Sistema Contra Incendis

Jaume Roca Badia
Enginyeria Tècnica en Informàtica de Sistemes

Jordi Bécares Ferrés

3 de gener del 2012

Resum

Aquest projecte ha consistit en desenvolupar un sistema contra incendis utilitzant per tal efecte dues motes cou24, emissora, receptora i un ordenador. La mota emissora pren valors dels sensors de temperatura i bateria els envia a la mota receptora per monitoritzar-los, si la bateria baixa d'un límit preestablert enviarà una alarma de bateria baixa a la mota receptora, si la temperatura supera el límit de temperatura preestablert enviarà l'alarma d'incendi a la mota receptora.

La mota receptora s'encarrega d'enviar els missatges que provenen de la mota emissora a l'ordenador on estigui connectada i també d'enviar els missatges que provenen de l'ordenador a la mota emissora.

S'ha desenvolupat una aplicació amb Java amb la que es monitoritzen els valors dels sensors i les alarmes provinents de la mota emissora, també es poden modificar una sèrie de paràmetres com són: interval de temps amb el que es prenen valors dels sensors, interval de temps amb el que s'envien els valors i límit de temperatura amb el que la mota comprovarà l'alarma d'incendi així com, si fos el cas, enviar la confirmació d'alarma d'incendi.

Índex de continguts

Índex de figures

1. Introducció

Amb aquest Treball de Final de carrera es desenvoluparà un sistema contra incendis que consisteix a nivell de maquinari en:

- 1 mota cou24 emissora
- 1 mota cou24 receptora
- 1 ordenador.

A nivell de software consisteix a desenvolupar dues aplicacions:

Una aplicació en la mota emissora per poder gestionar els valors dels sensors de bateria i de temperatura per si fos el cas enviar les possibles alarmes (de bateria baixa o d'incendi) al receptor.

L'altre aplicació en l'ordenador per poder monitoritzar els valors i les alarmes provinents de la mota emissora així com poder canviar el comportament de la mateixa modificant els valors d'intervals de temps de lectura i enviament dels valors del sensors com el límit de la temperatura amb el que la mota controlarà l'alarma d'incendi.

En els pròxims punts es descriurà com s'ha desenvolupat i la seva posada en marxa.

1.1 Justificació

Aquest projecte està destinat a la **protecció contra incendis** en concret en els edificis ja que les motes proporcionades no estan preparades per situar-se a l'exterior.

Amb la protecció contra incendis es tracta de aconseguir tres fites:

- Salvar vides humanes
- Minimitzar les pèrdues econòmiques produïdes pel foc
- Aconseguir que les activitats de l'edifici puguin reanudar-se el més aviat possible.

Una notació important es que com més apropiats siguin els medis de protecció contra incendis més barates seran les pòlisses d'assegurança de l'edifici, en quan a incendi es refereixi, així com a aconseguir les fites indicades.

Els medis utilitzats per la protecció contra incendis es poden classificar en dos tipus:

- Mesures passives: Es tracta de les mesures que afecten a la construcció de l'edifici, fent servir materials ignífugs, facilitant l'evacuació dels usuaris presents en cas d'incendi mitjançant passadissos i escales suficientment amples.
- Mesures actives: Es divideixen en diferents tipus:
  - Destinades a la detecció: Detectors automàtics de fums o de calor així com de manuals (timbres que qualsevol pot prémer si hi ha un incendi)
  - Destinades a l'Alerta i Senyalització: Es dona avís als ocupants mitjançant timbres o megafonia, s'assenyalen les vies d'evacuació amb cartells de color verd normalment lluminosos. És important que els edificis disposin d'un sistema d'il·luminació mínim alimentat per bateries perquè si fos el cas de fallada dels sistemes d'il·luminació normals de l'edifici

permetessin arribar fins a la sortida. En molts casos també es interessant que els edificis puguin avisar als bombers en cas d'incendi de manera autònoma.

- Extinció: Mitjançant agents extintors (d'aigua, pols, espuma....), boques d'incendi.

- Pressurització d'escales: S'utilitzen ventiladors de gran caudal que generen una circulació d'aire des de la planta baixa de l'edifici fins als respiradors superiors, amb el que s'aconsegueix una evacuació dels fums més eficient.

Aquest projecte està dins de les mesures actives de detecció, amb el que s'ajuda a la detecció i indicació d'incendi en una part de l'edifici on s'instal·li la mota emissora.

1.2 Descripció del projecte

Aquest projecte està destinar al control, detecció i indicació d'alarmes d'incendis en el lloc on estiguin situades.

Compleix una sèrie de requisits:

- Controlar la temperatura d'on estigui situada i indicar-la al receptor així com l'estat de la seva bateria.

- Monitoritzar els valors dels sensors indicats per la mota emissora.

- Avisar en cas d'una alarma d'incendi així com d'alarma de bateria per indicar que se li ha de canviar la bateria ja que s'està esgotant.

- Prendre valors dels sensors de temperatura i de bateria cada N segons, aquest valor es modificable per part de l'usuari.

- Enviar els valors dels sensors de temperatura i de bateria cada M segons, aquest valor es modificable per part de l'usuari.

1.3. Objectius del TFC

1.3.1 Estudi del sistema operatiu Tinyos i les seves possibilitats

Es procedeix a buscar informació i documentació sobre el sistema operatiu ***Tinyos*** amb el que controlarem i realitzarem part del projecte.

La documentació així com la realització dels tutorials es troben en les rutes:

http://cv.uoc.es/app/mediawiki14/wiki/P%C3%A0gina_principal

http://tinyos.net/

1.3.2 Prendre valors dels sensors cada N segons

La mota emissora ha de ser capaç de prendre valors del sensor de temperatura i del sensor de bateria cada N segons (ms).

1.3.3 Enviar les mesures dels sensors cada M segons

La mota emissora ha de ser capaç d'enviar a la mota receptora els valors del sensor de temperatura i del sensor de bateria cada N segons (ms).

La mota receptora envia aquests valors a l'aplicació Java instal·lada en el PC per monitoritzar-los.

1.3.4 Control d'alarma de temperatura

La mota emissora ha de ser capaç de controlar quan es produeix una alarma d'incendi i notificar-la al receptor

El control d'alarmes té les següents tasques:

- Comprovar el nivell de temperatura cada N ms

- En cas d'alarma d'incendi enviar l'alarma cada segon al receptor

- Segui enviant les dades del sensors al receptor.

- Quan es rebi la confirmació d'alarma per part del receptor apagar el sistema.

1.3.5 Control d'alarma de bateria baixa

La mota emissora ha de ser capaç de controlar quan es produeix una alarma de bateria baixa i notificar-la al receptor

- Si la bateria baixa d'un nivell establert fixat en el codi la mota envia una alarma cada 60s de nivell de bateria baix.

- Comprovar el nivell de bateria cada M ms

- Seguir comprovant el sensor de temperatura per possibles alarmes d'incendi

- En cas d'alarma d'incendi enviar l'alarma al receptor.

1.3.5 Aplicació Java receptora

L'aplicació receptora en el PC ha de ser capaç de:

- Enviar els valors a modificar a la mota receptora per USB i aquesta per ràdio els envia a la mota emissora. Els valors a modificar son : l'interval de temps amb el que es prenen els valors dels sensors, l'interval de temps amb el que s'envien els valors dels sensors i límit de temperatura amb el que la mota emissora controla quan es produeix un incendi.

- Enviar la confirmació d'alarma d'incendi.

1.4. Enfocament i mètode seguit

Després de tenir clara la idea del projecte juntament amb els objectius a assolir es va prosseguir a elaborar un pla de treball en el que s'indiquen els objectius desglossats amb les seves corresponents tasques així com unes fites clares a seguir per poder finalitzar el projecte.

Al principi es va recaptar informació i estudiar sobre el sistema operatiu ***Tinyos*** amb el que es controlarà els diferents components de les motes:

Tinyos proveeix d'una sèrie de components ja programats per controlar els diferents components de les motes:

- Controlar l'encesa i parada dels sensors gràcies al component ***HplAtm128GeneralIOC***
- Prendre dades dels sensors gràcies al component ***AdcReadClientC()***
- Controlar l'encesa i parada de la ràdio gràcies al component ***ActiveMessageC***
- Enviar els valors dels sensors al receptor gràcies al component ***AMSenderC***
- Rebre els missatges provinents de l'aplicació Java (receptora) gràcies al component ***AMReceiverC.***
- Crear un iterador per poder prendre valors dels sensors cada N ms gràcies al component ***TimerMilliC.***
- Crear un iterador per poder enviar els valors dels sensors al receptor cada M ms gràcies al component ***TimerMilliC.***
- Crear un iterador per poder enviar l'alarma d'incendi (si fos el cas) cada segon gràcies al component ***TimerMilliC.***
- Crear un iterador per poder enviar l'alarma de bateria baixa (si fos el cas) cada 60 segons gràcies al component ***TimerMilliC.***
- Crear les constants necessàries així com les estructures per poder enviar els valors dels sensors, alarmes i rebre els valors a modificar i confirmació d'alarma d'incendi del receptor.

En la mota receptora es compila i s'instal·la una aplicació ja existent (BaseStation) amb la que aconseguim comunicació entre **mota emissora-PC** i **PC-mota emissora.**

En el PC es fa servir el SerialForwarder, que es una eina desenvolupada en Java per poder comunicar-nos des de Java amb la mota receptora que ens ajudarà a desenvolupar des de Java l'aplicació receptora, la comunicació en el nostre cas es farà mitjançant un port USB on està connectada la mota receptora.

A més Tinyos ens ofereix l'eina MIG (Message Interface Generator) amb la que aconseguim generar una classe Java per cada estructura, que nosaltres vulguem, creada en l'aplicació de la mota emissora i així poder rebre i interactuar des de Java amb aquestes estructures quan rebem un valor o també crear aquest tipus d'estructures des de Java per poder enviar-les a la mota emissora. També ens ofereix l'eina MoteIF amb la que podem enviar i rebre missatges a motes.

Després de desenvolupar una primera versió es procedeix a realitzar proves, amb els errors detectats es depura una segona versió que es la que s'ha entregat.

1.5. Planificació del projecte

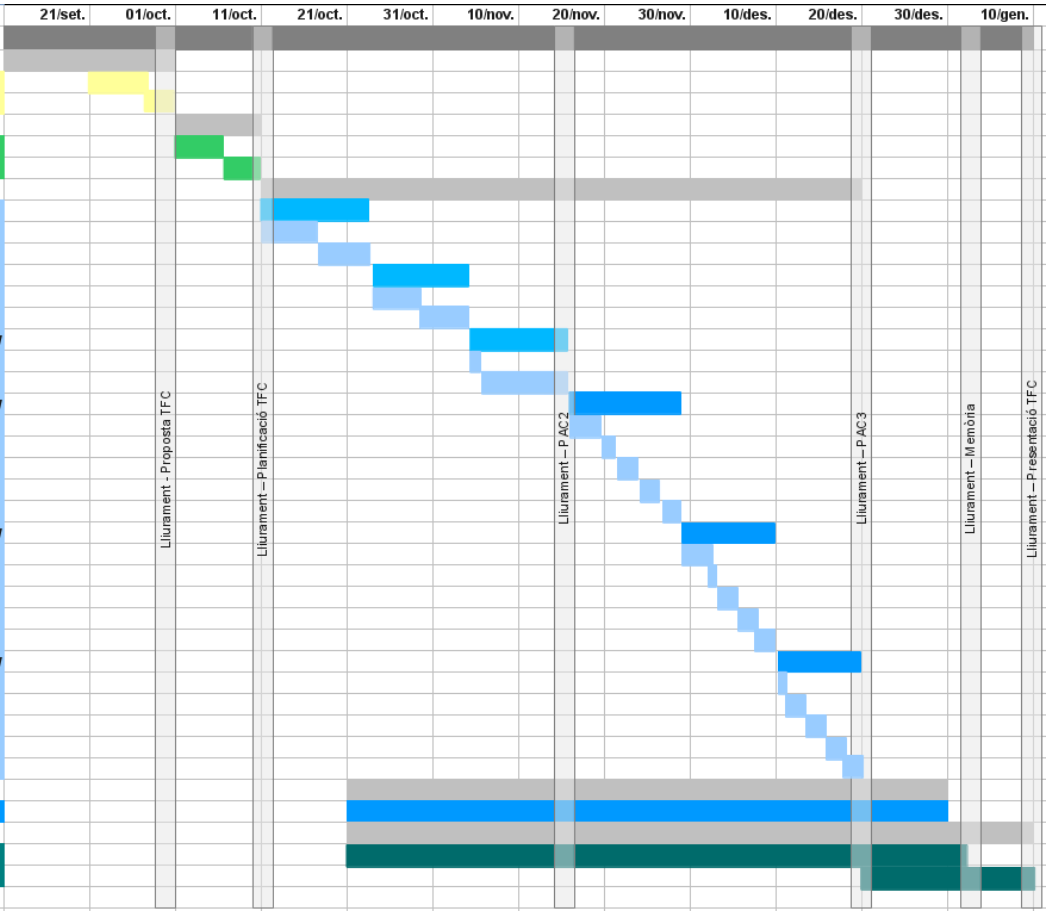
Una bona planificació és molt important per poder finalitzar el projecte amb garanties ja que disposem de molt poc temps “4 mesos”, per tant a l'inici es va realitzar un pla de treball indicant totes les pautes a seguir acotades per dates.

A continuació es mostra la planificació, juntament amb el diagrama de Gantt.

