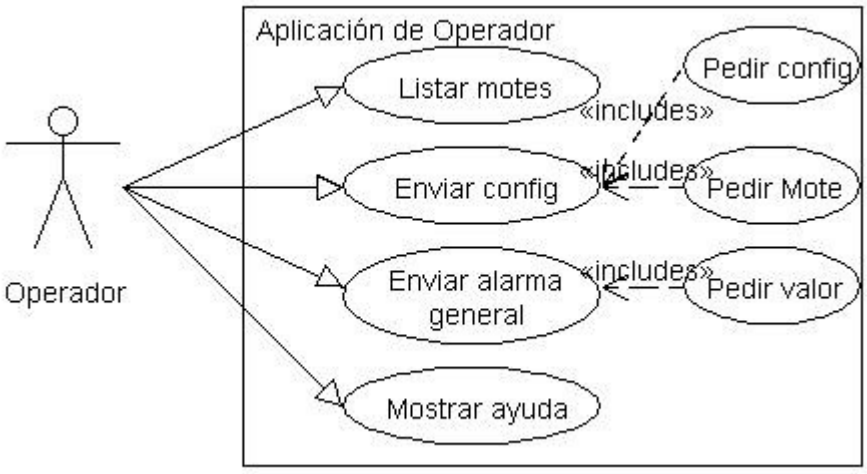


Ilustración 17: Aplicación de operador, caso de uso: menú de ventana

Desde el menú de ventana se exponen 4 opciones:

- Listar motes: Pinta una vista general del estado de las motes identificadas en el sistema y los valores de sus sensores en el área inferior de la ventana del operador.

- Enviar config: Funciona igual que la opción del menú contextual, pero nos permite enviarlo a 1 ó a todas las motes.
- Enviar alarma general: Envía una alarma general a todas las motes, pidiendo antes al operador que introduzca el valor de la alarma.
- Mostrar ayuda: Muestra el texto de ayuda de la aplicación en el área inferior de la ventana de operador.

Aplicación de mote en profundidad

La aplicación de las motes externas y la de la mote conectada al PC del operador es la misma, simplemente evaluando qué mote es la que está ejecutando la aplicación, se determinará el comportamiento. No obstante, el comportamiento no varía prácticamente nada: la mote conectada al PC simplemente puentea los mensajes que debe reenviar y viceversa (reenvía al PC los que recibe de las motes externas).

A continuación se muestra el esquema de componentes que conforman la aplicación de la mote, desarrollada en lenguaje NesC, el propio del sistema operativo TinyOS que usan estos dispositivos:

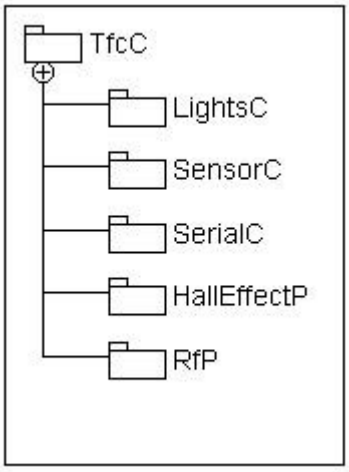


Ilustración 18: Aplicación de mote, componentes

- El componente principal de esta aplicación es TfcC.nc. En él está el punto de arranque de la aplicación (mediante la rutina Boot.booted, evento capturado en este componente y disparado por el sistema operativo TinyOS mediante la interfaz Main)
- El componente LightsC es el encargado de encender y apagar los 3 LEDs que tiene la mote.
- El componente SensorC permite trabajar con los sensores integrados y detectar los pulsados del botón de usuario.
- El componente SerialC y el componente RfP son los que permiten interactuar con las comunicaciones serie e inalámbricas respectivamente.

- El componente HallEffectP permite interactuar con el sensor de hall.

Proceso de arranque en las motes

Es totalmente automático, constando de los siguientes pasos:

- Arranque de temporizador de control general
- Se arranca el sensor de efecto hall (imán)
- Arranque de temporizador de procesamiendo de cola de mensajes inalámbrica
- Arranque de temporizador de procesamiendo de cola de mensajes serial (USB)
- Arranque de temporizador de control de cuelgue de la aplicación

Como se ve, la aplicación se basa en temporizadores, aunque también se atienden eventos como por ejemplo fin de envío ó recepción de un mensaje en una de las interficies.

Todo seguido se detalla qué sucede cuando uno de los temporizadores se dispara, y posteriormente se detallarán los bloques de código ejecutados cuando se produce un evento relevante para la aplicación.

Por último, igual que en el caso de la aplicación de operador, se presentarán los casos de uso de la aplicación para que tras tener una visión conceptual básica de los eventos, se entienda perfectamente la lógica aplicada.

Temporizadores principales de la aplicación de mote

- Arranque de temporizador de control general

  Es el temporizador más importante de la aplicación, en él se decide qué hacer en base al estado de la mote y además se encarga de actualizar las luces para que visualmente se pueda conocer el estado del dispositivo.

  Su lógica depende únicamente del estado de la mote:

  - Estado MS_INI_INI: Se envía un mensaje MACMSg para registrarse en el sistema y se actualizan los LEDs. Se sale de la rutina.
  - Si hay que notificar un cambio en el estado de alarma local, se notifica. Se sale de la rutina.
  - Estado MS_INI_OK: Se envía petición de configuración. Se pasa a estado MS_PENDING_CONFIG.
  - MS_COV_TESTING: Se envía mensaje CovMsg para prueba de covertura.
  - MS_PENDING_CONFIG: Si se ha recibido ya la configuración y no se ha pasado aún a estado MS_RUNNING, se pasa a dicho estado. Si aún no se ha recibido, se vuelve a estado MS_INI_OK.
  - MS_RUNNING: Se envía mensaje de sensor.
  - MS_LOCAL_ALARM: Se consultan los valores de los sensores para saber si se debe

salir del estado de alarma (debido a volver a los límites aceptables por configuración).

- MS_REMOTE_ALARM: Se envía mensaje de sensores y se decide si hay que salir del estado de alarma (por recepción de alarma general no activa) o si se debe pasar a estado de alarma local.

- Arranque del sensor de efecto halla:

  Se hace una llamada al componente HallEffectP, que arranca la captura del sensor.

- Arranque de temporizador de procesamiendo de cola de mensajes inalámbrica: Este temporizador lo que hace es enviar mensajes uno a uno que estén en la cola de salida de comunicación inalámbrica. Hasta que no se ha acabado de enviar un mensaje, no envía el siguiente. Este temporizador tiene sentido en todas las motes (externas y conectada a PC).

- Arranque de temporizador de procesamiendo de cola de mensajes serial (USB): Este temporizador tiene el mismo comportamiento que el indicado antes, pero con la diferencia de que trata los mensajes pendientes de enviar en la cola de comunicación serial. Por tanto, hablamos de paquetes que van dirigidos al PC al que está conectada la mote principal y sólo tiene sentido en dicha mote.

- Arranque de temporizador de control de cuelgue de la aplicación: Este temporizador se encarga de determinar, mediante la funcionalidad WatchDog, si la mote está colgada, y en tal caso, la reinicia. Se da un margen de 2 comprobaciones erróneas antes de reiniciarla. Durante la primera comprobación, la mote llegará a enviar un mensaje de error a la aplicación de operador indicando incoherencia de cambio de estado ó error en número de secuencia de mensajes (localmente). El segundo aviso no se garantiza que llegue ya que si la cola de mensajes tiene un gran número en ese momento, puede que la mote se reinicie antes de enviar el mensaje.

