Rapport du projet tutoré

Application de vidéo surveillance avec un réseau de capteurs multimédia sans fils

Master 1 technologie internet 2010 2011

CASAS CHICA Juan Pablo

NGUYEN Viet Minh

VU Ninh Thanh Hien

Responsable M.CONG DUC PHAM

6 mai 2011

Remerciements

Nous remercions tout particulièrement notre encadrant monsieur Pham d'avoir proposé ce sujet et de nous avoir prêté le matériel, monsieur Cariou de nous l'avoir attribué, et monsieur Laurent Lacayrelles de nous avoir aidé à installer l’environnement de travail.

Table des matières

1 Introduction

Dans le cadre de nos études en première année de master Technologie Internet à l'UPPA, un projet tutoré doit être réalisé. Il est choisi parmi tous ceux qui sont proposés. Il est pour but d'évaluer notre capacité d'adaptation face aux impératifs professionnels. La durée de ce projet est limitée dans le second semestre de la première année de master

Ce rapport sera présenté en trois parties principales. La première partie sera la présentation du sujet. La deuxième partie sera la description du matériel et l'environnement de travail, la troisième partie sera la description l'étude réalisée et du travail effectué.

2 Problématique

2.1 Présentation du sujet

Un capteur (voir figure) est un petit appareil autonome capable d'effectuer des mesures simples sur son environnement immédiat. L'utilisation de ces capteurs n'a rien d'une nouveauté, ceux-ci sont utilisés depuis longtemps dans des domaines comme l'aéronautique ou l'automobile. Ce qui est novateur, c'est la possibilité pour ces capteurs de communiquer de manière radio (réseaux sans-fils de type Wi-Fi) avec d'autres capteurs proches (quelques mètres) et pour certains d'embarquer de la capacité de traitement (processeur) et de la mémoire… On peut ainsi constituer un réseau de capteurs sans-fils (RCSF) qui collaborent sur une étendue assez vaste. Comme les ressources d'un capteur sont très limitées, on peut même envisager que la réalisation d'un service complexe puisse être effectuée grâce à une composition de services plus simples et donc à une forme de collaboration "intelligente" des capteurs. Ces réseaux de capteurs soulèvent un intérêt grandissant de la part des industriels ou d'organisations civiles où la surveillance et la reconnaissance de phénomène physique est une priorité. Par exemple, ces capteurs mis en réseau peuvent surveiller une zone délimitée pour détecter soit l'apparition d'un phénomène donné (apparition de vibrations, déplacement linéaire...) soit mesurer un état physique comme la température (détection des incendies en forêts) ou la pression. Dans beaucoup de scénarios de gestion de crise (séismes, inondations,...) ces réseaux de capteurs peuvent permettre une meilleure connaissance du terrain afin d'optimiser l'organisation des secours, ou bien même renseigner précisément les scientifiques sur les causes d'un phénomène physique particulier. Des capteurs plus avancés embarquent des caméras avec lesquelles il est possible de capturer des scènes vidéo. Ce sont ces réseaux de capteurs vidéo sans-fils (RCVSF) qui nous intéressent dans ce projet.