	Inici	Fi
-TFC Sistemes encastats	**21/09/2011**	**10/01/2012**
-Definició	**21/09/2011**	**30/09/2012**
-Estudi del projecte	21/09/2011	27/09/2011
-Elaboració de la proposta del projecte	28/09/2011	30/09/2011
-Planificació	**01/10/2011**	**11/10/2011**
-Estudi de la planificació	01/10/2011	08/10/2011
-Elaboració de la planificació del TFC	09/10/2011	11/10/2011
-Objectius i tasques	**12/10/2011**	**20/12/2011**
- Estudi del sistema operatiu Tinyos i les seves possibilitats	***12/10/2011***	***23/10/2011***
- Documentació i estudi sobre Tinyos	12/10/2011	17/10/2011
- Fer els tutorials de Tinyos	18/10/2011	23/10/2011
- Prendre valors dels sensors cada N segons	***24/10/2011***	***04/11/2011***
- Llegir el valor del sensor temperatura cada N ms	24/10/2011	29/10/2011
- Llegir el valor del sensor temperatura cada N ms	30/10/2011	04/11/2011
- Enviar les mesures dels sensors cada M segons	***05/11/2011***	***16/11/2011***
- Crear una estructura per poder desar les dades dels sensors	05/11/2011	15/11/2011
- Enviar les dades per ràdio cada M ms	16/11/2011	16/11/2011
- Control d'alarma de temperatura	***17/11/2011***	***28/11/2011***
- Comprovar el nivell de temperatura cada N ms	17/11/2011	19/11/2011
- Crear una estructura per poder enviar i confirmar l'alarma d'incendi	20/11/2011	21/11/2011
- En cas d'alarma d'incendi enviar l'alarma cada segon al receptor	22/11/2011	24/11/2011
- Segui enviant les dades dels sensors al receptor	25/11/2011	26/11/2011
- Quan es rebi la confirmació d'alarma per part del receptor apagar el sistema	27/11/2011	28/11/2011
- Control d'alarma de bateria baixa	***29/11/2011***	***10/12/2011***
- Comprovar el nivell de bateria cada M ms	29/11/2011	01/12/2011
- Crear una estructura per poder enviar l'alarma de bateria baixa	02/12/2011	02/12/2011
- En cas d'alarma de bateria enviar l'alarma cada 60 s	03/12/2011	05/12/2011
- Seguir comprovant el sensor de temperatura per possibles alarmes d'incendi	06/12/2011	08/12/2011
- En cas d'alarma d'incendi enviar l'alarma al receptor.	09/12/2011	10/12/2011
- Aplicació Java des de la que l'usuari pot modificar N, M i limit de temperatura	***11/12/2011***	***20/12/2011***
- Crear les classes des de les estructures	11/12/2011	11/12/2011
- Desenvolupar una interfície per poder monitoritzar i modificar els valors N,M i limit de temperatura	12/12/2011	14/12/2011
- Desenvolupar l'enviament de valors a modificar a la mota emissora	15/12/2011	16/12/2011
- Desenvolupar l'enviament de la confirmació d'alarma d'incendi	17/12/2011	18/12/2011
- Desenvolupar la monitorització d'alarmes per pantalla	19/12/2011	20/12/2011
-Proves	**21/10/2011**	**30/12/2011**
-Realització de proves	21/10/2011	30/12/2011
-Lliurament	**21/10/2011**	**10/01/2012**
-Elaboració de la memòria	21/10/2011	03/01/2012
-Elaboració de la presentació	20/12/2011	10/01/2012

1.5.1 Planificació del projecte

	Inici	Fi
-TFC Sistemes encastats	21/09/2011	10/01/2012
-Definició	21/09/2011	30/09/2012
-Estudi del projecte	21/09/2011	27/09/2011
-Elaboració de la proposta del projecte	28/09/2011	30/09/2011
-Planificació	01/10/2011	11/10/2011
-Estudi de la planificació	01/10/2011	08/10/2011
-Elaboració de la planificació del TFC	09/10/2011	11/10/2011
-Objectius i tasques	12/10/2011	20/12/2011
- Estudi del sistema operatiu Tinyos i les seves possibilitats	12/10/2011	23/10/2011
- Documentació i estudi sobre Tinyos	12/10/2011	17/10/2011
- Fer els tutorials de Tinyos	18/10/2011	23/10/2011
- Prendre valors dels sensors cada N segons	24/10/2011	04/11/2011
- Llegir el valor del sensor temperatura cada N ms	24/10/2011	29/10/2011
- Llegir el valor del sensor temperatura cada N ms	30/10/2011	04/11/2011
- Enviar les mesures dels sensors cada M segons	05/11/2011	16/11/2011
- Crear una estructura per poder desar les dades dels sensors	05/11/2011	05/11/2011
- Enviar les dades per ràdio cada M ms	06/11/2011	16/11/2011
- Control d'alarma de temperatura	17/11/2011	28/11/2011
- Comprovar el nivell de temperatura cada N ms	17/11/2011	19/11/2011
- Crear una estructura per poder enviar i confirmar l'alarma d'incendi	20/11/2011	21/11/2011
- En cas d'alarma d'incendi enviar l'alarma cada segon al receptor	22/11/2011	24/11/2011
- Segui enviant les dades dels sensors al receptor	25/11/2011	26/11/2011
- Quan es rebi la confirmació d'alarma per part del receptor apagar el sistema	27/11/2011	28/11/2011
- Control d'alarma de bateria baixa	29/11/2011	10/12/2011
- Comprovar el nivell de bateria cada M ms	29/11/2011	01/12/2011
- Crear una estructura per poder enviar l'alarma de bateria baixa	02/12/2011	02/12/2011
- En cas d'alarma de bateria enviar l'alarma cada 60 s	03/12/2011	05/12/2011
- Seguir comprovant el sensor de temperatura per possibles alarmes d'incendi	06/12/2011	08/12/2011
- En cas d'alarma d'incendi enviar l'alarma al receptor.	09/12/2011	10/12/2011
- Aplicació Java des de la que l'usuari pot modificar N, M i limit de temperatura	11/12/2011	20/12/2011
- Crear les classes des de les estructures	11/12/2011	11/12/2011
- Desenvolupar una interfície per poder monitoritzar i modificar els valors N,M i limit de temper	12/12/2011	14/12/2011
- Desenvolupar l'enviament de valors a modificar a la mota emissora	15/12/2011	16/12/2011
- Desenvolupar l'enviament de la confirmació d'alarma d'incendi	17/12/2011	18/12/2011
- Desenvolupar la monitorització d'alarmes per pantalla	19/12/2011	20/12/2011
-Proves	21/10/2011	30/12/2011
-Realització de proves	21/10/2011	30/12/2011
-Lliurament	21/10/2011	10/01/2012
-Elaboració de la memòria	21/10/2011	03/01/2012
-Elaboració de la presentació	20/12/2011	10/01/2012



1.5.2 Diagrama de Gantt

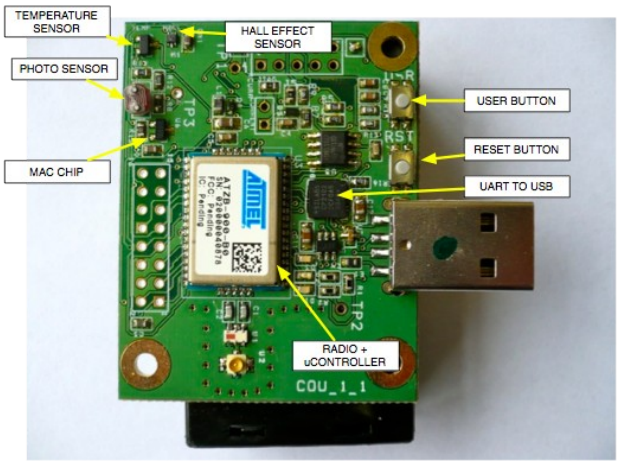
1.6. Recursos emprats

En aquest apartat s'enumeren i es descriuen el materials utilitzats tant de maquinari com de programari.

1.6.1 Recursos de maquinari

- Dues motes cou_1_2 24A2, una es l'encarregada de l'emissió dels valors dels sensors així com controlar i gestionar les possibles alarmes tant d'incendi com de bateria baixa.

I l'altre es l'encarregada de realitzar les tasques de repetidor, tot el que li arriba per USB o emet per ràdio i tot el que li arriba per ràdio o envia per USB, gràcies a l'aplicació BaseStation.



1.6.1.1 Mota Cou_1_2 24A2

Aquesta mota disposa d'uns components que descrivim a continuació:

- Microcontrolador de la casa Atmel: ATmega1281, es un microcontrolador de 8 bits amb 128 KB de memòria Flash programable, la taula d'especificacions es la següent:

Device	Flash	EEPROM	RAM	General Purpose I/O pins	16 bits resolution PWM channels	Serial USARTs	ADC Channels
ATmega640	64KB	4KB	8KB	86	12	4	16
ATmega1280	128KB	4KB	8KB	86	12	4	16
ATmega1281	128KB	4KB	8KB	54	6	2	8
ATmega2560	256KB	4KB	8KB	86	12	4	16
ATmega2561	256KB	4KB	8KB	54	6	2	8

1.6.1.2 Taula d'especificacions Atm1281

- 4KB de memòria EEPROM (*Enhanced-Erasable Programmable Read-Only Memory*), és un tipus de memòria ROM (*Read-only-memory*) que es pot programar, esborrar i reprogramar elèctricament, es pot llegir un nombre il·limitat de vegades però només es pot esborrar i reprogramar entre 100.000 i un milió de vegades.

- 8KB de memòria RAM (*Random-access memory*) és la memòria des don el processador rep les instruccions i desa els resultats.

- 54 pins d'entrada/Sortida amb els que es comunica amb els perifèrics.

- 16 bits de resolució per controlar el PWM (Pulse With Modulation), es una tècnica en la qual es modifica el cicle de treball d'un senyal periòdic normalment per enviar informació a través d'un canal de comunicacions.

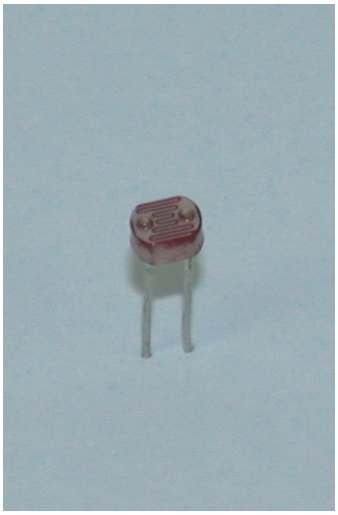
- 2 Transmissors-Receptors Asíncrons Universals, controlen les interrupcions dels dispositius connectats a ells, en el nostre cas pel port USB.

- 8 canals destinats al convertidor analògic digital (ADC), s'encarrega de convertir un senyal analògic (voltatge) a un senyal digital (binari).

Aquest microcontrolador està dissenyat per optimitzar el consum d'energia contra la velocitat de processament.

-Transceptor AT86RF230: A la banda de 2,4GHz compatible amb els protocols 802.15.4/ZigBee. Fins a 3dBm de potencia de sortida i una sensibilitat de recepció de fins a -101dBm.

- Sensor de llum de la casa API: (PDV-P9003-1):



1.6.1.3 Sensor de llum (PDV-P9003-1)

Esta dissenyat per detectar llum d'entre 400 a 700 nm de longitud d'ona.

- Sensor de temperatura de la casa Microchip: Al igual que el microcontrolador està dissenyat per optimitzar el consum d'energia, molt important en aquest tipus de components.

És un sensor de temperatura analògic de baix cost capaç de mesurar temperatures d'entre 0ºC a 70ºC amb un consum de 6mA.

- Sensor d'efecte Hall de la casa ROHM: També està dissenyat per optimitzar el consum d'energia, és un interruptor magnètic que el podríem fer servir per detectar l'obertura i tancament de portes, un exemple d'aquest us seria l'obertura i el tancament de les comportes d'un hivernacle.

-Tres díodes LED: per utilitzar-los com indicadors.

- Botó de reset: Serveix per reiniciar la mota i també per enviar la senyal d'interrupció necessària per carregar els programes compilats a la mota mitjançant l'aplicació ***meshprog.***

- Botó d'usuari: En cas d'haver-ho programat, compilar i carregat a la mota, amb aquest botó es podria interactuar amb la mota.

-Component Uart to USB: Serveix per traduir les interrupcions provinents de l'USB per on es connecta al PC.

-MAC chip: Media Access Controler amb el que controlarà la ràdio.

- Cable USB femella/mascle per connectar més còmodament la mota al PC:

1.6.1.4 Cable USB femella/mascle

- Dues piles AA per alimentar la mota emissora:

1.6.1.5 Piles AA

- PC de sobretaula:

PC	
Processador	Intel Core i7 2600K
Dissipador	Prolimatech Black Series Megahalems
Ventiladores	Noiseblocker BlackSilentPRO PLPS 120x120x25
	Noiseblocker BlackSilentPRO PLPS 120x120x25
Placa Base	Asus Sabertooth P67 B3
Targeta de Vídeo	MSI GeForce GTX580 Lightning 1536MB GDDR5 - VGA
RAM	Corsair Vengeance DDR3 1600 8GB CL9
Fuente	Seasonic X-760W Gold Modular 760W
Caixa	Silverstone Raven 3 RV03B
SSD	Crucial M4 128GB SATA3 MAESTRO Disco Duro SSD
HD	WD Caviar Black 1TB SATA3 64MB MAESTRO
Gravadora	Regrabadoras DVD Serial ATA ASUS DRW-24B3ST
Ctrl. Vent.	Scythe Kaze Master Ace 5.25" Frontal Multifunción

1.6.1.6 PC de sobretaula

1.6.2 Recursos de programari

- Sistema Operatiu del PC (Arch Linux x86_64)

1.6.2.1 Sistema Operatiu del PC

- TinyOS 2.1.1: Tiny és un sistema operatiu de codi obert amb llicència BSD dissenyat per a dispositius sense fils de baixa potència, com els utilitzats en xarxes de sensors, xarxes d'àrea personal, en els edificis intel·ligents i els mesuradors intel·ligents.

Una comunitat mundial d'estudi i indústria fa servir, desenvolupa i donar suport al sistema operatiu, així com les seves eines associades, amb una mitjana 35.000 descàrregues a l'any.

- Eclipse Indigo versió: 3.7.1: Eclipse és un entorn multiplataforma de desenvolupament programat principalment en Java, per a desenvolupar projectes en C, C++, COBOL, Python, Perl, PHP, i molts altres, sempre i quan s'instal·lin els connectors corresponents per a cada llenguatge de programació, cal afegir que és de codi lliure.

Fou desenvolupat originalment per IBM com el successor de la seva família d'eines per a VisualAge. Tot i això, actualment Eclipse és desenvolupat per la Fundació Eclipse, una

organització independent sense ànim de lucre que fomenta una comunitat de programari lliure i un conjunt de productes complementaris, capacitats i serveis.

- Yeti 2 – Plugin de TinyOS per Eclipse: Plugin per Eclipse, proporciona un editor per al llenguatge NesC i l'etorn TinyOS també proveeix a l'usuari la detecció d'errors, hipervincles, completat de codi i altres característiques útils.

Ha estat desenvolupat pel grup de recerca “Distributed Computing Group” de l'institut federal de tecnologia de Zurich (ETH) .

- Meshprog (Meshnetics serial programmer for Linux): Es una alternativa a interfícies dedicades com (ISP, JTAG) però només serveix per poder carregar els programes compilats a les motes a través de l'USB.

L'utilització és senzilla, s'ha de fer la crida:

meshprog -t <port usb on hi ha connectada la mota> -f <programa a carregar a la mota>.

Exemple:

meshprog -t /dev/ttyUSB0 -f ./build/cou24/main.srec

Un cop haguem fet la crida ***meshprog*** esperarà una senyal de la mota per enviar el programa, aquest senyal en el nostre cas es prémer el botó de reset.

- SerialForwarder: Aplicació desenvolupada amb Java que s'encarrega de recollir paquets que arriben pel port sèrie o USB i els reenvia a un socket TCP, gràcies a ella ens podrem comunicar mitjançant sockets amb la mota que estigui connectada al PC.

- NesC (Network Embedded Systems C): És un llenguatge de programació C optimitzat a les limitacions de memòria de las xarxes de sensors. Existeixen varies aplicacions que completen i faciliten el seu us desenvolupades la seva majoria en Java (Yeti 2) i en Bash. Altres eines i llibreries associades estan escrites principalment en C.

Està orientat a components i especialment dissenyat per desenvolupar aplicacions sobre xarxes de sensors, en particular amb el sistema operatiu TinyOS.