Eventos capturados en el componente TfcC.nc

- iHallC.Fired, Captura del evento de pase de imán: En el componente HallEffectP se ha implementado cierta lógica de captura de evento para que cuando una persona pase por primera vez un imán sobre el sensor, se comience a capturar durante un intervalo fijo de tiempo, el número de veces que pasa el imán. Ese intervalo establecido por código a 4 segundos, es suficiente para que la persona pueda pasarlo 2 ó 4 veces (para activar ó desactivar la captura y envío de datos de sensores). Una vez pasan los 4 segundos, el componente de hall lanza el evento que aquí se captura, indicando el número de veces que se ha pasado el imán. En caso de ser 2 ó 4 veces, se encenderán las luces durante 600 milisegundos y se activará o desactivará la funcionalidad.

- iSensorC.ButtonShortPress, captura de pulsador breve de botón de usuario: Se ha implementado, para confirmación de funcionamiento de la mote y recepción de

alarmas en la aplicación de operador, que cuando el usuario pulsa rápidamente una vez el botón de usuario, se envíe un mensaje de alarma, alternando en cada mensaje el valor de la alarma (activa ó no activa).

- iSensorC.AlarmButton, captura de evento de botón de usuario presionado 2 segundos: La forma establecida en este sistema de detección de incendios para que una persona pueda activar la alarma de una mote es mantener presionado 2 segundos el botón de usuario. Ésta es la rutina del código que lo captura. Desde el componente iSensorC (componente encargado de la interficie con los sensores y el botón de usuario), se envía este evento que el componente general TfcC captura y usa para establecer el nivel interno de alarma de la mote, pasando a pendiente de notificar el nuevo valor de alarma adquirido.

- iSensorC.CoverageMode, captura de evento de entrada en el modo de prueba de covertura: El componente SensorC avisará mediante este evento de la entrada en el modo de prueba de covertura; este evento será a su vez disparado por mantener, una persona, el botón de usuario presionado más de 8 segundos. Para salir, el usuario debe hacer lo mismo. Esta rutina simplemente cambia el estado interno de la mote, y en el siguiente ciclo del temporizador principal, este cambio se tendrá en cuenta.

- TimerAckCallBack.fired, temporizador de espera de confirmación disparado: Este temporizador, no mencionado antes, se activa únicamente cuando se envían paquetes de notificación de alarma ó de prueba de covertura. Mediante una variable interna de control y este temporizador, se controla que la aplicación de operador haya recibido el mensaje de alarma o prueba de covertura que haya enviado la mote. Este temporizador no es repetitivo, sólo se activa de forma puntual cuando se envía uno de los dos mensajes citados.

- SerialInterface.MessageSent, envío de mensaje serial finalizado:

  Este evento permite saber que, además de haber terminado de enviar un mensaje por interfaz serie, se ha enviado bien o no. Esto último, en el sistema de detección de incendios no se ha usado porque los únicos mensajes importantes son las notificaciones de alarma y ya se ha implementado un sistema, indicado en el punto inmediantamente superior, para reenviar los mensajes en caso de problemas.
  Al no usar esa característica, simplemente se liberará el bloqueo de semáforo sobre el envío de mensajes serie para que el temporizador correspondiente pueda proceder a hacer otro envío en su próximo ciclo.

- SerialInterface.MessageReceived, mensaje recibido en el interfaz serial: Evento disparado cuando se termina de recibir un paquete en la interfaz serial de la mote (por tanto, sólo sucederá en la mote conectada al PC).
  Según el tipo de mensaje recibido, se harán distintas acciones. Los 4 mensajes que recibirá la mote a través de este interaz, son:

  - HelloMsg: Si el mensaje es para ésta o todas las motes, la mote se irá a estado MS_INI_INI, lo que significa un reinicio funcional.
    Si el mensaje es para otra mote en concreto, simplemente lo enviará a la cola de envíos de radiofrecuencia.

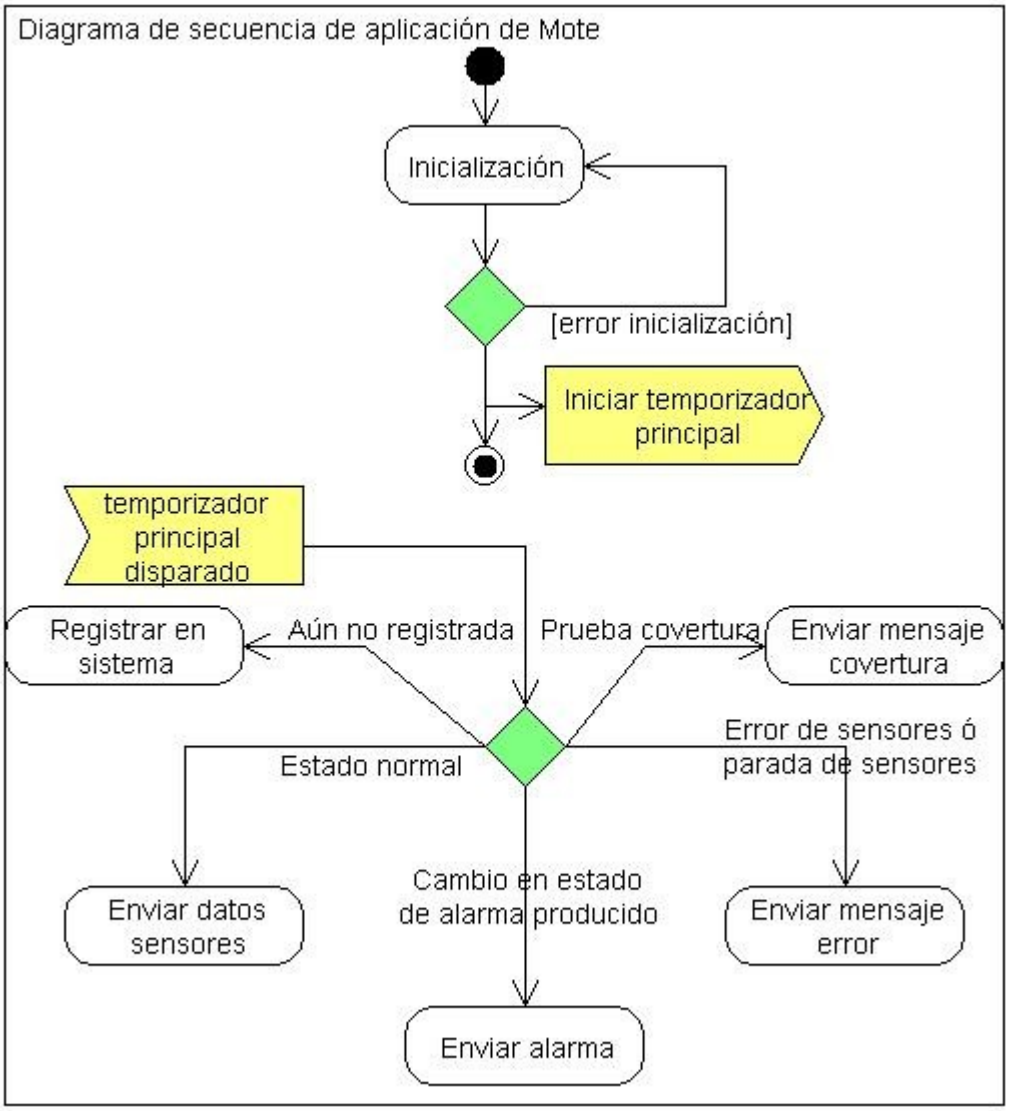
- AckMsg: Si el mensaje es para la mote que lo recibe: si es un mensaje de confirmación de lectura de un mensaje MACMsg y el estado es MS_INI_INI, la mote pasa a estado MS_INI_OK. Si el mensaje es de confirmación de lectura de una alarma, el estado de alarma pasa a notificada. Si es de confirmación de lectura de un mensaje CovMsg, se reinicializa el contador de reintentos de envío de mensajes de prueba de covertura. Si en cambio el mensaje es para otra mote, lo enviará a la cola de radiofrecuencia.
- ConfigMsg: Si el mensaje es para la mote que lo recibe: se aplica la configuración recibida, se pasa a estado MS_RUNNING (funcionamiento normal) si no está en estado de alarma (ya sea local o remota) y se envía confirmación de recepción. Si es para otra mote, se reenvía.
- AlarmMsg: Si es un mensaje de alarma general, se aplica el estado de alarma general recibido (activa o no activa). Si no va dirigido únicamente a esta mote, se coloca en la cola de radiofrecuencia. Si el mensaje es para esta mote y es un reconocimiento por parte del operador, el estado de alarma local pasa a estado de alarma reconocida.