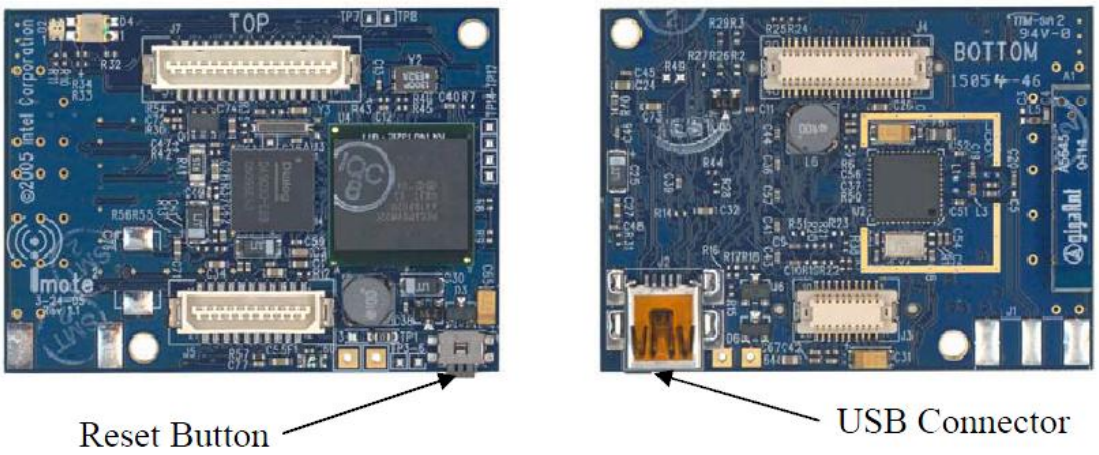
2.2 Etat actuel

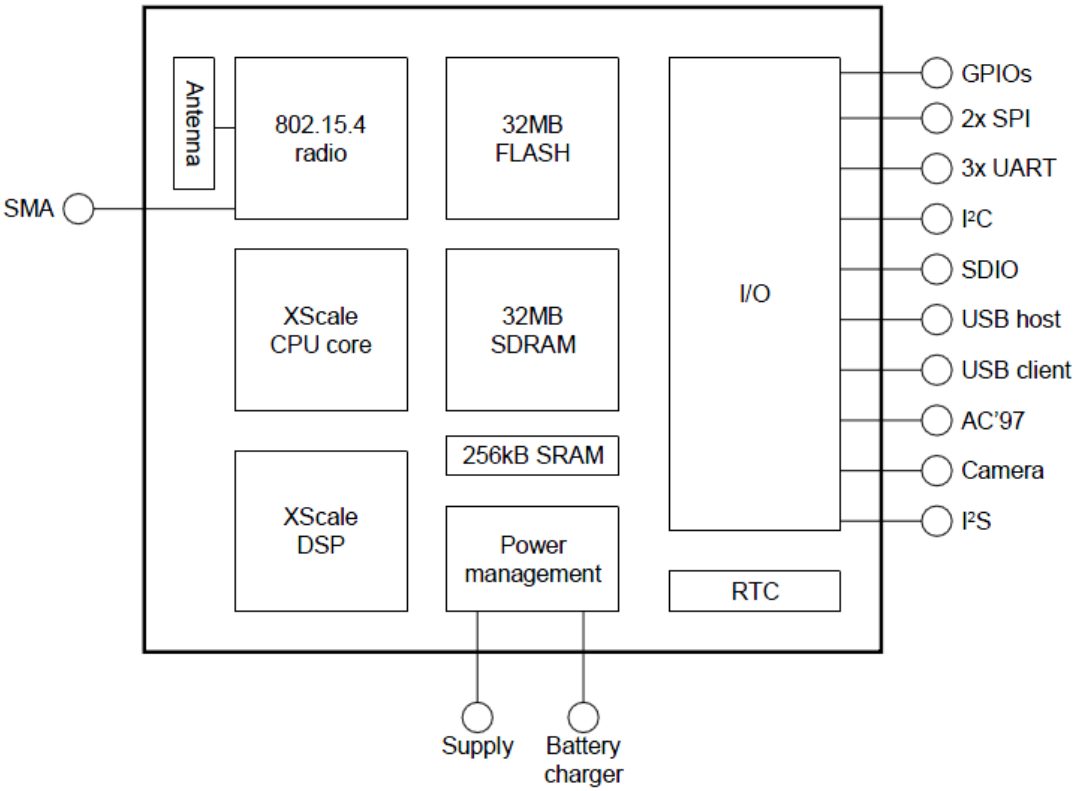
Dans notre sujet une application simple permettant de capturer de manière périodique une image existe déjà . Notre objectif suivant est d'améliorer cette application en incorporant le transfert d'image d'un capteur à un autre et de rajouter une partie de contrôle permettant de faire varier la vitesse de capture des images.