Una aplicació en NesC està estructurada mitjançant components, es poden crear components utilitzant a la vegada altres components ja creats. Dos components podran comunicar-se entre si mitjançant una interfície, la qual definirà una sèrie de mètodes (commands i events) els quals deuran ser implementats en cada component. Així, un mètode podrà sol·licitar l'execució d'un command d'un altre component; per altre banda, per enviar una notificació s'utilitzarà un event.

Tot component està dividit lògicament en tres partes: Configuració, Implementació i Mòdul.

La Configuració es coneguda como "wiring", que és una declaració de las interfícies utilitzades i que a la vegada són proporcionades por un component. Ex:

```
implementation {
        components MainC;
        components a as App;
        App.Boot -> MainC;
        }
```

Els Mòduls són el que coneixem com implementació, on es programaran les acciones que executarà el nostre component. Els mòduls estan dividits en tres partes: ***Provides***, són les interfícies que el nostre component ofereix, ***Uses***, les interfícies que fa servir el nostre component; ***Implementation*** on es realitzen les accions que volem que realitzi el nostre programa.

- UMLet: Aplicació de codi obert amb la que s'han realitzat diferents diagrames per explicar el funcionament del sistema.

1.7. Productes obtinguts

En aquest apartat es descriuen els dos principals productes obtinguts en aquest treball de final de carrera, també són productes associats a aquest projecte la planificació que es va fer a l'inici del projecte i també aquesta memòria.

1.7.1 Programari mota emissora

- Projecte *FireProtectionSystem:* Projecte desenvolupat en NesC amb el sistema operatiu TinyOS amb el que es gestiona el sistema contra incendis en la mota emissora. Controla les possibles alarmes de temperatura i bateria baixa així com enviar els valors dels sensors a la mota receptora.

1.7.2 Programari PC receptor

- Projecte *FireProtectionSystemJava:* Aquesta aplicació està desenvolupada amb Java i gràcies als components de TinyOS per Java MoteIF i MIG i amb l'ajuda de l'aplicació SerialForwarder és capaç de comunicar-se amb la mota connectada al PC per poder fer el següent:

- Monitoritzar els valors provinents de la mota emissora
- Enviar a la mota emissora el nou valor d'interval de temps de lectura dels sensors, en ms
- Enviar a la mota emissora el nou valor d'interval de temps d'enviament dels valors dels sensors, en ms
- Enviar a la mota emissora el nou valor del límit de temperatura abans de que es produeixi l'alarma d'incendi.
- Quan es rebi una alarma tant sigui d'incendi com de bateria baixa indicar-ho per pantalla.
- Enviar a la mota emissora la confirmació d'alarma d'incendi.

1.8. Breu descripció dels altres capítols de la memòria.

En els següents apartats es procedeix a descriure diferents aspectes del projecte:

Apartat 2. Antecedents: Es detalla quin és, actualment, l'estat de la tecnologia, NesC, TinyOS, motes que hi ha al mercat, protocols de comunicació per a xarxes de sensors sense fils.
S'estudia quines aplicacions hi ha semblants en el mercat, el sectors als que va dirigida.

Apartat 3. Descripció funcional: Es descriu el disseny del nostre sistema de les dues aplicacions desenvolupades: *FireprotectionSystem* i *FireProtectionSystemJava,* les decisions que s'han hagut de prendre al desenvolupar-la. Ens ajudarem de diferents tipus d'esquemes.

Apartat 4. Descripció detallada: S'explica de forma detallada i tècnica del sistema tant per part de l'aplicació *FireprotectionSystem* com de la *FireProtectionSystemJava.* En aquesta descripció no hi haurà codi font de l'aplicació però si que es detallarà fent servir diagrames, diagrama d'execució, de casos d'us, diagrama de flux, estructures utilitzades.

Apartat 5. Viabilitat tècnica: Es descriuen les tasques realitzades tant per part de desenvolupament com per part de les proves realitzades per a que el projecte funcioni correctament.

Apartat 6. Valoració econòmica: Es realitza un estudi de quin és el cost del projecte per part del desenvolupament i estudi.

Apartat 7. Conclusions: S'exposen les conclusions a les que s'ha arribat al desenvolupar aquest projecte, objectius aconseguits sobre la planificació inicial, propostes de millora.

Apartat 8. Glossari: El glossari del projecte.

Apartat 9. Bibliografia: Fonts d'informació utilitzades per a realitzar el projecte.

Apartat 10. Annexos: En els annexos hi trobarem el manual d'execució i compilació.

2. Antecedents

2.1. Estat de l'art

2.1.1 Xarxes de sensors sense fils

Una xarxa de sensors sense fils (*WSN wireless sensor network*) consisteix a distribuir nodes autònoms que disposen d'una sèrie de sensors de baix consum amb els que es poden monitoritzar diferents condicions físiques o mediambientals, (temperatura, so, vibració, pressió de l'aire, etc..). Aquestes dades s'envien de forma cooperativa mitjançant la xarxa al node principal que es l'encarregat de rebre-les i comunicar-les normalment a un PC.

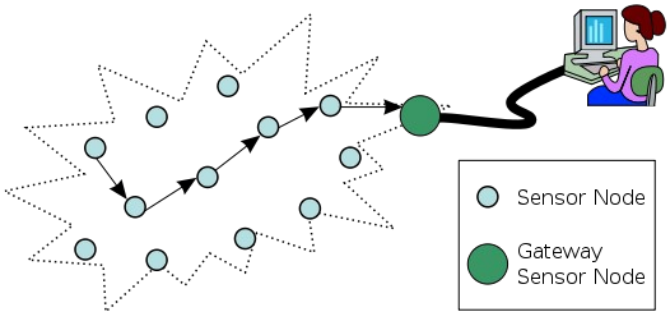


Figura 2.1.1.1 Exemple de WSN

Es caracteritzen per la seva facilitat de desplegament i per ser autoconfigurables, poden convertir-se en tot moment en emissor, receptor, oferir serveis d'encaminament entre nodes sense visió directa, així com registrar dades referents als sensors locals de cada node, una altre de les seves característiques es la seva gestió eficient de l'energia que les permet obtenir una tassa d'autonomia elevada.

La investigació d'aquestes xarxes va començar prop de l'any 1980 amb el projecte DSN (Distributed Sensor Networks) de l'agència militar d'investigació dels Estats Units, com a predecessor d'aquestes xarxes es pot considerar el projecte *(SOSUS) Sound Surveillance System*, es tracta d'una xarxa de boies submergides instal·lades als Estats Units durant la guerra freda per detectar submarins fent servir sensors de so.

Avui dia les xarxes de sensors es un tema molt actiu d'investigació en varies universitats, encara que ja comencen a existir aplicacions comercials basades en aquest tipus de xarxes.

Exposem els seus usos en l'àmbit civil:

- Eficiència energètica: Xarxa de sensors que s'utilitza per controlar l'us eficaç de l'electricitat.
- Entorns d'alta seguretat: Existeixen llocs que requereixen alts nivells de seguretat com per exemple centrals nuclears, aeroports, edificis del Gobern de pas restringit. Gràcies a les xarxes de sensors es possible detectar situacions que amb una simple càmera seria

impossible.

- Sensors ambientals: Es poden controlar múltiples variables com ara la temperatura, humitat, foc, activitat sísmica. Per exemple es pot instal·lar una xarxa de sensors en un volcà perquè gràcies a les diferents mesures dels sensors els científics puguin predir quan es pot produir una erupció, o també instal·lar aquest tipus de xarxa en una zona on es susceptible de originar-se terratrèmols perquè els científics puguin predir-los amb antelació i així poder evacuar a la gent afectada salvant milers de vides.
- Àmbit industrials: Hi ha fàbriques que tenen controls de qualitat molt complexes en els que aquestes xarxes hi tenen us.
- Automoció: Les xarxes de sensors son el complement ideal a les càmeres de tràfic ja que poden informa del tràfic en angles morts que no cobreixen i també poden informar als conductors de la situació en cas d'embús o accident i així poder escollir una ruta alternativa.
- Medicina: És un camp molt prometedor ja que es poden fer servir perquè els pacients que ho necessitin puguin tenir controlades les constants vitals i així millorar la qualitat de vida.
- Domòtica: El seu abast, economia i velocitat de desplegament fan d'aquest tipus de xarxa una opció molt vàlida per domotitzar la casa.

2.1.2 Motes

Una mota és un node d'una xarxa de sensors sense fil que és capaç de realitzar algun tipus de processament, recopilació de la informació sensorial i comunicar-se amb altres nodes connectats a la xarxa.

Disposen de diferents sensors per poder realitzar mesures (temperatura, moviment, pressió,....).

Diferents tipus de motes que existeixen actualment en el mercat amb les seves característiques:

Sensor Node Name	Microcontroller	Tranceiver	Program+Data Memory	External Memory	Programming	Remarks
Arago Systems WiSMote Dev	MSP430F5437	CC2520	RAM : 16 Kbytes Flash : 256 Kbytes	up to 8 Mbits	C	Contiki and 6LoWPan supported
Arago Systems WiSMote Mini	ATMEGA128 RFA2	ATMEGA 128 RFA2	RAM : 16 Kbytes Flash : 128 Kbytes E²PROM : 4Kbytes		C	Contiki and 6LoWPan supported
AVRraven Atmel AVR#Raven wireless kit	AtMega1284p + ATmega3290p	AT86RF230	128 Kbytes + 16 Kbytes	256 kB?	C	
COOKIES	ADUC841, MSP430	ETRX2 TELEGESIS, ZigBit 868/915	4 Kbytes + 62 Kbytes	4 Mbit	C	Platform with hardware reconfigurability (Spartan 3FPGA based or Actel Igloo)

BEAN	MSP430F169	CC1000 (300-1000 MHz) with 78.6 kbit/s		4 Mbit		YATOS Support
BTnode	Atmel ATmega 128L (8 MHz @ 8 MIPS)	Chipcon CC1000 (433-915 MHz) and Bluetooth (2.4 GHz)	64+180 K RAM	128K FLASH ROM, 4K EEPROM	C and nesC Programming	BTnut and TinyOS support
COTS	ATMEL Microcontroller 916 MHz					
Dot	ATMEGA163		1K RAM	8-16K Flash	weC	
EPIC mote	Texas Instruments MSP430 microcontroller	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash		TinyOS
Egs[1]	ARM Cortex M3	CC2520, Mitsumi's class 2 Bluetooth module		2 Gbit		TinyOS
Eyes	MSP430F149	TR1001		8 Mbit		PeerOS Support
EyesIFX v1	MSP430F149	TDA5250 (868 MHz) FSK		8 Mbit		TinyOS Support
EyesIFX v2	MSP430F1611	TDA5250 (868 MHz) FSK		8 Mbit		TinyOS Support
FlatMesh FM1	16 MHz	802.15.4-compliant		660 sensor readings	Over-air control	Commercial system, for digital sensors
FlatMesh FM2	16 MHz	802.15.4-compliant		660 sensor readings	Over-air control	Commercial system, built-in tilt sensor
GWnode	PIC18LF8722	BiM (173 MHz) FSK	64k RAM	128k flash	C	Custom OS
IMote	ARM core 12 MHz	Bluetooth with the range of 30 m	64K SRAM	512K Flash		TinyOS Support
IMote 1.0	ARM 7TDMI 12-48 MHz	Bluetooth with the range of 30 m	64K SRAM	512K Flash		TinyOS Support
IMote 2.0	Marvell PXA271 ARM 11-400 MHz	TI CC2420 802.15.4/ ZigBee compliant	32 MB SRAM	32 MB Flash		Microsoft .NET Micro, Linux, TinyOS Support

		radio				
INDriya_CS_03A14 [2]	Atmel ATmega 128L[3]	IEEE 802.15.4 compliant XBee radios	128 KB FLASH + 4 KB RAM	Expansion available	C-programming & nesC compliant	Comprehensive Sensor mote with: Ambient Light,temperature,accelerometer,JPEG camera,PIR, sound sensor, TinyOS TinyOS compliant, IPv6 network supportive stacks for internetworking & so on.
Iris Mote	ATmega 1281	Atmel AT86RF230 802.15.4/ ZigBee compliant radio	8K RAM	128K Flash	nesC	TinyOS, MoteWorks Support
KMote	TI MSP430	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash		TinyOS and SOS Support
Mica	ATmega 103 4 MHz 8-bit CPU	RFM TR1000 radio 50 kbit/s	128+4K RAM	512K Flash	nesC Programming	TinyOS Support
Mica2	ATMEGA 128L	Chipcon 868/916 MHz	4K RAM	128K Flash		TinyOS, SOS and MantisOS Support
Mica2Dot	ATMEGA 128		4K RAM	128K Flash		
MicaZ	ATMEGA 128	TI CC2420 802.15.4/ ZigBee compliant radio	4K RAM	128K Flash	nesC	TinyOS, SOS, MantisOS and Nano-RK Support
Monnit WIT	TI CC1110	868/900 MHz	4K RAM		C#	multi sensor boards
Mulle	Renesas M16C	Atmel AT86RF230 802.15.4 / Bluetooth 2.0	31K RAM	384K+4K Flash, 2 MB EEPROM	nesC, C programming	Contiki, TinyOS, lwIP: TCP/IP and Bluetooth Profiles: LAP, DUN, PAN and SPP Support
NeoMote	ATmega 128L	TI CC2420 802.15.4/ ZigBee compliant radio	4K RAM	128K Flash	nesC	TinyOS, SOS, MantisOS, Nano-RK and Xmesh Support, Industrial end-use product.
Nymph	ATMEGA128L	CC1000		64 kB EEPROM		MantisOS Support
PowWow	MSP430F161	TI	55kB Flash + 5kB		C	Open source;

	2	CC2420 802.15.4/ ZigBee compliant radio	RAM		programming : MSPGCC, IAR	open hardware; research platform
Redbee	MC13224V	2.4 GHz 802.15.4	96 KB RAM + 120 KB Flash		GCC (see mc1322x.devl.org), IAR	Contiki; standalone
Rene	ATMEL8535	916 MHz radio with bandwidth of 10 kbit/s	512 bytes RAM	8K Flash		TinyOS Support
SenseNode	MSP430F1611	Chipcon CC2420	10K RAM	48K Flash	C and NesC programming	GenOS and TinyOS Support
Shimmer	MSP430F1611	802.15.4 Shimmer SR7 (TI CC2420)	48 KB Flash 10 KB RAM	2 GB microSD Card	nes C and C Programming	TinyOS Support. Built in 3 Axis Accel, Tilt/Vib Sensor. Full range of expansion modules.
SunSPOT	ARM 920T	802.15.4	512K RAM	4 MB Flash	Java	Squawk Java ME Virtual Machine
Telos	MSP430		2K RAM			
TelosB	Texas Instruments MSP430 microcontroller	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash		Contiki, TinyOS, SOS and MantisOS Support
Tinynode	Texas Instruments MSP430 microcontroller	Semtech SX1211	8K RAM	512K Flash	C Programming	TinyOS
T-Mote Sky	Texas Instruments MSP430 microcontroller	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash		Contiki, TinyOS, SOS and MantisOS Support
Waspmote	Atmel ATmega 1281	ZigBee/802.15.4/DigiMesh/RF , 2.4 GHz/ 868/900 MHz	8K SRAM	128K FLASH ROM, 4K EEPROM, 2 GB SD card	C/Processing	GPRS, Bluetooth, GPS modules, sensor boards..
weC	Atmel AVR AT90S2313	RFM TR1000 RF				
Wireless	Atmega 128L	Chipcon	4k RAM	128k Flash		Xmesh, TinyOS