- RFCInterface.MessageSent, envío de mensaje de radiofrecuencia finalizado: Indica que se ha terminado de enviar un mensaje por el interfaz de radiofrecuencia, por tanto se libera el semáforo de bloqueo de envíos por radiofrecuencia para que el temporizador que actúa sobre dicha cola pueda continuar enviando mensajes encolados.

- RFCInteraface.MessageReceived, recepción de mensaje en interfaz de radiofrecuencia: Evento disparado cuando la interficie de radiofrecuencia termina de recibir un mensaje. Este evento, respecto de su evento homólogo sobre la interaz serial, tiene la particularidad de que sucederá en los dos tipos de motes: la conectada al PC y las externas. Sin embargo, se ha desarrollado de forma que prácticamente sean igual en cuanto a codificación, se detalla:
  Si el mensaje es recibido en la mote conectada al PC, simplemente lo coloca en la cola de envío serial. Esto es así de simple porque la mote conectada al PC no desarrolla lógica funcional del sistema sobre la recepción de los mensajes, esto se ha delegado totalmente en la aplicación de operador, además de no ver la necesidad de hacerlo de otra forma. Si el mensaje se recibe en una mote externa, el comportamiento es exactamente igual que para el caso de la interficie serial, pero no teniendo la necesidad de reenviar el mensaje a ninguna otra cola: el mensaje, o es para la mote que lo recibe, o es para todas; en el primer caso, se atiende, y en el segundo se descarta.

Diagrama de secuencia de la aplicación

Ilustración 19: Aplicación de mote, diagrama de secuencia



En esta ilustración se aprecia una visión general de la lógica secuencial que sigue la aplicación de la mote. La inicialización ocurre hasta que se realiza sin errores, y en ese momento se programa el temporizador principal, y se sale del hilo de ejecución.

Al ser disparado el temporizador principal, se evalúan ciertas variables y se actúa en consecuencia.

Casos de uso de la aplicación de mote:

Caso de uso #1:

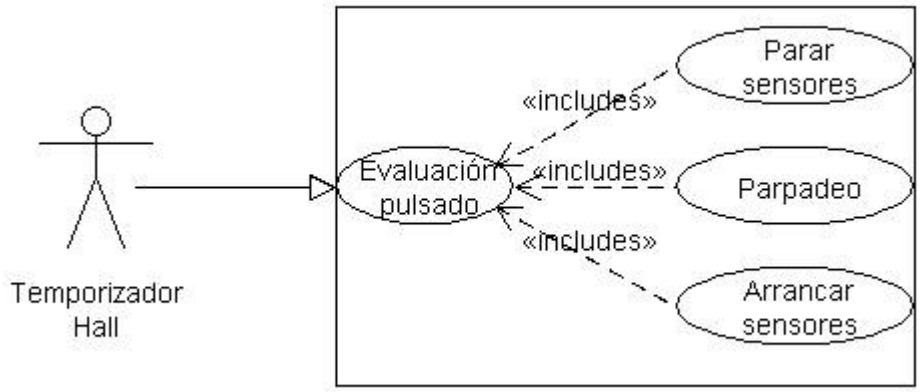


Ilustración 20: Aplicación mote, caso de uso temporizador hall disparado

Como se comentó, una vez se detecta el pase de un imán sobre el sensor de hall, se comenzará una captura del número de pases de imán sobre dicho sensor.

¿Qué significa y qué supone esto? La primera vez que se pase un imán, se comprobará si ya se está capturando el número de pases de imán, esto se controla con una variable interna del módulo del sensor de hall. En caso de no ser así, se arrancará el temporizador que se disparará una vez haya pasado el tiempo máximo estipulado para que el usuario haya efectuado todos los pases de imán que desee.

Mientras dicho temporizador siga en marcha, cada pase de imán supondrá un aumento del contador interno. Cuando salte el temporizador, se lanzará el evento hacia la clase principal del programa de la mote, TfcC.nc, indicando el número de pases de imán capturados.

La clase TfcC.nc, evaluará entonces (punto de entrada de la imagen) el número de pulsaciones (y en caso de no ser 2 ó 4, las establecidas para activar o desactivar respectivamente los sensores, se sale de la rutina en este mismo punto), lanzará un breve parpadeo de los 3 LEDs (para que el usuario sepa que se ha tenido en cuenta su acción) y se pararán o arrancarán los sensores según el número de pases de imán notificados por el módulo de sensor de hall.

Caso de uso #2:

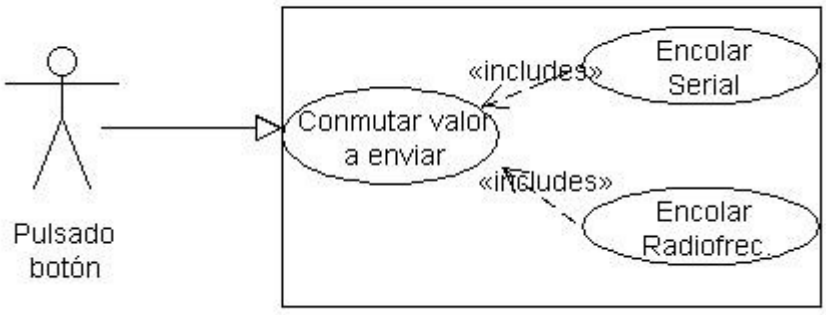


Ilustración 21: Aplicación de mote, caso de uso pulsado rápido botón usuario

En el caso del pulsado rápido de botón de usuario (es decir, pulsado de menos de 2 segundos), el módulo de control de sensores emite un evento a la clase principal (TfcC.nc) y ésta envía un mensaje de alarma local con valor distinto al último enviado (almacenado en una variable de este módulo). Es decir, la primera vez que esto pase tras un reinicio de la mote (reinicio, no reinicialización ya que guarda el estado de dicha variable) enviará un mensaje con valor alarma activa, y la siguiente vez con valor alarma inactiva.

Según se trate de la mote conectada al PC ó una mote externa, el mensaje se encolará en la cola serial ó de radiofrecuencia respectivamente.

Caso de uso #3:

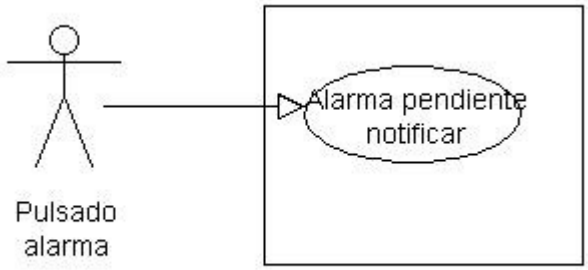


Ilustración 22: Aplicación de mote, caso de uso pulsado alarma

En el caso del pulsado del botón de alarma durante 2 a 10 segundos, el módulo de control de sensores enviará un evento a la clase principal indicando tal suceso. La clase principal, entonces, actualizará su estado interno de alarma en base al tipo de alarma sucedida (en este caso, manual) y también pasará su estado de notificación de alarma a "pendiente notificar".