3 Description du matériel et l'environnement de travail

3.1 Imote2 Radio Processor Board (IPR2400)

Le Imote2 Crossbow est une plate-forme de capteurs avancés nœud de réseau conçues pour répondre aux applications sans fils du réseau de capteurs nécessitant élevée du processeur / performances de la liaison DSP et sans fils et la fiabilité. La plate-forme est construite autour d'Intel XScale ®, PXA271. Il intègre une radio 802.15.4 (TI CC2420) avec une antenne à bord. Il expose une "carte de base du capteur" interface, composée de deux connecteurs d'un côté de la planche, et un «conseil de pointe du capteur" interface, composée de deux connecteurs à haute densité de l'autre côté de la planche. Le Imote2 est une plateforme modulaire empilables et peuvent être empilés avec les conseils du capteur de personnaliser le système pour une application spécifique, avec un «conseil de la batterie" pou





3.2 Le système d'exploitation pour RCSF

TinyOS est un système d'exploitation intégré, modulaire, destiné aux réseaux de capteurs miniatures. Cette plate-forme logicielle ouverte et une série d'outils développés par l'Université de Berkeley est enrichie par une multitude d'utilisateurs. En effet, TinyOS est le plus répandu des OS pour les réseaux de capteurs sans-fil. Il est utilisé dans les plus grands projets de recherches sur le sujet (plus de 10.000 téléchargements de la nouvelle version). Un grand nombre de ces groupes de recherches ou entreprises participent activement au développement de cet OS en fournissant de nouveaux modules, de nouvelles applications,... Cet OS est capable d'intégrer très rapidement les innovations en relation avec l'avancement des applications et des réseaux eux même tout en minimisant la taille du code source en raison des problèmes inhérents de mémoire dans les réseaux de capteurs. La librairie TinyOS comprend les protocoles réseaux, les services de distribution, les drivers pour capteurs et les outils d'acquisition de données. TinyOS est en grande partie écrit en C mais on peut très facilement créer des applications personnalisées en langages C, NesC, et Java.

3.3 NesC : le langage de programmation de TiniOS

NesC est un langage conçus pour incarner les concepts structurant et le modèle d'exécution de TinyOS. C'est une extension du langage C orientée composant ; il support alors la syntaxe du langage C et il est compilé vers le langage C avant sa compilation en binaire.

Concepts de nesC

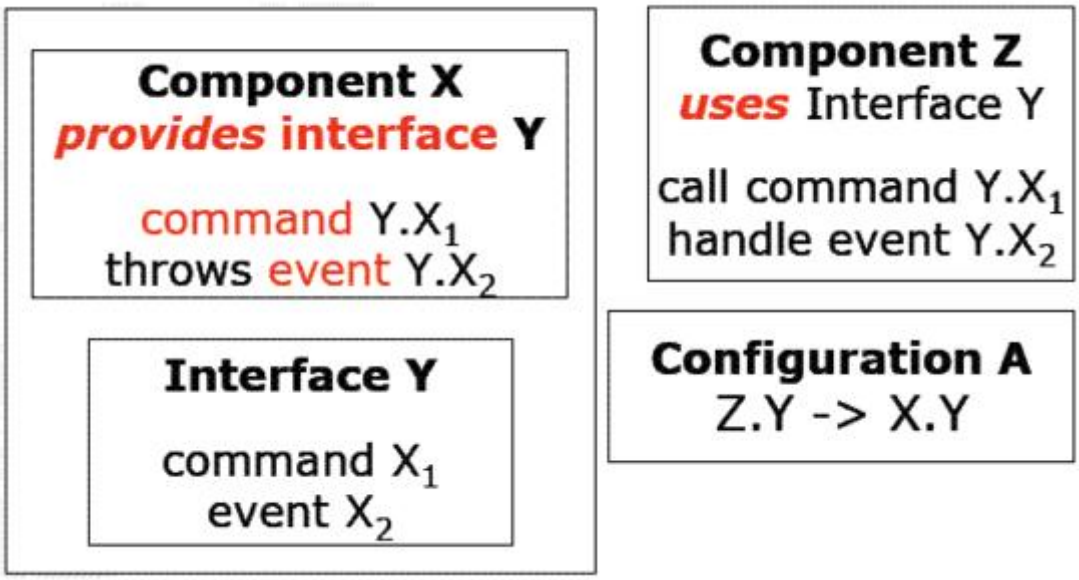
- L'unité de code de base de nesC est le composant "component"
- Un composant
  - Exécute des Commandes
  - Lance des Events
  - Dispose d'un Frame pour stocker l'état local
  - Utilise la notion de Tasks pour gérer la concurence
- Un Composant implémente des interfaces utilisées par d'autres composants pour communiquer avec ce composant

Composant

Il existe deux types de composants

1. Module : composant implémenté avec du code
2. Configuration : composants reliés ensemble pour former un autre composant

Application TinyOS



Interface

Une interface définie les interactions entre deux composants.