RS485		CC2420 + Amplifier 250 kbit/s 2.4 GHz IEEE 802.15.4				
XYZ	ML67 series ARM/THUMB microcontroller	CC2420 Zigbee compliant radio from Chipcon	32K RAM	256K Flash	C Programming	SOS Operating System Support
Zolertia Z1	Texas Instruments MSP430F2617	Chipcon CC2420 2.4 GHz IEEE 802.15.4 Wireless Transceiver	8 KB RAM	92 KB Flash	C, nesC	Contiki and TinyOS Support. 16 Mbit external flash + 2 digital on-board sensors
FireFly	Atmel ATmega 1281	Chipcon CC2420	8K RAM	128K FLASH ROM, 4K EEPROM	C Programming	Nano-RK RTOS Support
Ubimote1	TI's CC2430 SOC based on 8051 Core	TI's CC2430	8K RAM	128K FLASH ROM	C Programming	TI's ZStack, TinyOS Support
Ubimote2	TI's MSP430F2618	TI's CC2520	8K RAM	116K FLASH ROM	C Programming	TI's ZStack Support
VEmesh	TI MSP430	Semtech SX1211/1231, TI TRF6903	512B RAM	8K FLASH	Over-the-air Programming	FHSS; Interface to MODBUS, DALI, RS-232/485, TCP/IP

2.1.3 Sistemes operatius per a motes

En l'actualitat hi ha varietat de sistemes operatius per a motes que descrivim a continuació:

Contiki: Ha estat dissenyat per cobrir les necessitats dels sistemes encastats amb poca memòria. Consisteix en un nucli orientat a esdeveniments el qual fa us de processos sobre els quals els programes son carregats i descarregats dinàmicament, es comunica entre processos enviant missatges a través d'esdeveniments.
Pot funcionar en una varietat de plataformes, des de microcontroladors encastats com el MSP430 i el AVR com a Pcs.

Erika Enterprise: Es un sistema operatiu de codi obert, inclou RT-Druid el qual es un entorn de desenvolupament distribuït com un conjunt de plugins d'Eclipse.

La idea principal és poder obtenir un sistema bàsic i usable per aplicacions en l'àmbit de l'automoció així com per a sistemes encastats que requereixin d'un suport de temps real, és ideal per a xarxes de sensors.

Nano-RK: És un (RTOS) Sistema operatiu de temps real desenvolupat a la Universitat Carnegie Mellon i dissenyat per se executat en microcontroladors de xarxes de sensors.

Suporta tasques de prioritat fixa, Nano significa que consumeix pocs recursos, i RK (Resource Kernel) indica al sistema com ha de consumir els recursos, està escrit en C.

TinyOS: És un sistema operatiu de codi obert basat en components per a xarxes de sensors sense fils, escrit amb el llenguatge NesC com un conjunt de tasques i processos que col·laboren entre si, dissenyat per incorporar novetats ràpidament i per funcionar sota les importants restriccions de memòria que es donen en xarxes de sensors.
Està desenvolupat per un consorci liderat per la Universitat de Califòrnia en Berkley amb cooperació amb *Intel Research*. Les aplicacions per Tinyos s'escriuen en NesC, un dialecte del llenguatge C.
Proporciona interfícies, mòduls i configuracions específiques que permeten als programadors construir programes amb una sèrie de mòduls que fan tasques específiques. Els mòduls de TinyOS ofereixen interfícies pels tipus estàndard d'entrades i sortides de maquinari i sensors.

Com ja s'ha comentat prèviament en aquest projecte s'ha fet servir el sistema operatiu TinyOS.

LiteOS: Es un sistema operatiu de temps real (RTOS) desenvolupat per l'Universitat d'Illinois, s'ajusta als nodes de sensors de memòria limitada, permet als usuaris operar xarxes de sensors sense fils com ho fa Unix, proporciona un entorn de programació familiar basat en Unix i C, és de codi obert i està escrit en C.

2.1.4 Estàndards de comunicació per xarxes de sensors sense fils

Els estàndards de comunicació predominants en les xarxes de sensors sense fils son els següents:

WirelessHART: És una tecnologia de xarxa de tipus malla que opera a la banda de ràdio ISM (Industria, Científica i Mèdica) de 2,4GHz. Utilitza ràdios compatibles amb l'estàndard IEEE 802.15.4, modulació DSSS i salts en freqüència basats en enviaments de paquet per paquet, la topologia està dissenyada per a que operi com a tipus malla però també pot funcionar amb topologies tipus estrella o ramificada.

ISA100.11a: Té per objectiu proporcionar comunicació sense fils confiable i segura per aplicacions de monitorització no crítica, alerta, control de supervisió.
El comitè d'estàndards ISA100 va ser establert pe la ISA (Societat Internacional d'Automatització) per fer front a les demandes en el sentit d'establir un marc regulador per les solucions de comunicació sense files en àrees que inclouen:
- L'entorn en el que se desplega la tecnologia sense fils
- La tecnologia i cicle de vida dels equips i sistemes sense fils
- L'aplicació de la tecnologia sense fils.

El propòsit dels estàndards ISA100 es millorar la confiança i la disponibilitat dels components o sistemes utilitzats per la fabricació o control i establir aquells criteris per l'adquisició i la aplicació de la tecnologia sense fils en l'entorn del sistema de control. També es satisfan els requisits per evitar les interferències que es poguessin trobar en els ambients industrials i amb sistemes sense fils existents no compatibles amb l'estàndard ISA100.11a.

IEEE 1451: És un grup de normes de caràcter universal i obert per estandarditzar sensors intel·ligents tenint en compte tres característiques fonamentals: Integració de la intel·ligència propera al punt de mesura, capacitat de còmput i interfícies digitals estàndards per la transferència de dades, va ser creat per l'institut d'Enginyers Elèctric Electrònics amb la finalitat de poder definir un estàndard de comunicació per les xarxes de sensors intel·ligents i inclús possibilitar la configuració de sensors "Plug&Play" sense importar la tecnologia o forma de comunicació.

ZigBee: És el nom de l'especificació d'un conjunt de protocols d'alt nivell de comunicacions sense fils per la seva utilització en radiodifusió digital de baix consum, basada en l'estàndard IEEE 802.15.4 de xarxes sense fils d'àrea personal (WPAN).
El seu objectiu son les aplicacions que requereixen comunicacions segures amb baixa taxa d'enviament de dades i maximització de la vida útil de les seves bateries.
En principi, l'àmbit on es preveu que aquesta tecnologia tingui més força es en domòtica ja que té les següents característiques:
- Baix consum
- Topologia de xarxa en malla
- La seva fàcil integració (es poden fabricar nodes amb molt poca electrònica).

IEEE 802.15.4: És un estàndard que defineix el nivell físic i el control d'accés al medi de xarxes sense fils d'àrea personal amb tasses baixes de transmissió de dades.
El propòsit d'aquest estàndard és definir els nivells de xarxa bàsic per donar servei a un tipus específic de xarxa sense fils d'àrea personal centrada en l'habilitació de comunicació entre dispositius de baix cost i baixa velocitat de transmissió.
Es dona èmfasi al baix cost de comunicació entre nodes propers i amb molt poca infraestructura per afavorir encara més el baix consum, en la seva forma bàsica té una àrea d'actuació de 10 metres amb una tassa de transferència de 250 kbps.
Entre els aspectes més importants es troben l'adequació del seu us per a temps real per medi d'slots de temps garantits, control de col·lisions por CSMS/CA i suport integrat a les comunicacions segures. També s'inclouen funcions de control del consum d'energia com qualitat de l'enllaç i detecció d'energia.

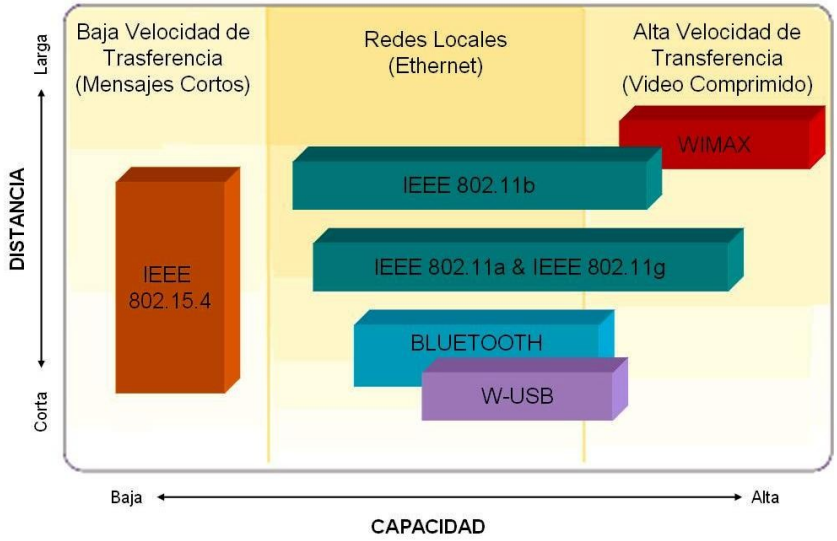


Figura 2.1.4.1 Estàndards de xarxa.

2.2. Estudi de mercat

Aquest projecte està orientat a la detecció d'incendis en els edificis, avui dia hi ha una gran varietat d'empreses que es dediquen a vendre seguretat pels edificis i una característica que va implícita es la detecció i extinció dels incendis que es puguin produir en les diferents habitacions i plantes. Exemple d'aquest tipus d'empreses poden ser: http://www.aguilera.es/ http://www.icima.es

El sistema desenvolupat en aquest projecte es cert que funciona però per poder-lo portar al mercat falta madurar-lo bastant en el sentit de poder instal·lar-lo en un edifici complert amb el que això implica, un desplegament d'una xarxa important així com unes comunicacions fiables entre els diferents nodes, també s'hauria de fer més robust a les col·lisions que produeix el medi.

Per tant cal destacar que el sistema encara que funciona està encara en fase de millora i maduració.

3. Descripció funcional

3.1. Sistema contra incendis

El funcionament del sistema contra incendis consisteix en una sèrie de procediments que es descriuen a continuació:
La mota emissora pren valors dels sensors de temperatura i de bateria cada 500ms (per defecte) i els envia a la mota receptora cada 1000ms (per defecte) que a la vegada els notifica a l'aplicació Java del PC gràcies a que hi està connectada a un port USB i s'ha generat una connexió amb l'aplicació SerialForwarder. L'aplicació Java del PC en aquest cas s'encarrega de monitoritzar els valors dels sensors de bateria i de temperatura.

Captura de pantalla de l'aplicació monitoritzant els valors:

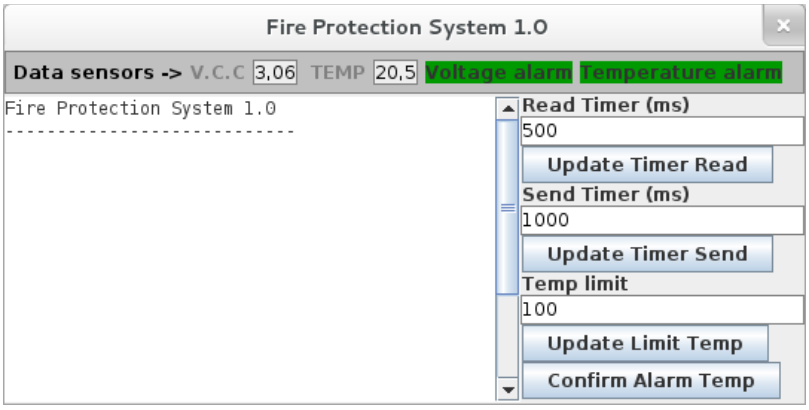


Figura 3.1.1 FireProtectionSystemJava monitoritzant els valors dels sensors.

Durant el període de temps de lectura dels sensors es comprova si es produeix alguna alarma d'incendi o de bateria baixa contrastant les lectures dels sensors amb els límits establerts per defecte (limit de temperatura per defecte 100ºC, límit bateria per defecte 2 Volts), el procés

d'enviament de les lectures dels sensors a la mota receptora no varia.
En cas que es produeixi una alarma d'incendi la mota la notificarà a la mota receptora cada segon i aquesta la repetirà a l'aplicació Java del PC. L'aplicació Java mostrarà per pantalla que s'ha produït l'alarma d'incendi.

Captura de pantalla de l'aplicació indicant l'alarma d'incendi:

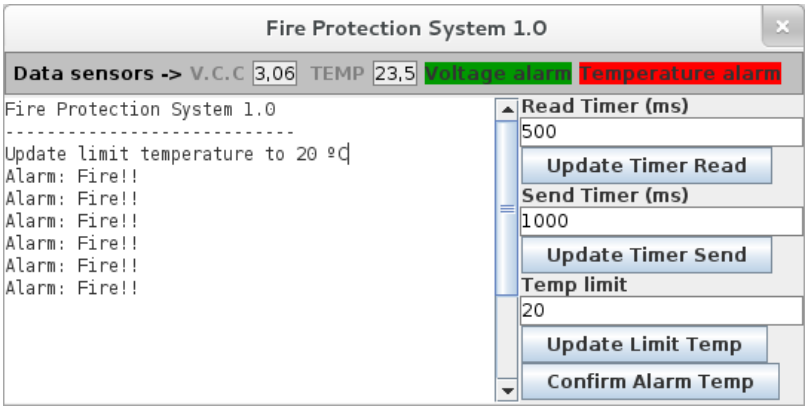


Figura 3.1.2 FireProtectionSystemJava indicant alarma d'incendi

En cas que es produeixi una alarma de bateria baixa la mota la notificarà a la mota receptora cada 60 segons i aquesta la repetirà a l'aplicació Java del PC. L'aplicació Java mostrarà per pantalla que s'ha produït l'alarma de bateria baixa. En aquest cas es deixa d'enviar les dades dels sensors per estalviar bateria però es segueix comprovant si es produeix alarma d'incendi.