Caso de uso #4:

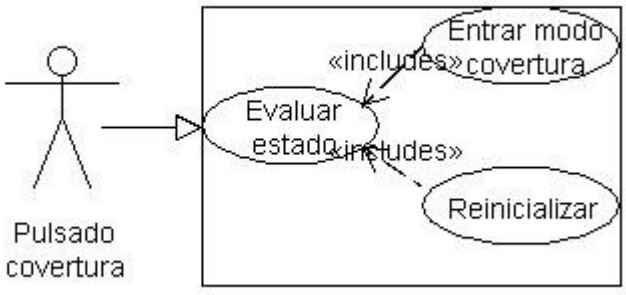


Ilustración 23: Aplicación de mote, caso de uso pulsado de covertura

En el caso del pulsado de covertura (esto es, pulsado del botón de usuario durante más de 10 segundos), el módulo de control de sensores envia un evento a la clase principal indicando tal suceso, y ésta, evaluando el estado actual de la mote (si ya está o no en modo de prueba de covertura), decide si entrar en el modo (lo cual envía un mensaje de error con un valor concreto a la aplicación de operador) ó reinicializar la mote (el estado interno de la mote pasa a MS_INI_INI, lo cual incurre en un envío de un mensaje MACMsg, esperar respuesta ACK, luego una petición de configuración, ...).

Caso de uso #5:

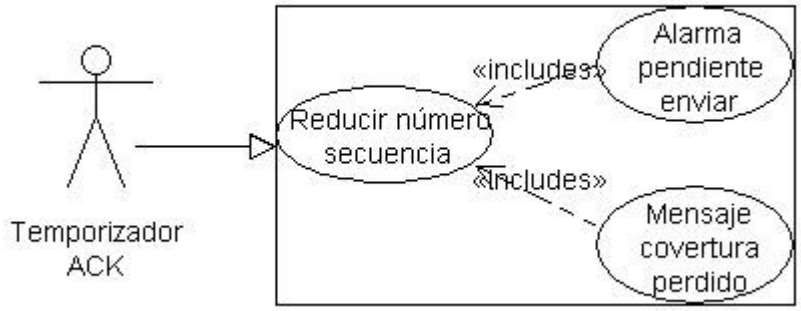


Ilustración 24: Aplicación de mote, caso de uso temporizador de ACK disparado

Para los mensajes que se ha implementado una lógica de confirmación de lectura de mensaje, caso de envío de alarma y prueba de covertura, el temporizador para mensajes ACK es my importante.

Este temporizador se programa cuando se envía uno de estos dos tipos de mensajes. Cuando se dispara, si no se ha recibido aún el mensaje esperado, se reduce el número de secuencia interno y se aumenta el contador de reintentos de envío de mensajes de covertura en caso de estar esperando un mensaje de covertura.

El hecho de saber si la aplicación está esperando o no un mensaje ACK es una variable interna que se establece a valor afirmativo cuando se envía uno de estos dos mensajes, y a valor falso cuando se recibe el mensaje ACK esperado ó en su defecto, cuando el temporizador ACK salta.

En el caso de la prueba de covertura, simplemente aumentar el contador interno de reintentos es suficiente. En el caso de la notificación de alarma, la variable de control de notificación de alarma local interna pasará de "pendiente ACK" a "pendiente notificación", lo cual incurrirá en que en el próximo ciclo del temporizador principal del módulo principal (TfcC.nc) se vuelva a notificar la alarma.

Caso de uso #6:

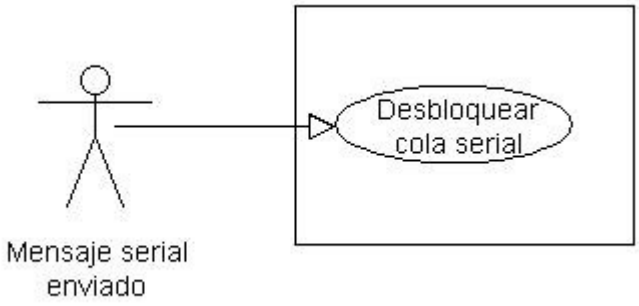


Ilustración 25: Aplicación de mote, caso de uso mensaje serial enviado

Como se ha comentado, el evento enviado por el módulo de comunicación serial al módulo principal (SerialC -> TfcC) supone simplemente liberar el semáforo de bloqueo de la cola serial para poder continuar enviando en el próximo disparo del temporizador que trabaja sobre la cola serial.

No se evalúa si el resultado del envío ha sido exitoso o no, ya que si realmente se necesita la garantía de que el mensaje ha llegado a destino, la lógica ACK implementado sobre los mensajes cruciales de la aplicación (covertura y alarma) forzará un reenvío del mensaje hasta tener confirmación de recepción.

Caso de uso #7:

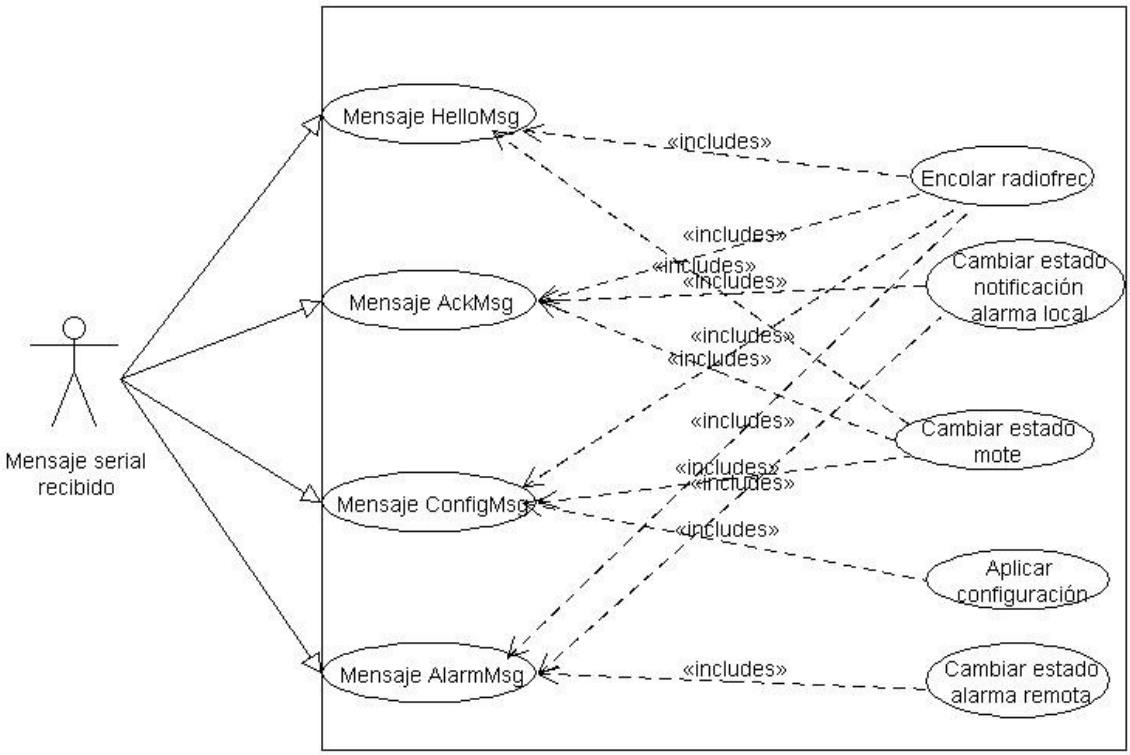


Ilustración 26: Aplicación de mote, caso de uso recepción mensaje serial

El caso de la recepción de mensajes serial, se debe dividir según el tipo de mensaje recibido, aunque todos ellos acaben llamando a las mismas rutinas internas, aunque siempre evaluando ciertos campos del mensaje para saber si se debe llamar realmente a una cierta rutina.