Les interfaces sont bidirectionnelles

Elles spécifient un ensemble de fonctions à implémenter par les composants fournisseurs de l'interface (commands), et un ensemble à implémenter par les composants utilisateurs de l'interface (events)

command vs. event

Les commands font typiquement des appels du haut vers le bas (des composants applicatifs vers les composants plus proches du matériel), alors que les events remontent les signaux du bas vers le haut.

Exemple d'interface

```
interface SendMsg {

command result_t send(uint16_t address, uint8_t length, TOS_MsgPtr msg);

event result_t sendDone(TOS_MsgPtr msg, result_t success);

}
```

Appeler une commande et signaler un événement

Appeler une commande

```
call Send.send(1, sizeof(Message), &msg1);
```

Signaler un event

```
signal Send.sendDone(&msg1, SUCCESS);
```

Modules

Un module est un composant qui implemente une ou plusieurs interfaces et peut utiliser une ou plusieurs interfaces

```
module Provider
{
   provides interface StdControl;
   provides interface X;
   uses interface Z;
}
implementation
{
    // C code
    ....
}
```

Syntaxe d'un module

Schamatisation de la déclaration de trois modules

module C1 {

requires interface triangle;

} implementation { ... }

module C2 {

provides interface triangle in;

requires { interface triangle out; interface rectangle side; }

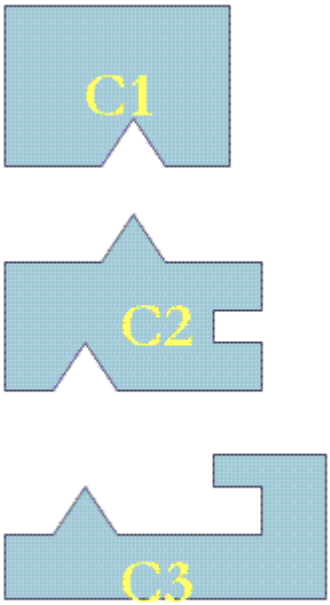
} implementation { ... }

module C3 {

provides interface triangle;

provides interface rectangle;

} implementation { ... }



Exemple de déclaration de modules

Configurations

Dans nesC, deux composants sont reliés ensemble en les connectant (wiring). Les interfaces du composant utilisateur sont reliées (wired) aux mêmes interfaces du composant fournisseur. Il existe 3 possibilités de connexion (wiring statements) dans nesC:

- endpoint1 = endpoint2
- endpoint1 -> endpoint2
- endpoint1 <- endpoint2 (equivalent: endpoint2 -> endpoint1)

Les éléments connectés doivent être compatibles : Interface à interface, "command" à "command", "event" à "event". Il faut toujours connecter un utilisateur d'une interface à un fournisseur de l'interface.

Illustration d'une configuration

```
configuration app { }

implementation {

uses c1, c2, c3;

c1 -> c2; // implicit interface selection.
```

c2.out -> c3.triangle;

c3 <- c2.side;

}

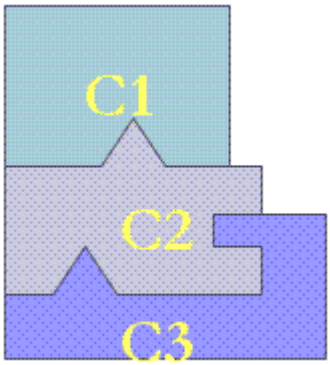


Illustration d'une configuration basée sur la connexion de 3 modules

Notion de tâche et concurrence dans nesC

Tâche

Une tâche est un élément de contrôle indépendant défini par une fonction retournant void et sans arguments:

task void myTask() { ... }

Les tâches sont lancées en les préfixant par post:

post myTask();

Commands vs. Events vs. Tasks

Command

- Ne doit pas être bloquante i.e. prend les paramètres, commence le traitement et retourne dans l'application;
- Reporte le travail qui consomme du temps en postant une tâche
- Peut appeler des commandes sur d'autres composants

Event

- Peut appeler des commandes, signaler d'autres événements, poster des tâches mais ne peut pas être signalé par des commandes
- Peut interrompre une tâche mais pas l'inverse.