Captura de pantalla de l'aplicació indicant l'alarma de bateria baixa:

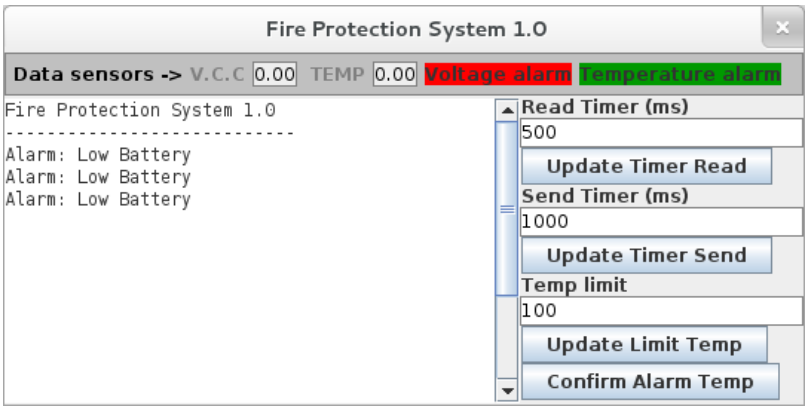


Figura 3.1.3 FireProtectionSystemJava indicant alarma de bateria baixa

Cas en el que mentre hi ha alarma de bateria baixa s'ha produït un incendi, en aquest cas s'ha fet saltar l'alarma d'incendi modificant la temperatura limit a 20ºC.

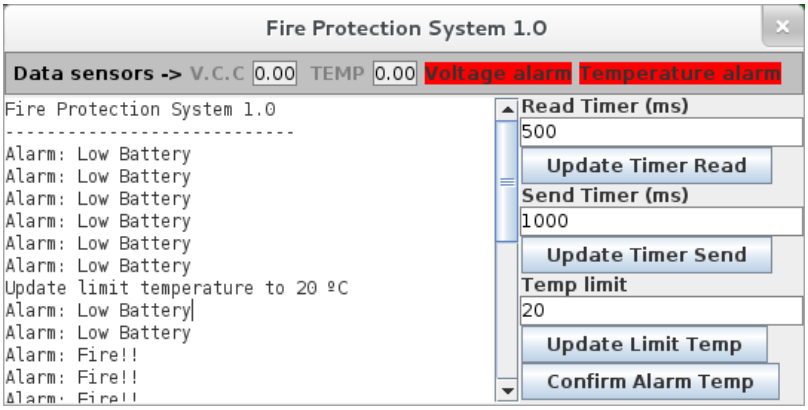


Figura 3.1.4 FireProtectionSystemJava indicant alarma de bateria baixa més alarma d'incendi

Des de l'aplicació Java es pot modificar el comportament de la mota emissora en quant al temps d'iteració en que es produeixen les lectures dels sensors, temps d'iteració en que s'envien les lectures i límit de temperatura amb el que es contrastarà amb el valor del sensor de temperatura si s'ha produït alarma d'incendi o no.
També hi ha un botó de confirmació d'alarma d'incendi, al confirmar l'alarma d'incendi es para el sistema de la mota emissora ***FireprotectionSystem***.

Captura amb els camps per poder fer les modificacions i els botons per executar l'enviament dels valors a modificar a la mota emissora:

Figura 3.1.5 Valors modificables.

Tot seguit s'ajunta el diagrama de seqüència, el diagrama de casos d'us i el diagrama de blocs del sistema amb el que es pot veure de manera gràfica el seu funcionament exposat en aquest apartat.

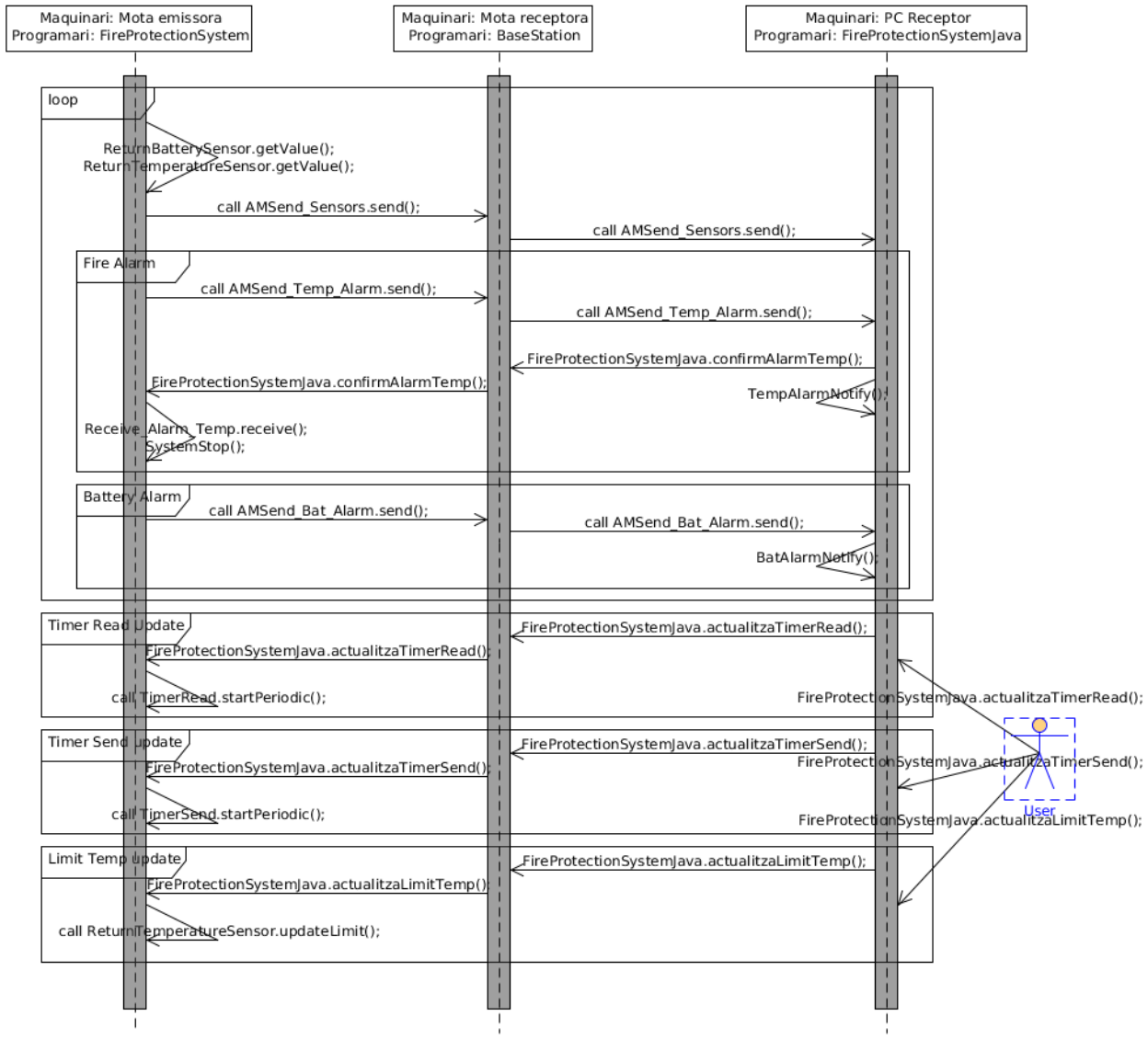


Figura 3.1.6 Diagrama de seqüència

Diagrama de casos d'us:

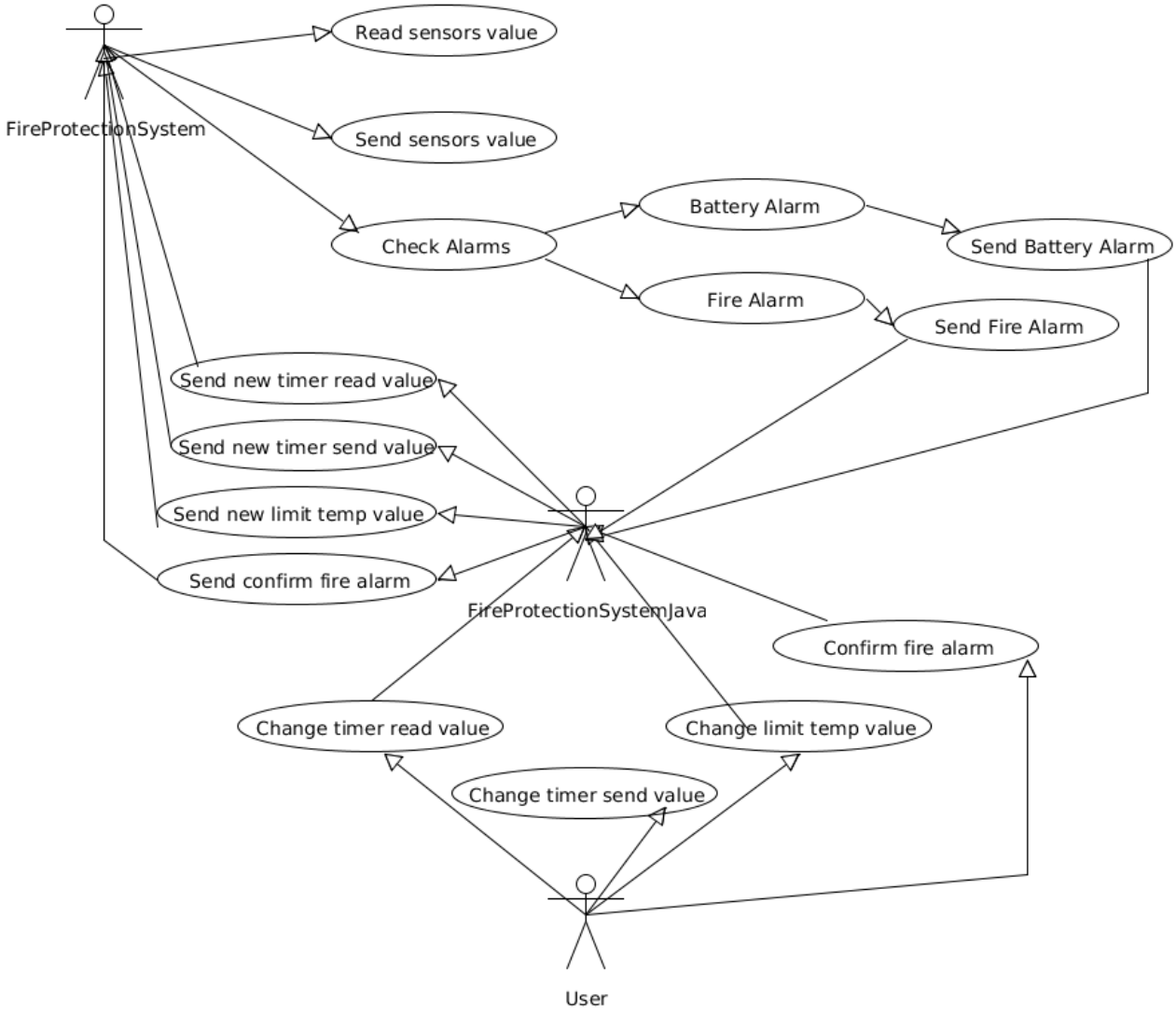


Figura 3.1.7 Diagrama de casos d'us

Diagrama de blocs: En aquest diagrama es pot observar com interaccionen els diferents components del sistema.
En la mota emissora els sensors de temperatura i voltatge interaccionen amb el processador ***ATmega1281*** en el qual s'està executant l'aplicació ***FireprotectionSystem*** que els hi demana els diferents valors i comprova les alarmes. La comunicació amb la mota receptora es realitza per ràdio amb el component de maquinari ***AT86RF230***, en la mota receptora tot el que li arriba per ràdio ho repeteix pel component ***Uart*** en aquest cas per USB a l'aplicació receptora del PC, i tot el que li arriba per USB ho repeteix per ràdio a la mota emissora.
En el PC tot el que es rep per ***Uart*** (USB) es rep per l'aplicació ***SerialForwarder*** la qual està fent de pont entre la mota receptora i l'aplicació ***FireProtectionSystemJava***.

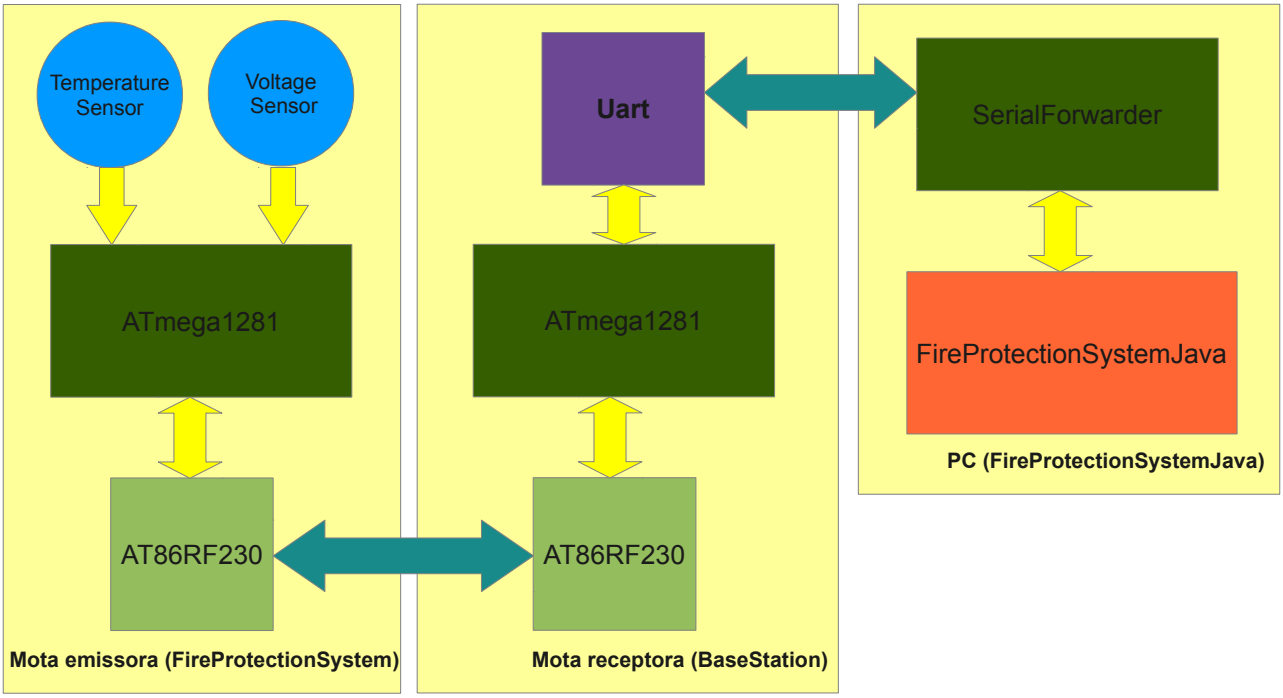


Figura 3.1.8 Diagrama de blocs

3.2. Sistema emissor (Mota)

El funcionament del sistema emissor, ***FireProtectionSystem***, ja s'ha explicat en el punt anterior, aquí ajuntem el diagrama d'activitats amb el que es pot veure d'una manera clara el seu funcionament:

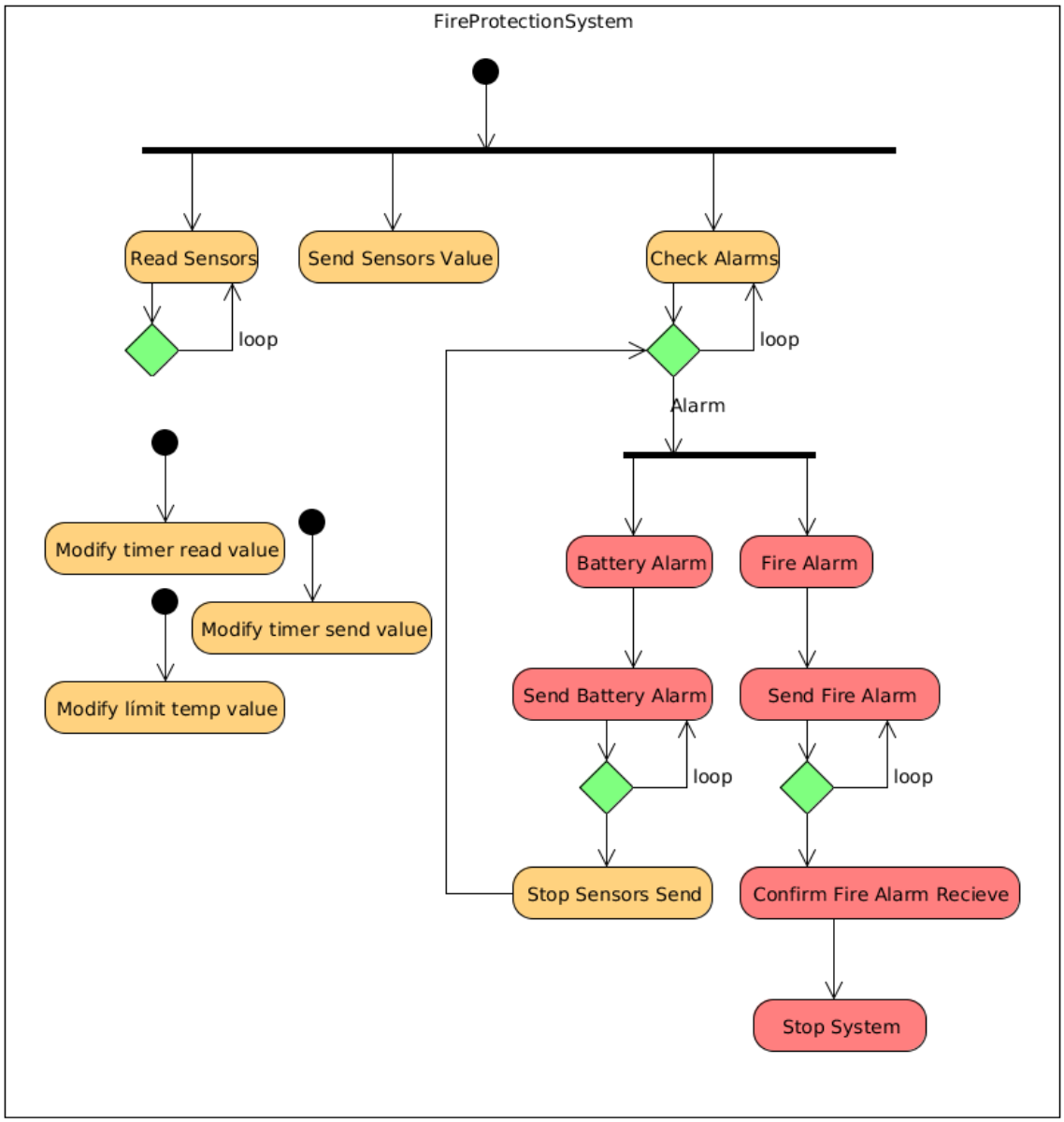


Figura 3.2.1 Diagrama d'activitats FireProtectionSystem

3.3. Sistema receptor (PC)

El funcionament del sistema receptor, ***FireProtectionSystemJava***, ja s'ha explicat en el punt anterior, aquí ajuntem el diagrama d'activitats amb el que es pot veure d'una manera clara el seu funcionament:

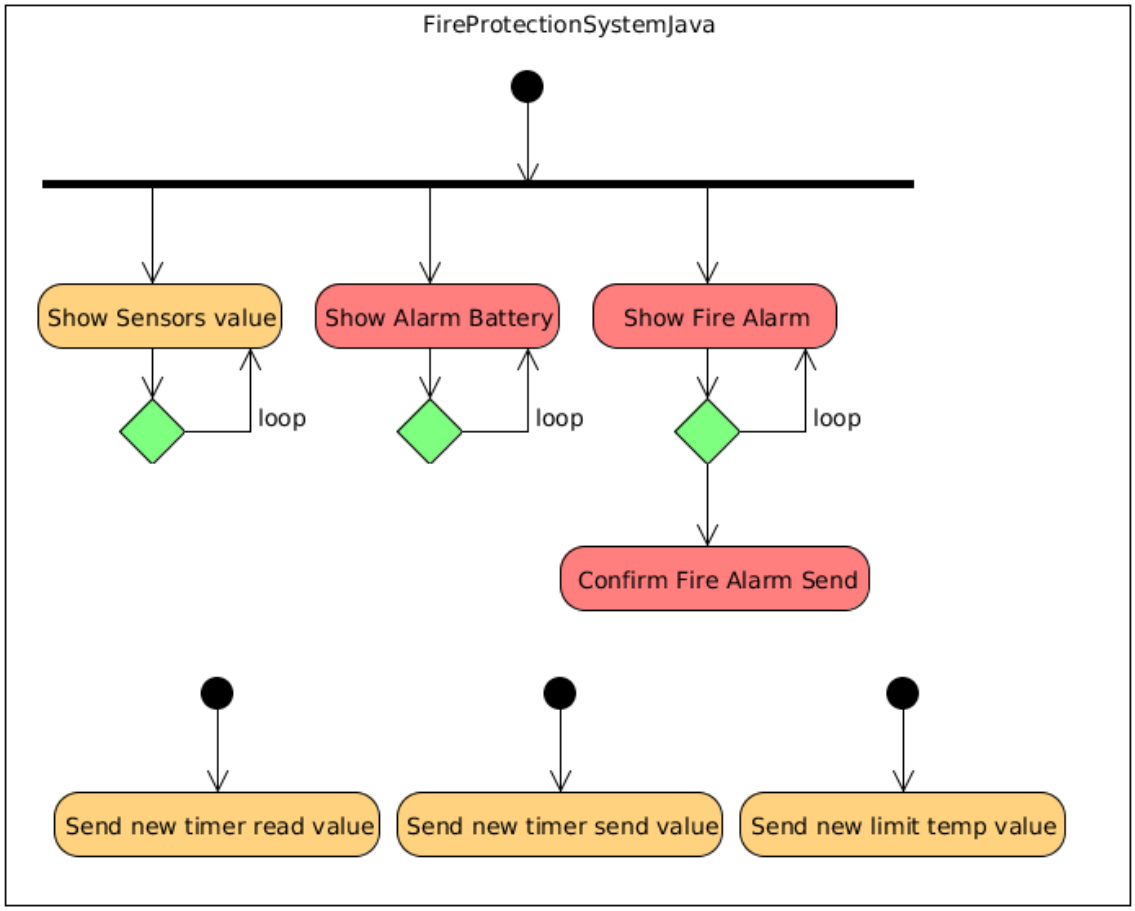


Figura 3.3.1 Diagrama d'activitats FireProtectionSystemJava

4. Descripció detallada

En aquest punt es descriu de manera tècnica les aplicacions desenvolupades, ***FireProtectionSystem*** per la mota emissora i ***FireProtectionSystemJava*** pel PC.

4.1. FireProtectionSystem

Hi ha definits 4 Timers que es posen en funcionament a l'inici:
Primer s'engega la ràdio de la mota i quan la ràdio s'ha encès es prossegueix a iniciar els 4 timers següents:

- **TimerRead:** Timer amb un període de 500 ms (es pot modificar des de l'aplicació Java del PC ***FireProtectionSystemJava***), per cada iteració si l'estat es READY o READ_TEMP es farà una lectura dels sensors i a més es comprovarà si s'ha produït alguna alarma d'incendi o de bateria baixa, gràcies a l'interfície ***ReturnSensor*** amb la que conjuntament amb els components ***TemperatureSensorC*** i ***BatterySensorC*** podem llegir els valors dels sensors, comprovar si s'ha produït alguna alarma i en el cas del ***TemperatureSensorC*** poder modificar el límit de temperatura.

- **TimerSend:** Timer amb un període de 1000ms (es pot modificar des de l'aplicació Java del PC ***FireProtectionSystemJava***), per cada iteració s'envia un missatge a la mota receptora gràcies al component que proporciona TinyOS: **AMSender** que ens permet enviar una estructura creada per nosaltres (en aquest cas l'estructura **SensorMsg**) encapsulada dins d'un **meesage_t** i a més s'encén el Led0.

- **TimerSendBatAlarm:** Timer amb un període de 60000ms, per cada iteració si s'ha produït alarma de bateria baixa s'envia un missatge a la mota receptora gràcies al component que proporciona TinyOS: **AMSender** que ens permet enviar una estructura creada per nosaltres (en aquest cas l'estructura **AlarmBattery**) encapsulada dins d'un **meesage_t** i a més s'encén el Led1.

- **TimerSendTempAlarm:** Timer amb un període de 1000ms, per cada iteració si s'ha produït alarma d'incendi s'envia un missatge a la mota receptora gràcies al component que proporciona TinyOS: **AMSender** que ens permet enviar una estructura creada per nosaltres (en aquest cas l'estructura **AlarmTemp**) encapsulada dins d'un **meesage_t** i a més s'encén el Led1.

Hi ha una sèrie d'esdeveniments programats per quan la mota rep missatges:

Receive_Timer_Read.receive: Quan es rep el nou valor del temps d'iteració de les lectures dels sensors, es procedeix a modificar-lo.
Receive_Timer_Send.receive: Quan es rep el nou valor del temps d'iteració del enviaments dels valors dels sensors, es procedeix a modificar-lo.

Receive_Limit_Temp.receive: Quan es rep el nou valor del límit de temperatura, es procedeix a modificar-lo.

Receive_Alarm_Temp.receive: Quan es rep la confirmació d'alarma d'incendi, es procedeix a parar el sistema.

Tot seguit exposem els fitxers escrits en ***NesC*** que composen l'aplicació:

FireProtectionSystemC.nc: Fitxer de configuració principal de l'aplicació, es on s'especifiquen quins components es fan servir així com les connexions (**wiring**) definides per connectar-los.

Cal recordar el concepte de **wiring**: A.c -> B.c, vol dir que el component A fa servir la interfície c i el component B la proporciona.

A continuació una imatge dels components i connexions (**wiring**) que es fan en aquest fitxer:

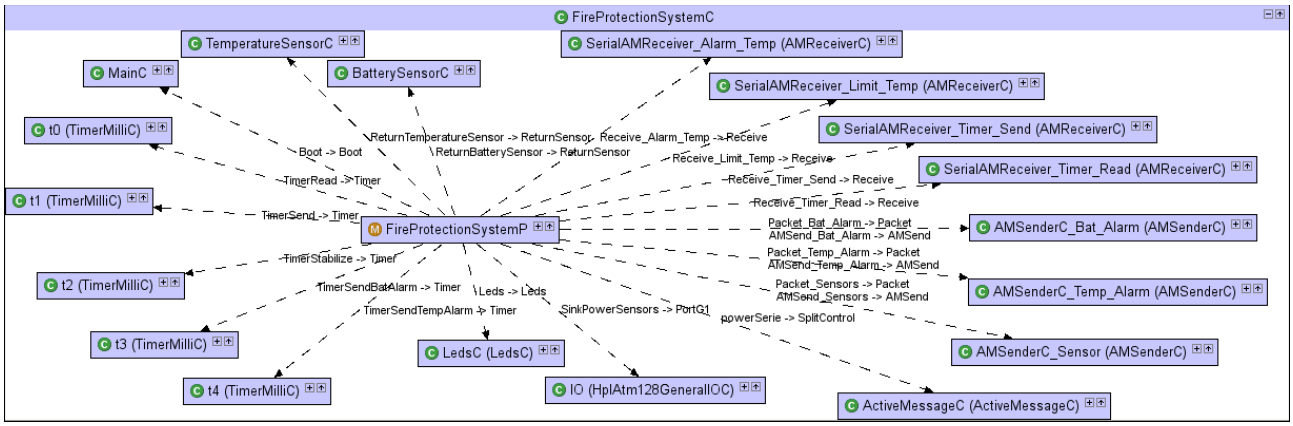


Figura 4.1.1 FireProtectionSystemC.nc

Components:

FireProtectionSystemP: Component on hi ha la gestió principal de l'aplicació.

MainC: Fem servir aquest component perquè ens proporciona la interfície Boot necessària per iniciar l'aplicació.

TimerMilliC: Component necessari per poder crear els bucles d'iteració (timers) amb els que llegirem els valors dels sensors, enviarem els valors dels sensors, sincronitzarem l'encesa dels sensors i enviarem les alarmes de bateria baixa i d'incendi.

LedsC: Component necessari per poder interactuar amb els leds de que disposen les motes, encendre i parar els leds.

HplAtm128GeneralIOC: Component necessari per poder encendre i parar els sensors.

ActiveMessageC: Component necessari per poder encendre i parar la ràdio.

AMSenderC: Component necessari per poder enviar missatges per ràdio.

AMReceiverC: Component necessari per poder rebre missatges per ràdio.

BatterySensorC: Component desenvolupat per nosaltres amb el que agafem el valor del sensor de bateria i controlem si hi ha una alarma de bateria.

TemperatureSensorC: Component desenvolupat per nosaltres amb el que agafem el valor del sensor de temperatura i controlem si hi ha una alarma d'incendi.

Connexions (wirings):

FireProtectionSystemP.ReturnBatterySensor -> BatterySensorC.ReturnSensor
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **ReturnBatterySensor** i el component **BatterySensorC** la proporciona. És fa servir per agafar el valor del sensor de bateria i a més per controlar si es produeix una alarma de bateria baixa.

FireProtectionSystemP.ReturnTemperatureSensor -> TemperatureSensorC.ReturnSensor
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **ReturnTemperatureSensor** i el component **TemperatureSensorC** la proporciona. És fa servir per agafar el valor del sensor de temperatura i a més per controlar si es produeix una alarma d'incendi.

FireProtectionSystemP.Boot -> MainC
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **Boot** i el component **MainC** la proporciona. És fa servir per iniciar l'aplicació (Boot).

FireProtectionSystemP.TimerRead -> TimerMilliC
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **TimerRead** i el component **TimerMilliC** la proporciona. És fa servir per iterar la lectura dels sensors de bateria i de temperatura.

FireProtectionSystemP.TimerSend -> TimerMilliC
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **TimerSend** i el component **TimerMilliC** la proporciona. Es fa servir per iterar l'enviament dels valors dels sensor a la mota receptora.

FireProtectionSystemP.TimerStabilize -> TimerMilliC
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **TimerStabilize** i el component **TimerMilliC** la proporciona. Es fa servir per iterar la sincronització de l'encesa dels sensors fins que no s'estabilitzin no es pot agafar mostres dels sensors.

FireProtectionSystemP.TimerSendBatAlarm -> TimerMilliC
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **TimerSendBatAlarm** i el component **TimerMilliC** la proporciona. Es fa servir per iterar l'enviament d'alarma de bateria a la mota receptora.