En el caso de un mensaje de bienvenida HelloMsg, el mensaje se encolará en caso de ser para todas u otra mote, y en caso de que sea para ésta o todas, se reinicializará la mote.

El mensaje AckMsg, podrá cambiar el estado de la mote y su estado de notificación de alarma interna o irse a la cola de mensajes de radiofrecuencia.

El mensaje de configuración, podrá ser encolado simplemente ó cambiar el estado de la mote y además aplicar una nueva configuración sobre ella.

El mensaje de alarma podrá cambiar el estado de alarma remota interno de la mote, ser encolado y cambiar el estado de notificación de la alarma local.

Caso de uso #8:

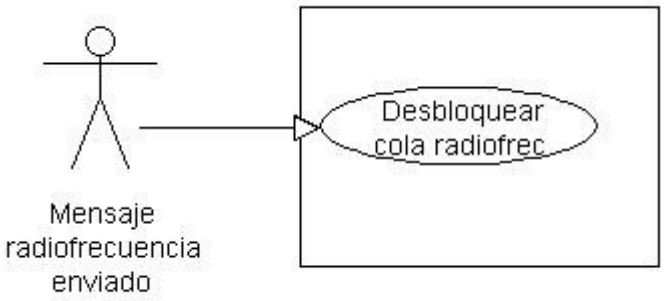


Ilustración 27: Aplicación de mote, caso de uso mensaje radiofrecuencia enviado

Al igual que en el caso de mensaje serial enviado, esta rutina simplemente libera el bloqueo sobre el tratamiento de la cola de mensajes, en este caso de mensjes de radiofrecuencia.

Al igual que en el caso serial, no se evalúa el resultado del envío, si se requiere confirmación de recepción del mensaje, la lógica ACK sobre los mensajes de covertura y alarma forzará un reenvío del mensaje hasta tener la confirmación.

Caso de uso #9:

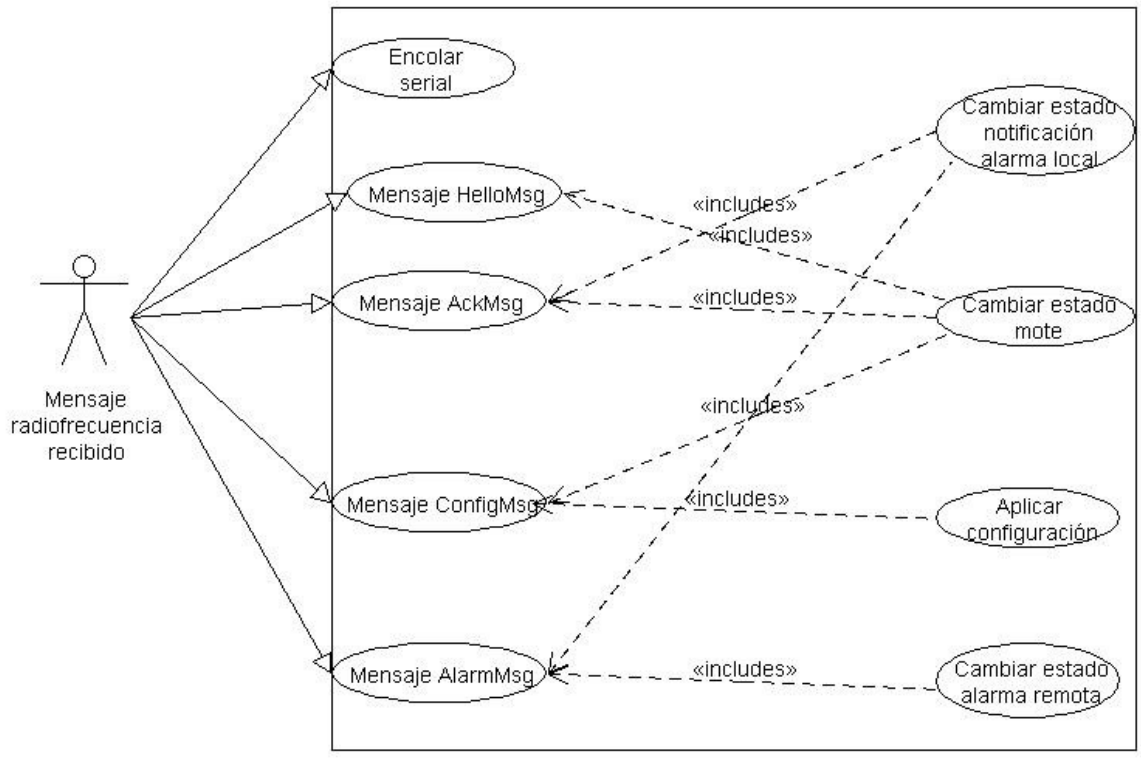


Ilustración 28: Aplicación de mote, caso de uso recepción de mensaje radiofrecuencia

Este caso es un poco más sencillo que el de la recepción de mensajes serial. Los efectos son los mismos y la casuística también, pero eliminamos la necesidad de encolar el mensaje recibido sobre la cola de radiofrecuencia, ya que este mensaje se está recibiendo en una mote externa y por tanto no hay que reenviarlo, o se está recibiendo en la mote conectada al PC y por tanto el mensaje directamente se encola para ser enviado al PC.

El reenvío a la cola serial se ha puesto como un punto a parte para más claridad, ya que realmente en cada tipo de mensaje se evaluará primeramente si se debe o no encolar, sin atender a nada más.

Viabilidad técnica del proyecto

Punto de vista general

El sistema es totalmente plasmable en la realidad. Durante el desarrollo, no ha habido mayores problemas, de hecho se puede sacar aún mucho más potencial, pero el planteamiento inicial es suficientemente funcional.

Punto de vista organizativo

Una sola persona, sin conocimiento previo de este tipo de dispositivos, disponiendo de 2 motes, ha desarrollado el sistema. Teniendo en cuenta este hecho, de este proyecto se puede sacar gran rentabilidad, ya que el desarrollo es puntual (luego sólo se necesitarían pocas horas para adaptaciones bajo petición y posibles depuraciones de errores).

En cuanto a los dispositivos, inicialmente la opción es buscar un proveedor, pero a largo plazo desarrollarlos en la misma organización que lo quiera distribuir, el coste sería mínimo tras una pequeña inversión en investigación para poder realizar motes tan capaces como las usadas en este proyecto (cou24, revisión 2). Además, en caso de invertir en desarrollar unas propias, se podrían adaptar mejor a las posibles necesidades surgidas de extensiones a las funciones del sistema, ó a nuevos proyectos de la organización.

Puntos fuertes del sistema

Es totalmente escalable, de fácil instalación y de fácil mantenimiento (tanto en hardware como en software). Además, flexible geográficamente, adaptándose a cualquier entorno de producción destino.

Es totalmente ampliable en cuanto a mensajes debido a la gran ventaja que da el sistema operativo de las motes de abstraerse de las capas de comunicaciones de bajo nivel. Además, la forma en que se han orientado los mensajes definidos en la aplicación, nos da la posibilidad de poder definir uno de propósito general que pueda englobar los de alarma, error y posibles nuevos mensajes (por ejemplo un juego de órdenes que las motes puedan recibir de la aplicación).

Puntos débiles del sistema

En esta versión del sistema, no se ha contemplado la opción de actualización remota del código de las motes ó la existencia de motes de redifusión. No obstante, es algo totalmente viable.