Task

- Ordonnancement FIFO
- Non préemptive par une autre tâche, préemptive par un événement
- Utilisée pour réaliser un travail qui nécessite beaucoup de calculs
- Peut être postée par une "command" ou un "event".

Instruction "atomic"

L'instruction "atomic" guarantie que l'exécution de l'instruction se fait comme si aucun autre calcul ne se fait simultanément. Une instruction "atomic" doit être courte. nesC interdit dans une instruction atomique: call, signal, goto, return, break, continue, case, default, label

Accès à une ressource critique avec "atomic"

```
bool busy; // global

void f() {

bool available;

atomic {

available = !busy;

busy = TRUE;

}

if (available) do_something;

atomic busy = FALSE;

}
```

Compilation et exécution

Le compilateur traîte les fichiers nesC en les convertissant en un fichier C gigantesque. Ce fichiet contiendra l'application et les composants de l'OS utilisés par l'application. Ensuite un compilateur spécifique à la plateforme cible compile ce fichier C. Il deviendra alors un seul exécutable. Le chargeur installe le code sur le "mote" (Mica2, Telos, etc.)

Convention de nommage en nesC

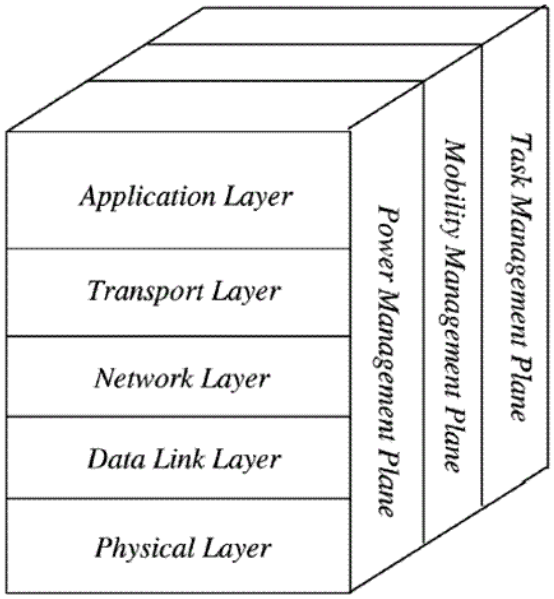
- Extension des fichiers nesC: .nc

- Clock.nc : soit une interface (ou une configuration)
- ClockC.nc : une configuration
- ClockM.nc : un module

3.4 La communication dans les RCSF

3.4.1 Modèle en couches

Le rôle de ce modèle consiste à standardiser la communication entre les composants du réseau afin que différents constructeurs puissent mettre au point des produits (logiciels ou matériels) compatibles. Ce modèle comprend 5 couches qui ont les mêmes fonctions que celles du modèle OSI ainsi que 3 couches pour la gestion de la puissance d'énergie, la gestion de la mobilité ainsi que la gestion des tâches (interrogation du réseau de capteurs). Le but d'un système en couches est de séparer le problème en différentes parties (les couches) selon leur niveau d'abstraction. Chaque couche du modèle communique avec une couche adjacente (celle du dessus ou celle du dessous). Chaque couche utilise ainsi les services des couches inférieures et en fournit à celle de niveau supérieur.



Modèle en couches pour la communication dans les RCSF

Rôle des couches

- La couche physique : Spécifications des caractéristiques matérielles, des fréquences porteuses, etc...

- La couche liaison : Spécifie comment les données sont expédiées entre deux noeuds/routeurs dans une distance d'un saut. Elle est responsable du multiplexage des données, du contrôle d'erreurs, de l'accès au media,... Elle assure la liaison point à point et multi-point dans un réseau de communication.
- La couche réseau : Dans la couche réseau le but principal est de trouver une route et une transmission fiable des données, captées, des noeuds capteurs vers le puits "sink" en optimisant l'utilisation de l'énergie des capteurs. Ce routage diffère de celui des réseaux de transmission ad hoc sans fils par les caractéristiques suivantes:
    - il n'est pas possible d'établir un système d'adressage global pour le grand nombre de nSuds.
    - les applications des réseaux de capteurs exigent l'écoulement des données mesurées de sources multiples à un puits particulier.
    - les multiples capteurs peuvent produire de mêmes données à proximité d'un phénomène (redondance).
    - les nSuds capteur exigent ainsi une gestion soigneuse des ressources.