FireProtectionSystemP.TimerSendTempAlarm -> TimerMilliC
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **TimerSendTempAlarm** i el component **TimerMilliC** la proporciona. Es fa servir per iterar l'enviament d'alarma d'incendi a la mota receptora.

FireProtectionSystemP.Leds -> LedsC
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **Leds** i el component **LedsC** la proporciona. Es fa servir per gestionar l'encesa i parada dels leds de que disposen les motes.

FireProtectionSystemP.SinkPowerSensors -> IO.PortG1
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **SinkPowerSensors** i el component **IO** la proporciona amb la interfície **PortG1**. Es fa servir per gestionar l'encesa i parada dels sensors de que disposen les motes.

FireProtectionSystemP.powerSerie -> ActiveMessageC
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **powerSerie** i el component **ActiveMessageC** la proporciona. Es fa servir per gestionar l'encesa i parada de la ràdio de les motes.

FireProtectionSystemP.Packet_Sensors -> AMSenderC_Sensor
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **Packet_Sensors** i el component **AMSenderC_Sensor** la proporciona. Es fa servir per poder encapsular els missatges amb els valors dels sensors i així poder enviar-los.

FireProtectionSystemP.AMSend_Sensors -> AMSenderC_Sensor
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **AMSend_Sensors** i el component **AMSenderC_Sensor** la proporciona. Es fa servir per poder enviar els missatges amb els valors dels sensors per ràdio.

FireProtectionSystemP.Packet_Temp_Alarm -> AMSenderC_Temp_Alarm
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **Packet_Temp_Alarm** i el component **AMSenderC_Temp_Alarm** la proporciona. Es fa servir per poder encapsular els missatges amb l'alarma d'incendi i així poder enviar-los.

FireProtectionSystemP.AMSend_Temp_Alarm -> AMSenderC_Temp_Alarm
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **AMSend_Temp_Alarm** i el component **AMSenderC_Temp_Alarm** la proporciona. Es fa servir per poder enviar els missatges amb l'alarma d'incendi per ràdio.

FireProtectionSystemP.Packet_Bat_Alarm -> AMSenderC_Bat_Alarm
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **Packet_Bat_Alarm** i el component **AMSenderC_Bat_Alarm** la proporciona. Es fa servir per poder encapsular els missatges amb l'alarma de bateria baixa i així poder enviar-los.

FireProtectionSystemP.AMSend_Bat_Alarm -> AMSenderC_Bat_Alarm
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **AMSend_Bat_Alarm** i el component **AMSenderC_Bat_Alarm** la proporciona. Es fa servir per poder enviar els missatges amb l'alarma de bateria baixa per ràdio.

FireProtectionSystemP.Receive_Timer_Read -> SerialAMReceiver_Timer_Read.Receive
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **Receive_Timer_Read** i el component **SerialAMReceiver_Timer_Read** la proporciona amb la interfície **Receive**. Es fa servir per poder rebre per ràdio els missatges amb el nou valor del temps d'iteració de lectura de valors dels sensors.

FireProtectionSystemP.Receive_Timer_Send -> SerialAMReceiver_Timer_Send.Receive
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **Receive_Timer_Send** i el component **SerialAMReceiver_Timer_Send** la proporciona amb la interfície **Receive**. Es fa servir per poder rebre per ràdio els missatges amb el nou valor del temps d'iteració d'enviament de valors dels sensors.

FireProtectionSystemP.Receive_Limit_Temp -> SerialAMReceiver_Limit_Temp.Receive
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **Receive_Limit_Temp** i el component **SerialAMReceiver_ Limit_Temp** la proporciona amb la interfície **Receive**. Es fa servir per poder rebre per ràdio els missatges amb el nou valor del límit de temperatura.

FireProtectionSystemP.Receive_Alarm_Temp -> SerialAMReceiver_Alarm_Temp.Receive
Ens indica que el component **FireProtectionSystemP** fa servir la interfície **Receive_Alarm_Temp** i el component **SerialAMReceiver_ Alarm_Temp** la proporciona amb la interfície **Receive**. Es fa servir per poder rebre per ràdio els missatges de confirmació d'alarma d'incendi amb el que es procedeix a parar el sistema de la mota emissora (FireProtectionSystem).

FireProtectionSystemP.nc: Fitxer mòdul en el que hi ha el codi font de la gestió principal de l'aplicació.

FireProtectionSystem.h: Fitxer en el que s'han definit les diferents constants i estructures necessàries per l'aplicació.

Constants:

READY: Constant que indica que ja es pot començar el procés de lectura i control d'alarmes.

ERROR: Constant que es farà servir per indicar que s'ha produït algun error.

READ_BAT: Constant per indicar al sistema que faci una lectura del valor del sensor de bateria i a més es comprova si es produeix l'alarma de bateria baixa.

READ_TEMP: Constant per indicar al sistema que faci una lectura del valor del sensor de temperatura i a més es comprova si es produeix l'alarma d'incendi.

ALARM_BAT: Constant per indicar al sistema que s'ha produït l'alarma de bateria baixa.

ALARM_TEMP: Constant per indicar al sistema que s'ha produït l'alarma d'incendi.

TIMER_MILI_SEND_BAT_ALARM: Constant per indicar el temps d'iteració en el que s'enviarà, quan es produeixi, l'alarma de bateria baixa, val 60000ms que equival a 1 minut.

TIMER_MILI_SEND_FIRE_ALARM: Constant per indicar el temps d'iteració en el que s'enviarà, quan es produeixi, l'alarma d'incendi, val 1000ms que equival a 1 segon.

TIMER_MILI_READ: Constant per indicar el temps d'iteració en el que es llegiran els valors dels sensors, per defecte val 500 que equival a 0,5 segons. Com ja s'ha indicat en punts anteriors, aquest valor es pot modificar des de l'aplicació Java del PC *FireProtectionSystemJava*.

TIMER_MILI_SEND: Constant per indicar el temps d'iteració en el que s'enviaran els valors dels sensors, per defecte val 1000 que equival a 1 segon. Com ja s'ha indicat en punts anteriors, aquest valor es pot modificar des de l'aplicació Java del PC *FireProtectionSystemJava*.

AM_SENSORMSG: Constant per diferenciar els diferents Active Message que té l'aplicació aquest component així ho necessita per què si no en el receptor no es sabria que s'està rebent, val 40.

AM_LIMITTEMP: Constant per diferenciar els diferents Active Message que té l'aplicació aquest component així ho necessita per què si no en el receptor no es sabria que s'està rebent, val 41.

AM_ALARMTEMP: Constant per diferenciar els diferents Active Message que té l'aplicació aquest component així ho necessita per què si no en el receptor no es sabria que s'està rebent, val 42.

AM_ALARMBATTERY: Constant per diferenciar els diferents Active Message que té l'aplicació aquest component així ho necessita per què si no en el receptor no es sabria que s'està rebent, val 43.

AM_TIMERREADMSG: Constant per diferenciar els diferents Active Message que té l'aplicació aquest component així ho necessita per què si no en el receptor no es sabria que s'està rebent, val 44.

AM_TIMERSENDMSG: Constant per diferenciar els diferents Active Message que té l'aplicació aquest component així ho necessita per què si no en el receptor no es sabria que s'està rebent, val 45.

VREFMILIVOLTS: Constant per indicar al sistema els volts amb els que treballa la mota val 2560.
LIMIT_BAT: Constant per indicar al sistema el límit amb el que comprovarà si s'ha produït alarma de bateria baixa, val 345 que equival a 2 Volts.
LIMIT_TEMP: Constant per indicar al sistema el límit amb el que comprovarà si s'ha produït alarma d'incendi, val 600 que equival a 100ºC.
ALARMBAT_OK: Constant per indicar al sistema que s'ha produït alarma de bateria baixa.
ALARMBAT_KO: Constant per indicar al sistema que no s'ha produït alarma de bateria baixa.
ALARMFIRE_OK: Constant per indicar al sistema que s'ha produït alarma d'incendi.
ALARMFIRE_KO: Constant per indicar al sistema que no s'ha produït alarma d'incendi.

Estructures:

Estructura que serveix per enviar al receptor les dades del sensors de bateria i de temperatura

```
typedef nx_struct SensorMsg {
        nx_uint8_t moteId;
        nx_uint16_t countsBat;
        nx_uint16_t vrefmilivolts;
        nx_uint16_t countsTemp;
} SensorMsg;
```

Estructura que serveix perquè el receptor pugui enviar el limit d'alarma d'incendi

```
typedef nx_struct LimitTemp {
        nx_uint16_t countsTemp;
} LimitTemp;
```

Estructura que serveix per indicar al receptor l'alarma d'incendi
Si es rep del receptor un missatge d'aquest tipus serà el missatge de confirmació d'alarma

```
typedef nx_struct AlarmTemp {
        nx_uint8_t moteId;
} AlarmTemp;
```

Estructura que serveix per indicar al receptor l'alarma de bateria baixa.

```
typedef nx_struct AlarmBattery {
        nx_uint8_t moteId;
} AlarmBattery;
```

Estructura que serveix per actualitzar el període amb el que es llegeixen les dades dels sensors

```
typedef nx_struct TimerReadMsg{
        nx_uint32_t timerVal;
} TimerReadMsg;
```

Estructura que serveix per actualitzar el període amb el que s'envien les dades dels sensors

```
typedef nx_struct TimerSendMsg{
        nx_uint32_t timerVal;
} TimerSendMsg;
```

ReturnSensor.nc: Interfície creada per nosaltres que proporciona els mètodes següents:

command uint16_t getValue() : Retorna el valor del sensor

command uint16_t ReturnAlarm(): Retorna si s'ha produït una alarma o no

command void updateLimit(uint16_t val): Actualitza el valor del límit del sensor

es fa servir en els components ***TemperatureSensorC*** i ***BatterySensorC.***

TemperatureSensorC.nc: Fitxer de configuració pel mòdul ***TemperatureSensorP***

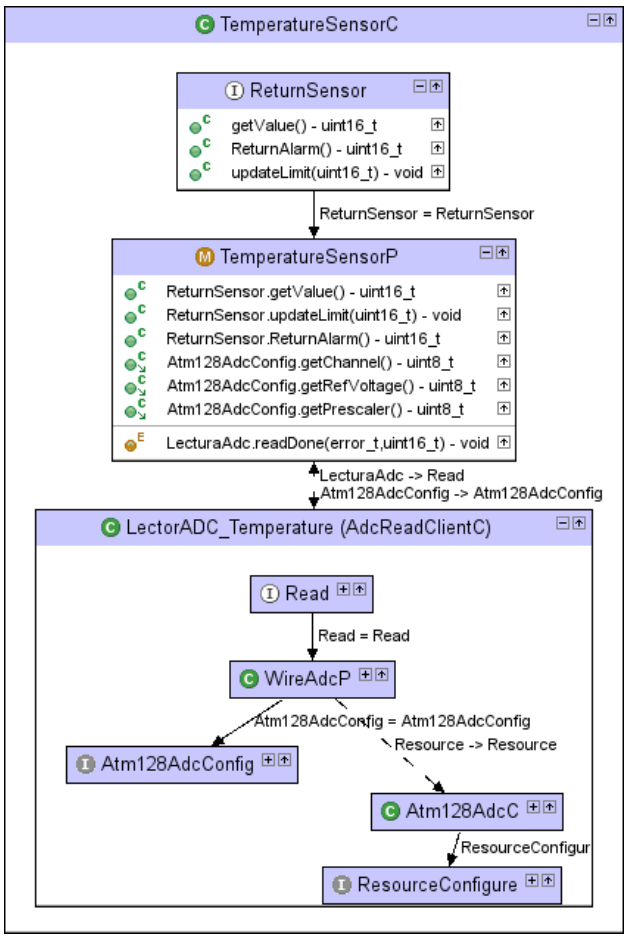


Figura 4.1.2 TemperatureSensorC.nc

TemperatureSensorP.nc: Fitxer que proporciona el codi font pel mòdul ***TemperatureSensorP*** el farem servir per llegir el valor del sensor de temperatura, comprovar si s'ha produït alarma d'incendi i modificar el valor del límit de temperatura.

BatterySensorC.nc: Fitxer de configuració pel mòdul ***BatterySensorP***

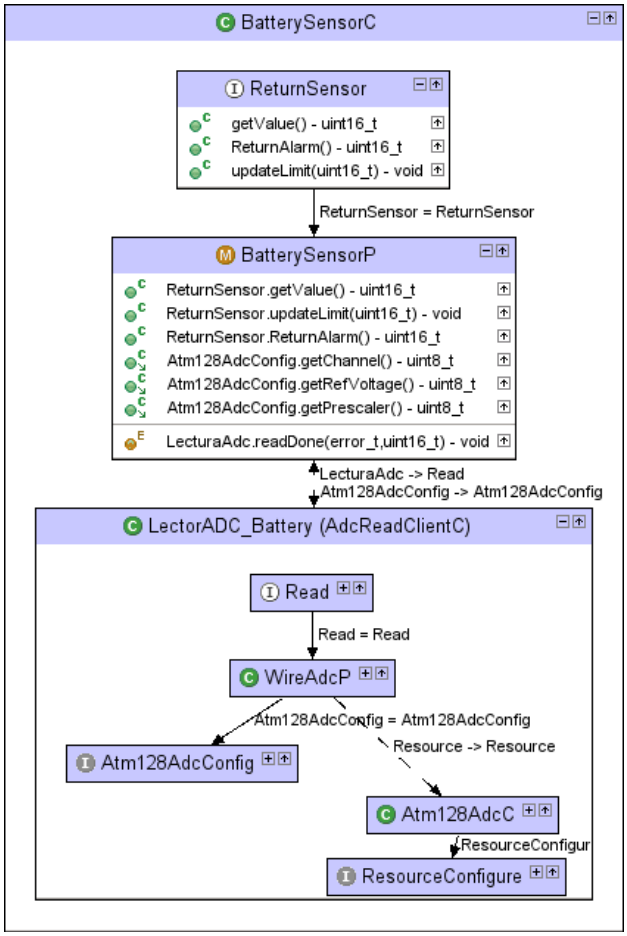


Figura 4.1.3 BatterySensorC.nc

BatterySensorP.nc: Fitxer que proporciona el codi font pel mòdul ***BatterySensorP*** el farem servir per llegir el valor del sensor de bateria, comprovar si s'ha produït alarma de bateria baixa i si fos el cas es podria modificar el valor del límit de bateria baixa, no s'ha implementat ja que el projecte en principi no ho requeria.

MakeFile: Fitxer en el que s'indiquen una sèrie de paràmetres necessaris per la compilació de l'aplicació.