Las motes no se conectan de forma dinámica al sistema en cuanto a identificadores: cada una lleva el identificar definido en tiempo de compilación. Lo ideal sería definir una codificación en base a la MAC de cada mote. Así sería aún más sencillo el mantenimiento del sistema en producción.

Valoración económica

Costes ya asumidos

El coste del análisis y el desarrollo son costes ya asumidos en el momento de presentar este documento. Como se comenta en el punto anterior, se ha iniciado con un recurso humano carente de conocimiento, y todo el análisis y el desarrollo han supuesto alrededor de 200 horas. Este volumen de horas reduce su coste si se contratan en un paquete, pudiendo hablar de **3000 €** (15 € / hora).

En cuanto al precio de las motes, rondan los 80 € cada una, por tanto se han asumido **160 €**.

Al ser software libre, no se necesitan licencias.

Costes de infraestructura

Estos costes comprenden:

- máquina de operador: un simple PC, **400 €**.
- baterías: cargador de baterías + baterías: **35 €**.
- preparación máquina de operador (Sistema operativo + aplicación operador) (opcional si la organización puede hacerlo): **35 €**

Costes de mantenimiento

El sistema no necesita mantenimiento, ya que lo único que se debe hacer periódicamente es cambiar las baterías de las motes, cosa que los propios usuarios (o el operador en entornos de producción sin personas) pueden hacer.

Costes de personal

El perfil del operador no necesita ningún tipo de conocimiento, sólo saber usar un ordenador. Esto se puede conseguir por un salario BRUTO de **1000 € / mes**. Si la organización ya cuenta con personal encargado de monitorizar / vigilar la organización, puede ahorrarse este gasto añadiéndolo a las funciones que desarrolle dicho personal.

Coste de venta del SDI

El precio de venta del sistema será de **4000 €** contando con 5 motes y el PC de operador preparado. Cada mote supondrá **250 € extra**.

Tabla resumen de costes

Infraestuctura		
	PC operador	400
	Baterías	35
	Preparación PC operador	35
Mantenimiento		
	Sueldo operador	1000
		1.470 €

Venta del SDI		
	SDI + 5 motes + PC operador listo	4000
		4.000,00 €

Como se ve, el precio de venta compensa en gran medida el coste de desarrollo del proyecto. Así mismo, para el cliente, el sistema no supone gran coste fijo.

Conclusiones

Objetivos

Los objetivos del SDI indicados en el apartado "1.3 Objetivos", han sido cumplidos. Así pues, no sólo se consigue un SDI firme, sino que también se convierte en una herramienta imprescindible para una organización que pretenda sentirse segura ante incendios.

Se indica brevemente qué decidisiones / particularidades funcionales han llevado a conseguir cada punto, desde un punto de vista técnico de aquellos puntos que realmente han requerido un planteamiento puntual:

- Notificación de alarmas locales: Se ha desarrollado un sistema de doble confirmación para la alarma como se ha visto en los puntos anteriores. Este sistema consiste en una confirmación automática por parte de la aplicación de operador de que se ha recibido la entrada o salida de estado de alarma local en la mote y, posteriormente en caso de que el estado de alarma sea activo, un mensaje de reconocimiento por parte del operador. Todo esto, visible para un usuario que esté junto a la mote mediante el código de luces implementado.
- Activación manual de alarma: Para este punto, no sólo se ha tenido en cuenta una pulsación del botón de usuario, sino que también se ha ampliado el abanico de posibilidades de interacción entre el usuario y la mote, combinando el tiempo de pulsado y el pulsado (del botón de usuario) para obtener distintos resultados, siendo uno de ellos el de la activación manual de alarma en una mote (y siendo los otros el de emisión puntual de mensaje de alarma y el de activación/desactivación de modo de priueba de covertura).
- Notificación fiable de alarmas: A parte de lo indicado en el primer punto de este epígrafe, la notificación fiable de alarmas se consigue mediante un envío recurrente de notificación del "nuevo" estado de alarma de la mote; esto es: cuando la mote cambia su estado de alarma local (ya sea porque entre o salga del estado local), se enviarán notificaciones del nuevo estado a la aplicación de operador hasta que se reciba una confirmación.
  Debido al trabajo sobre colas de mensajes en la aplicación de la mote, se da el caso de que a veces la aplicación de operador recibe notificaciones después de haber enviado la confirmación de recepción a la mote. Esto se debe a que se antepone la redundancia a la carencia de seguridad. Esta recepción redundante de la notificación no tiene ningún impacto negativo sobre la aplicación de operador.
- Indicar visualmente el estado de la mote: Aunque ha sido difícil determinar con 3 LEDs los distintos estados de la mote, se ha conseguido. Como se ha explicado anteriormente en este documento, cada estado tiene su particular modo de expresión visual. Se ha intentado no disparar el consumo de batería con el módulo visual, por lo que en la medida de lo posible se han usado parpadeos en vez de encendidos constantes. De hecho, un encendido constante debería producirse el menor tiempo posible, y de lo contrario, hay algún problema (error ó alarma local, alarma remota ó mensaje perdido / falta de recepción de mensaje necesario desde la aplicación de operador).
  En la documentación del manual del operador se indica el código de luces.
- Protección de caídas de las motes: Ante la posibilidad de un estado inconsistente de la mote, la funcionalidad WatchDog, comentada en más de una ocasión en este documento, se ocupará de reiniciarla.
  También, el reinicio de la aplicación de operador, siendo una aplicación "joven" (sin haber

recibido el testeo de varios usuarios y distintos entornos de producción que la hagan evolucionar adaptándose mejor a ambos aspectos) puede ocasionar un estado inconsistente informativo, por lo que poder enviar el mensaje HelloMsg para que las motes se reinicialicen, es una funcionalidad bastante importante. Saber que el operador puede volver a enviarlo bajo demanda a una mote en particularidad, también da la opción de reiniciarla en caso de existir sospechas de mal funcionamiento.

- Prueba de covertura: El modo de probar que una mote esté o no dentro del rango de covertura de comunicación inalámbrica respecto del PC de operador, se ha dejado como una funcionalidad que hará interactuar a un usuario que posea la mote en sus manos y la propia aplicación de operador. Esto es así porque durante el funcionamiento normal del SDI, una mote que pierda covertura producirá un mensaje en la ventana del operador cada vez que venza el tiempo de recepción de otro mensaje de sensores.
  No obstante, en este modo se puede entrar en cualquier momento si el usuario así lo desea, por lo que en caso de una restructuración de mobiliario / disposición de las paredes del edficio que nos haga sospechar de pérdida de covertura, se puede probar.
- Activación y desactivación de una mote: Esta funcionalidad permite poder no tener en cuenta los datos de sensores de una mote si por ejemplo sabemos que por algún motivo aumentará la luminosidad o temperatura de forma controlada (por ejemplo, obras). No por estar desactivada, estará al margen de alarmas remotas ó reinicializaciones por orden de la aplicación de operador.
- Aplicación de operador simple y de fácil despliegue: La aplicación es clara visualmente, dividiendo la ventana del operador en dos áreas, una de interacción e información y la otra sólo de información.
  El despliegue es sencillo, sólo se necesita partir de un PC preparado, y esto se consigue simplemente instalando un sistema operativo Linux y las librerías indicadas. Una empresa de servicios lo podría hacer en una hora de trabajo en caso de que nadie de la organización pudiera. No obstante, se recomienda tener dos PCs de operador en caso de que uno falle.
- Comunicaciones sencillas e independencia de caídas de motes: El SDI pretende reducir el número de mensajes, su complejidad y su diversidad. También, la caída de motes es transparente de cara a la funcioalidad del sistema (obviamente, la caída de la mote conectada al PC de operador supone la pérdida de los mensajes en sus dos colas e indisponibilidad del sistema hasta su recuperación).
- Aplicación de mote clara y mantenible: Si la organización pretende tomar el control de la aplicación del SDI que ha adquirido, puede hacerlo sin problema, la aplicación se ha modularizado de forma que no sea difícil entenderla y modificarla.

Propuesta de mejoras

Como puntos fuertes a implementar sobre el SDI, y que se habrían implementado de poder extender en el tiempo más el sistema sin perder fiabilidad de lo ya implementado, están:

- Mejoras visuales en la aplicación de operador: Esta aplicación, tal cual se presenta en este documento, en su versión inicial, no tiene una gran perfección visual. Se ha buscado que sea funcional sin molestarse en la parte gráfica. Se podría mejorar para que sea más vistosa.
- Soporte web: Convertir la aplicación de operador en un sitio web. Para ello, se crearía un servicio de sistema operativo que recogiera los mensajes recibidos a través de la comunicación serie y los colocaría en una base de datos de la organización. También en esta BBDD tendríamos los mensajes que la aplicación de operador necesitase enviar a las motes. Esta base de datos sería consultada por un sitio web de operador, que interactuaría con ella para colocar nuevos mensajes a enviar a las motes.
  Esto da al SDI la posibilidad de tener un management remoto del sistema, ya que este sitio web podría ser consultado desde internet, ya sea por VPN ó filtrado por IP para tener seguridad de quién entra en el site y tenerlo sólo en la intranet organizativa, evitando ataques si lo tenemos en internet a través de una IP pública.
  En este caso, la primera propuesta de mejora no tendría sentido sobre la aplicación actual de operador, sino que ya sobre su versión web se desarrollaría lo que se creyese conveniente.
- Incorporar la funcionalidad del SerialForwarder en el SDI: Un punto importante que daría más facilidad a la hora de desplegar el SDI es que la funcionalidad del programa SerialForwarder, el programa que pasa los mensajes recibidos por el puerto serie a la red de la máquina del operador, se pasase o a la aplicación de operador que se entrega con el sistema ó en el posible servicio que se desarrollase si se cumpliese el punto anterior de propuesta de mejora. Así, sería menos trabajo para el operador. Además, una caída del SerialForwarder, actualmente tira la aplicación de operador. En este caso, se implementaría que una caída de la comunicación con el puerto serie, no supusiese una caída de la aplicación, si no reintentos constantes para recuperar el sistema.
- Sub-redes: Posibilidad de poder fragmentar el SDI en dos ó más subredes. Esto es, que una serie de motes puedan subscribirse a los mensajes diferidos por una mote en concreto, la cual sería una base station. Así, podríamos tener la monitorización de una organización por plantas ó departamentos.
- Sistema redundante de motes basestation: Desarrollar la posibilidad de tener dos ó más motes conectada al PC del operador (basestations) de forma que sólo una esté en funcionamiento, y en caso de caída de ésta, la siguiente se active.
  Esto daría versatilidad, ya que podría tenerse un segundo PC al que conectar una mote y así, si lo que se queda fuera de servicio es el PC, la otra mote se activaría automaticamente, minimizando el tiempo de recuperación, que en caso de ser el PC el que se cae, es al menos el tiempo de arranque del propio PC en caso de no tener problemas de hardware ó software.
- Desarrollar la funcionalidad del programa SerialForwarder para otros sistemas operativos como por ejemplo Windows. Esto es un punto muy fuerte, ya que el SerialForwarder envía a la red de la máquina los paquetes, por lo que podría colocarse la mote basestation en cualquier máquina de la red, y el paquete ser tratado por una aplicación de operador en un PC con linux en otro punto de la organización.
  Un siguiente paso sería desarrollar todo el sistema para Windows, reduciendo el trastorno técnico para la organización de tener que aprender a trabajar con máquinas Linux, ya que casi todas las organizaciones trabajand con Windows.

Autoevaluación

El resultado conseguido me ha dejado bastante contento. Me hubiera gustado tener más tiempo para poder desarrollarlo y darle más funcionalidades, pero me ha sido imposible.

La segmentación modular creo que es uno de los puntos más fuertes del código, ya que nos permite abstraernos de toda la funcionalidad de la mote salvo la que realmente nos interese investigar. También, la interacción entre módulos se queda mucho mejor definida, por lo que la búsqueda de errores es más fácil.

He visto que un mero error de código sobre NesC puede suponer muchas horas hasta encontrarlo, como me ha pasado cuando trabajaba sobre el puntero a un mensaje recibido y al encolarlo apuntando a la misma dirección de memoria, luego se enviaba información corrupta. Hasta encontrar que el problema estaba en esta línea olvidada (y derivada de inicialmente no usar colas, sino hacer los envíos directamente), perdí unas 6-8 horas. Además, ha sido un error tonto para mí, ya que estoy acostumbrado a trabajar a bajo nivel, pero en este caso entendí mal la segmentación de memoria de la mote y el redimensionamiento de variables a través de un cast para obtener una copia del bloque de memoria... Gracias a estos problemas, he perfeccionado un poco más mis técnicas de programación de cara a la búsqueda de errores.

La imposibilidad ó gran dificiltad que parece existir para poder programar desde windows hacia NesC me ha intrigado y si dispongo de tiempo en los próximos meses, investigaré cómo poder solucionar esto, o al menos construir un SerialForwarder para windows, de forma que la comunidad TinyOS se beneficie de ello y futuros alumnos de este aula puedan ver mejorado su testeo al poder conectar las motes a PCs con Windows, que son los más comunes. Esto también supone una mejora para el SDI, que se explica en el último punto del apartado de mejoras.

El proyecto, no se quedará aquí, tengo pensado trabajar con motes y comercializar productos basados en ellas y ofrecerlos en mi empresa. El problema, será que deberé desarrollarlos sobre motes estándar que pueda adquirir y más adelante desarrollar mis propias motes para reducir los costes y poder ofrecer motes completas, que detecten todo lo que se pueda desde una mote (luz, temperatura, movimiento externo, movimiento de la mote, hall,viento...).

Glosario

- Interfaz: Módulo de interconexión.
- Mote: Dispositivo con interfaz inalámbrico y serial para poder conectarse a un PC. También tiene sensores. Funciona con baterías.
- Linux: Sistema operativo para Pcs.
- Java: Lenguaje de programación para PCs. En este lenguaje de programación se desarrolla la aplicación de operador.
- NesC: Lenguaje de programación para las aplicaciones de las motes.
- MAC: Dirección física grabada de forma permanente en un interfaz de red. Su código viene determinado por dos partes: la primera es un código fijo perteneciente al fabricante, la segunda es su numerador de serie dentro de la cadena productiva del fabricante.
- Byte: Medida de almacenamiento lógico. Se compone de 8 bits, que son las unidades de almacenamiento lógico más pequeñas.

Anexo I

Manual de usuario

1. Arranque

Una vez iniciada la aplicación, mandará un mensaje de bienvenida a la red. Este mensaje hará que las motes se reinicialicen y envíen una petición de configuración.
Una vez reciban su configuración, comenzarán a enviar los datos de los sensores.

2. Gestión configuración

Por defecto se indican unos parámetros dentro del funcionamiento normal de las motes. Se pueden modificar desde el menú contextual (botón derecho sobre la línea correspondiente a una mote en su listado)
o desde el menú "Motes" de la ventana, apartado Configurar, pudiendo desde ahí hacer el envío de la nueva
configuración a una o todas las motes.
Por cualquiera de los dos caminos, se pedirá al usuario que indique cada valor de configuración.