JmakeFile: Fitxer en el que s'especifiquen una sèrie de paràmetres perquè es puguin crear les classes necessàries per la part de Java, ens ajudarem de l'eina **mig** (Message Inteface Generator) la proporciona **TinyOS** per poder crear des de les estructures utilitzades en l'aplicació ***FireprotectionSystem*** les classes Java necessàries per l'aplicació ***FireProtectionSystemJava.***

4.2. FireProtectionSystemJava

TinyOS proporciona una llibreria per Java ***MoteIF*** amb la que podrem rebre i enviar missatges a la mota connectada al **socket** que ha creat l'aplicació **SerialForwarder**.
Es creen dos instàncies d'aquesta classe per poder monitoritzar els valors dels sensors provinents de la mota així com les alarmes i poder enviar missatges a la mota emissora per indicar-li els nous valors de temps d'iteració per llegir i enviar els valors dels sensors i a més per enviar el missatge de confirmació d'alarma d'incendi.

Està composada per una sèrie de fitxers que exposem a continuació.
FireProtectionSystemJava.java: És l'encarregat de rebre els missatges que envia l'aplicació *FireProtectionSystem* amb els valors dels sensors, indicació d'alarma d'incendi i indicació d'alarma de bateria baixa, també s'encarrega d'enviar els missatges amb els valors a modificar de l'interval de temps amb el que es llegeixen els valors dels sensors, interval de temps amb el que s'envien els valors dels sensors, límit de temperatura i confirmació d'alarma d'incendi.

GraphicalUserInterface.java: Codi font pertinent a l'interfície gràfica s'han fet servir les llibreries de Java swing i awt. Adjuntem captura de pantalla:

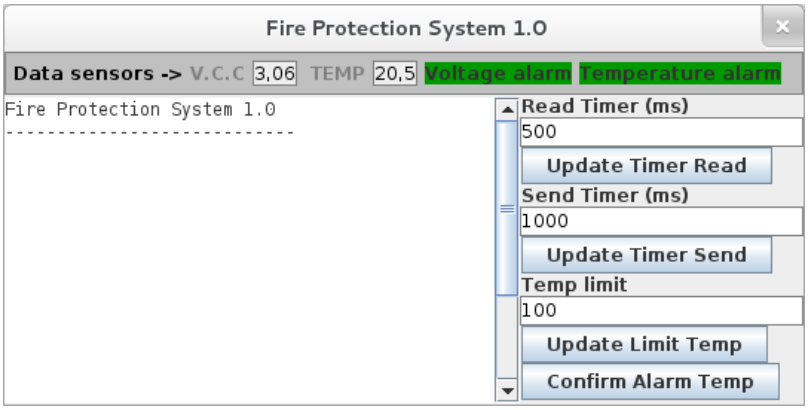


Figura 4.2.1 GraphicalUserInterface.java

AlarmBattery.java: Classe Java generada amb l'eina **mig** (Message Interface Generator) basada en l'estructura ***AlarmBattery*** definida en el fitxer ***FireProtectionSystem.h*** que ens servirà per detectar quan la mota emissora envia l'alarma de bateria baixa.

AlarmTemp.java: Classe Java generada amb l'eina **mig** (Message Interface Generator) basada en l'estructura ***AlarmTemp*** definida en el fitxer ***FireProtectionSystem.h*** que ens servirà per detectar quan la mota emissora envia l'alarma d'incendi i per enviar-li la confirmació d'alarma d'incendi.

LimitTemp.java: Classe Java generada amb l'eina **mig** (Message Interface Generator) basada en l'estructura ***LimitTemp*** definida en el fitxer ***FireProtectionSystem.h*** que ens servirà per enviar el nou valor del límit de temperatura a la mota emissora.

SensorMsg.java: Classe Java generada amb l'eina **mig** (Message Interface Generator) basada en l'estructura ***SensorMsg*** definida en el fitxer ***FireProtectionSystem.h*** que ens servirà per monitoritzar els valors dels sensors provinents de la mota emissora.

TimerReadMsg.java: Classe Java generada amb l'eina **mig** (Message Interface Generator) basada en l'estructura ***TimerReadMsg*** definida en el fitxer ***FireProtectionSystem.h*** que ens servirà per enviar el nou valor del temps d'iteració de lectura dels valors dels sensors, (TimerRead).

TimerSendMsg.java: Classe Java generada amb l'eina **mig** (Message Interface Generator) basada en l'estructura ***TimerSendMsg*** definida en el fitxer ***FireProtectionSystem.h*** que ens servirà per enviar el nou valor del temps d'iteració d'enviament dels valors dels sensors, (TimerSend).

5. Viabilitat tècnica

Cal destacar que aquest projecte ara per ara no es podria comercialitzar ja que es troba en una fase de maduració i millora, per poder-lo implementar s'haurien d'assolir una sèrie objectius que no ha donat temps d'acabar-los:

- Fer el sistema de comunicació sense fils més robust a les col·lisions produïdes pel medi, implementant un sistema d'acks.
- Modificar el codi per poder monitoritzar més d'una mota a la vegada.
- Desenvolupar un sistema d'interconnexió entre múltiples nodes, perquè per exemple es pogués monitoritzar diferents sales d'una planta d'un edifici.

El punts forts d'aquest projecte son les eines utilitzades pel seu desenvolupament, Java i TinyOS ja que actualment són projectes mantinguts i en fase emergent, així ens assegurem en un futur que aquest projecte no estigui desfasat.

El punts febles serien tots els objectius que falta per implementar.

6. Valoració econòmica

Es fa una valoració econòmica del que s'ha fet fins ara, tot i que falta bastant perquè el projecte es pugui implementar en un entorn de producció ja que aquest tipus de sistemes han de ser totalment confiables i robust a fallades, ara per ara no compleix aquest dos requisits.

Punts a valorar:

- 40€ per mota = 40x2 = 80€
- Temps d'aprenentatge de TinyOS, meshprog, SerialForwarder, Java, NesC, wiring.
- Temps de desenvolupament
- Temps de les proves realitzades.
- Temps per documentar-ho tot, memòria i presentació

La valoració econòmica del temps depèn de cadascú, hem tingut 4 mesos hàbils per realitzar el projecte però es evident que no hi hem dedicat 4 mesos sencers ja que en aquest període he fet 2 assignatures més.

7. Conclusions

7.1. Conclusions

Gràcies a aquest projecte he profunditzat en el món del sistemes encastats i les xarxes de sensors sense fils (WSN) que desconeixia. Al principi del projecte fins i tot em vaig plantejar si m'havia equivocat al haver triat aquest camp ja que a l'inici tot es fa molt costa amunt i fins que no comences a entendre el funcionament de ***TinyOS, NesC, meshprog, SerialForwarder*** es fa realment difícil avançar amb la conseqüència de futur nerviosisme, però al final estic molt content del que he arribat a desenvolupar, tot i que amb una mica més de temps el sistema estaria més depurat i millorat.

Si algú llegeix aquesta memòria adverteixo que aquest camp (Sistemes Encastats) es un camp de dificultat elevada i que requereix de molta dedicació en quan a aprenentatge de totes les parts, es requereix un nivell d'anglès acceptable ja que gairebé tota la documentació sobre les eines a utilitzar en concret TinyOS està en anglès, però es evident que amb dedicació i ganes es treu endavant.

7.2. Proposta de millores

Les propostes de millores ja s'han esmentat amb anterioritat, les exposem com a objectius:

- Fer el sistema de comunicació sense fils més robust a les col·lisions produïdes pel medi, implementant un sistema d'acks.
- Modificar el codi per poder monitoritzar més d'una mota a la vegada.
- Desenvolupar un sistema d'interconnexió entre múltiples nodes, perquè per exemple es pogués monitoritzar diferents sales d'una planta d'un edifici.

7.3. Errors detectats

Hi ha un error detectat a l'hora d'enviar els missatges per modificar els valors (intervals de temps i límit de temperatura) des de l'aplicació Java a la mota emissora, en concret en el fixer java: ***FireProtectionSistemJava.Java***

A vegades el que s'envia a la mota no arriba i és un problema de col·lisió entre el missatges transmesos i el medi, per arreglar-ho s'hauria d'implementar un mètode de reconeixement de paquets (Ack).

7.4. Autoavaluació

A la finalització del projecte comprovem que s'han assolit tots els objectius proposats, els tornem a exposar a continuació:

La mota pren una mesura de temperatura i bateria cada N ms, la bateria es pren per saber quin nivell de pila li queda a la mota.

- Llegir el valor del sensor temperatura cada N ms.
- Llegir el valor del sensor de la bateria cada N ms.

La mota només envia la dada cada M ms

- Crear una estructura per poder desar les dades dels sensors
- Enviar les dades per ràdio cada M ms.

L'usuari pot configurar els temps M i N.

- Crear una interfície gràfica amb la que l'usuari pot modificar els temps M i N.
- Enviar els temps M i N a la mota emissora i modificar-los.

Si la temperatura supera un nivell (modificable per l'usuari) s'envia una alarma cada 1s fins que l'usuari envii un missatge de confirmació de l'alarma

- Comprovar el nivell de temperatura cada N ms
- En cas d'alarma d'incendi enviar l'alarma cada segon al receptor
- Segui enviant les dades del sensors al receptor.
- Quan es rebi la confirmació d'alarma per part del receptor apagar el sistema.
- L'usuari pot modificar el límit de temperatura des de la interfície gràfica.

Si la bateria baixa de un nivell establert fixat en el codi, la mota envia una alarma cada 60s de nivell de bateria baix.

- Comprovar el nivell de bateria cada M ms
- En cas d'alarma de bateria enviar l'alarma cada 60 ms.

- Seguir comprovant el sensor de temperatura per possibles alarmes d'incendi
- En cas d'alarma d'incendi enviar l'alarma al receptor.

En una primera fase tots els valors fixes, i després afegirem la configuració de nivells per part de l'usuari.

- Crear una interfície d'usuari per poder modificar els valors N, M i de límit de temperatura
- Enviar les modificacions a la mota emissora i actualitzar els valors.

He tingut problemes a l'hora d'implementar el control de les alarmes comparant el valor dels sensors amb el límit preestablert, al final ho he resolt comparant-ho a l'interfície ReturnSensor de cada sensor ja que des del programa principal no funcionava com s'esperava.

8. Glossari

TinyOS: És un sistema operatiu de codi obert basat en components per a xarxes de sensors sense fils.

MIG: Message Interface Generator, eina per generar les classes Java a partir de les estructures en NesC.

COU_1_2 24: Motes amb les que s'ha realitzat aquest projecte.

802.15.4: És un estàndard que defineix el nivell físic i el control d'accés al medi de xarxes sense fils d'àrea personal

ZigBee: És el nom de l'especificació d'un conjunt de protocols d'alt nivell de comunicacions sense fils per la seva utilització en radiodifusió digital de baix consum

Meshprog: Programa per carregar a les motes COU24 el codi ***NesC*** compilat.

GUI: (Graphical User Interface)**,** Interfície gràfica d'usuari.

JDT: Java Development Tools.

Eclipse IDE: Plataforma de codi obert de programació multi llenguatge.

WSN: Xarxa de sensors sense fils.

ADC: Convertidor Analògic a Digital

UART: Transmisor/Receptor Universal Asíncron.

Java: Llenguatge de programació orientat a l'objecte.

UML: Llenguatge de Modelat Unificat.

ATmega1281: Microcontrolador de la marca ATMEL.

MCP9700: Transistor especialment dissenyat per mesurar temperatura. També es caracteritza pel seu baix consum.

CP2102: Circuit integrat que fa de pont entre un bus USB i la mota.

NesC: Llenguatge de programació C optimitzat a les limitacions de memòria de las xarxes de sensors.

9. Bibliografia

http://es.wikipedia.org/wiki/Protecci%C3%B3n_contra_incendios

http://cv.uoc.es/app/mediawiki14/wiki/P%C3%A0gina_principal

http://tinyos.net/

http://es.wikipedia.org/wiki/Memoria_de_acceso_aleatorio

http://es.wikipedia.org/wiki/EEPROM

http://en.wikipedia.org/wiki/Memory-mapped_I/O

http://ca.wikipedia.org/wiki/Modulaci%C3%B3_per_ampl%C3%A0ria_d%27impuls

http://es.wikipedia.org/wiki/UART

http://edison.upc.edu/curs/llum/fotometria/magnitud.html

http://es.wikipedia.org/wiki/Luz

http://tos-ide.ethz.ch/wiki/index.php

www.eclipse4you.com/?q=en/eclipse_plugins/yeti02

http://pervasive.researchstudio.at/portal/projects/meshprog

http://wikipedia.orange.es/wiki/NesC

http://es.wikipedia.org/wiki/Red_de_sensores http://www.sase.com.ar/2011/files/2010/11/SASE2011-Hardware_para_Redes_de_Sensores_Inalambricos2.pdf

http://networks.cs.ucdavis.edu/~yick/research/SensorCost.html http://en.wikipedia.org/wiki/Sensor_node

http://www.cister.isep.ipp.pt/wsn/(S(tlsbse55b0wjyh55vrcaqriw))/MainPage.ashx?NoRedirect=1

http://atc.ugr.es/~aprieto/TIC_socio_sanitario/A11_4_05_Redes_sensores.pdf

http://en.wikipedia.org/wiki/Sensor_node

http://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes

http://en.wikipedia.org/wiki/Contiki

http://en.wikipedia.org/wiki/ERIKA_Enterprise

http://en.wikipedia.org/wiki/Nano-RK

http://en.wikipedia.org/wiki/TinyOS

http://en.wikipedia.org/wiki/LiteOS

http://rii.galeon.com/aficiones2095541.html

http://web.automaticaeinstrumentacion.com/comunicaciones-inalambricas-bajo-el-estandar-isa100-11a/

http://academica-e.unavarra.es/handle/2454/1875?show=full

http://es.wikipedia.org/wiki/ZigBee

Java SE 6 – Manual Imprescindible – F.Javier Moldes Teo – ANAYA

10. Annexos

10.1. Execució i compilació

Tot i que les aplicacions ja estan compilades:

Per compilar l'aplicació ens hem de situar en el directori on hi hagi el codi font de l'aplicació FirePrtoectionSystem i executar:

make cou24

L'aplicació Java es pot importar en un projecte de Java, atenció s'ha de tenir configurat i en funcionament l'entorn de TinyOS.

Tot seguit:

1. Instal·lar l'aplicació ***BaseStation*** en la mota receptora:

meshprog -t/dev/ttyUSB0 -f./build/cou24/main.srec

on ttyUSB0 es el port USB per on està connectada la mota a l'ordinador

2. Instal·lar l'aplicació ***FireProtectionSystem*** en la mota emissora:

meshprog -t/dev/ttyUSB0 -f./build/cou24/main.srec

on ttyUSB0 es el port USB per on està connectada la mota a l'ordinador

3. Connectar la mota receptora a un port USB de l'ordinador i executar el ***SerialForwarder*** per establir la comunicació entre la mota receptora i el PC amb Java:

java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB0:19200

on ttyUSB0 es el port USB per on està connectada la mota a receptora

4. Posar piles a la mota emissora i situar-la a on vulguem perquè comenci a enviar dades dels sensors i comprovar les alarmes.

5. Executar l'aplicació ***FireProtectionSystemJava.jar*** per rebre les dades dels sensors, alarmes i poder enviar els nous valors d'interval de temps i de límit de temperatura a la mota emissora:

java -jar FireProtectionSystem.jar