3. Gestión de Alarmas remotas

Las alarmas serán notificadas automáticamente por las motes cuando se detecten valores fuera de los
límites especificados.
Cuando una alarma llega, el aplicativo envía automáticamente un mensaje de recepción, quedando pendiente
que el usuario haga un reconocimiento manual de la alarma (mediante el menú contextual del listado de motes).
Justo al texto que nos indica el tipo de alarma remota producido (en el listado de motes), aparecerá la hora
a la que se ha producido.

4. Gestión de Alarmas generales

Es posible enviar una alarma general del sistema a todas las motes. Esto se realiza desde el menú "Motes" de
la ventana, apartado "Alarma General". Se nos pedirá indicar un valor: 1 significa alarma, 2 significa fuera de
alarma.
Una vez introducido, el mensaje será enviado inmediatamente a las motes, y ellas contestarán con un mensaje para
garantizar su recepción. Hasta que no se reciba dicho mensaje de cada mote, justo a su estado de alarma aparecerá
el texto "Pdte. ACK".

Preparación del entorno de desarrollo

Qué se necesita:

- PC con sistema operativo Ubuntu versión 9 ó 10. No se ha verificado que funcione con otras versiones.
- "Árbol" de TinyOS (sistema operativo de las motes) instalado sobre dicho PC (se indica cómo obtenerlo en la wiki del aula)
- Entorno java 1.6 ó superior instalado en el PC (se garantiza funcionamiento con 1.6 y 1.7). Se puede descargar de www.java.com.
- También hace falta descargarse las bibliotecas y código fuente para los chips AVR, lo cual se encuentra detallado en la wiki de TinyOS. Deberemos tener en cuenta dónde extraeremos los packetes de AVR, porque deberemos cambiar una línea en el código fuente de las motes para poder compilarla: el include del fichero wdt.h que se hace en la tercera línea de TfcC.nc. Este include ha sido necesario para la funcionalidad WatchDog implementada sobre las motes.
- una mote conectada vía USB (la basestation) al PC en cuestión

Instrucciones de compilación

Abriremos una consola de linux y procederemos de la siguiente forma.

BaseStation:

- Nos situamos en la carpeta /TFC/src de la raíz del código fuente:

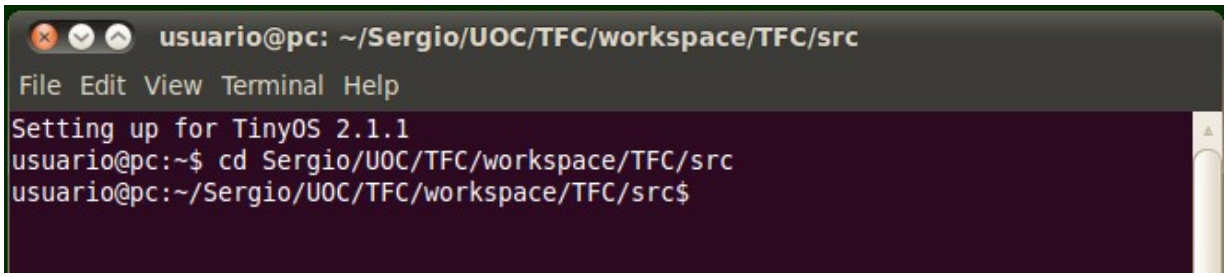


Ilustración 29: Situarse en carpeta de código fuente

- "make cou24"

```
usuario@pc:~/Sergio/UOC/TFC/workspace/TFC/src$ make cou24
mkdir -p build/cou24
    compiling TFCAppC to a cou24 binary
ncc -o build/cou24/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all -
target=cou24 -fnesc-cfile=build/cou24/app.c -board= -DDEFINED_TOS_AM_GROUP=0x22
--param max-inline-insns-single=100000 -DIDENT_APPNAME=\"TFCAppC\" -DIDENT_USERN
AME=\"usuario\" -DIDENT_HOSTNAME=\"pc\" -DIDENT_USERHASH=0x02173ef2L -DIDENT_TIM
ESTAMP=0x4fb60ab6L -DIDENT_UIDHASH=0x9b567730L -fnesc-dump=wiring -fnesc-dump='i
nterfaces(!abstract())' -fnesc-dump='referenced(interfacedefs, components)' -fne
sc-dumpfile=build/cou24/wiring-check.xml TFCAppC.nc -lm
    compiled TFCAppC to build/cou24/main.exe
          26502 bytes in ROM
           1093 bytes in RAM
avr-objcopy --output-target=srec build/cou24/main.exe build/cou24/main.srec
avr-objcopy --output-target=ihex build/cou24/main.exe build/cou24/main.ihex
    writing TOS image
usuario@pc:~/Sergio/UOC/TFC/workspace/TFC/src$
```

Ilustración 30: Compilar aplicación para mote BaseStation

- "meshprog -t/dev/ttyUSB**0** -f ./build/cou24/main.srec" (y una vez ejecutado, pulsamos el botón de reset de la mote)

```
,[]
,[]
,[]
,[]
,[]
,[]
,[]
.........,[i]
Starting transmission.
finished!
usuario@pc:~/Sergio/UOC/TFC/workspace/TFC/src$
```

Ilustración 31: Actualizar programa en mote BaseStation

Motes remotas:

- Nos situamos en la carpeta /TFC/src de la raíz del código fuente:
- make cou24 install,**2**
- meshprog -t/dev/ttyUSB**0** -f ./build/cou24/main.srec.out-**2**

Siendo en ambos casos:

- **0**: El Id del puerto USB al que está conectada la mote. Este valor lo obtenemos ejecutando en consola el comando "motelist":

```
usuario@pc: ~/Sergio/UOC/TFC/workspace/TFC/src
File  Edit  View  Terminal  Help
usuario@pc:~/Sergio/UOC/TFC/workspace/TFC/src$ motelist
Reference  Device           Description
---------- ---------------- ----------------------------------------
0001       /dev/ttyUSB0     Silicon Labs CP2102 USB to UART Bridge Controller
usuario@pc:~/Sergio/UOC/TFC/workspace/TFC/src$
```

Ilustración 32: Comando motelist para mostrar el puerto de conexión de la mote

- **2**: El Id de la mote remota sobre la que vayamos a cargar el programa. Este Id lo determinamos nosotros mismos. Sólo debemos tener en cuenta que esté dentro del máximo manejable por la aplicación de operador (constante definida en la clase Main como "MAXMOTES") y que no se repita en ninguna otra mote activa. El número 0 se reserva por código para la aplicación de operador, y el 1 para la BaseStation (por omisión, es el Id que se asigna en la compilación "make cou24").

Aplicación de operador:

Al tratarse de una clase java, es suficiente situarse en la carpeta de código de la aplicación del operador, /AplicOperador/, y ejecutar "java gui" para compilar + lanzar la aplicación. Gui es la clase principal (Graphical User Interface), existe una clase primitiva, Main, que es una versión más ligera para consola, se puede ejecutar con "java Main".

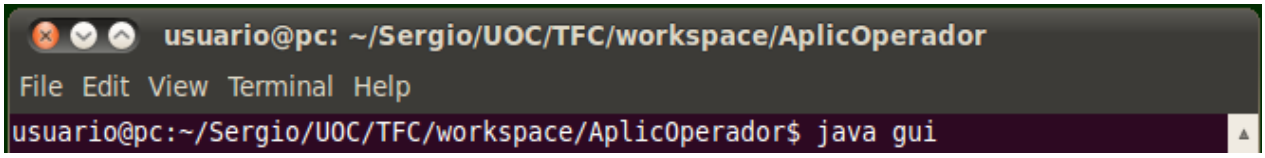


Ilustración 33: Compilación-Ejecución de la aplicación de operador