  En raison de ces différences, plusieurs nouveaux algorithmes ont été proposés pour le problème de routage dans les réseaux de capteurs

- La couche transport : Cette couche est chargée du transport des données, de leur découpage en paquets, du contrôle de flux, de la conservation de l'ordre des paquets et de la gestion des éventuelles erreurs de transmission.
- La couche application : Cette couche assure l'interface avec les applications. Il s'agit donc du niveau le plus proche des utilisateurs, géré directement par les logiciels.

3.4.2 Plans de gestion

Les plans de gestion d'énergie, de mobilité et de tâche contrôlent l'énergie, le mouvement et la distribution de tâche au sein d'un nSud capteur. Ces plans aident les nSuds capteurs à coordonner la tâche de captage et minimiser la consommation d'énergie. Ils sont donc nécessaires pour que les nSuds capteurs puissent collaborer ensemble, acheminer les données dans un réseau mobile et partager les ressources entre eux en utilisant efficacement l'énergie disponible. Ainsi, le réseau peut prolonger sa durée de vie.

- Plan de gestion d'énergie : contrôle l'utilisation de la batterie. Par exemple, après la réception d'un message, le capteur éteint son récepteur afin d'éviter la duplication des messages déjà reçus. En outre, si le niveau d'énergie devient bas, le nSud diffuse à ses voisins une alerte les informant qu'il ne peut pas participer au routage. L'énergie restante est réservée au captage ;
- Plan de gestion de mobilité : détecte et enregistre le mouvement du nSud capteur. Ainsi, un retour arrière vers l'utilisateur est toujours maintenu et le nSud peut garder trace de ses nSuds voisins. En déterminant leurs voisins, les nSuds capteurs peuvent balancer l'utilisation de leur énergie et la réalisation de tâche ;
- Plan de gestion de tâche : balance et ordonnance les différentes tâches de captage de données dans une région spécifique. Il n'est pas nécessaire que tous les nSuds de cette région effectuent la tâche de captage au même temps ; certains nSuds exécutent cette tâche plus que d'autres selon leur niveau de batterie.

4 Description de l'étude réalisée et du travail effectué

Nous disposons d'une carte imote2 qui sert comme une station de base reliant avec l'ordinateur en série et avec les autres cartes imote2 appelées nœud.

Pour réaliser l'interconnexion, nous commençons par implanter la connexion d'un nœud à la station de base

4.1 Réseau de capteurs sans fil

Un **réseau de capteurs sans fil** est un réseau ad hoc avec un grand nombre de *nœuds* qui sont des micro-capteurs capables de récolter et de transmettre des données environnementales d'une manière autonome. La position de ces nœuds n'est pas obligatoirement prédéterminée. Ils peuvent être aléatoirement dispersés dans une zone géographique, appelée *« champ de captage »* correspondant au terrain d'intérêt pour le phénomène capté.

4.2 Architecture de réseau de capteurs sans fils

Un RCSF est composé d'un ensemble de nœuds capteurs. Ces nSuds capteurs sont organisés en champs « sensor fields » (voir figure suivante). Chacun de ces noeuds a la capacité de collecter des données et de les transférer au nSud passerelle (dit "sink" en anglais ou puits) par l'intermédiaire d'une architecture multi-sauts. Le puits transmet ensuite ces données par Internet ou par satellite à l'ordinateur central «Gestionnaire de taches» pour analyser ces donner et prendre des décisions.

4.3 Radios et Antenne

Comme nous avons introduit, la carte imote2 intègre un émetteur-récepteur radio 802.15.4 de Chipcon (CC2420). 802.15.4 est un standard IEEE décrivant les couches physiques et MAC d'une portée radio de faible puissance faible, destinés à des applications de contrôle et de surveillance. Le CC2420 prend en charge un ko 250 / s de données avec les taux de 16 canaux dans la bande 2,4 GHz. Autres modules radio externes tels que 802.11 et Bluetooth peut être activée via les interfaces supportées (SDIO, UART, SPI, etc).

Le protocole qui est alors utilisé pour faire la diffusion radio est ZigBee

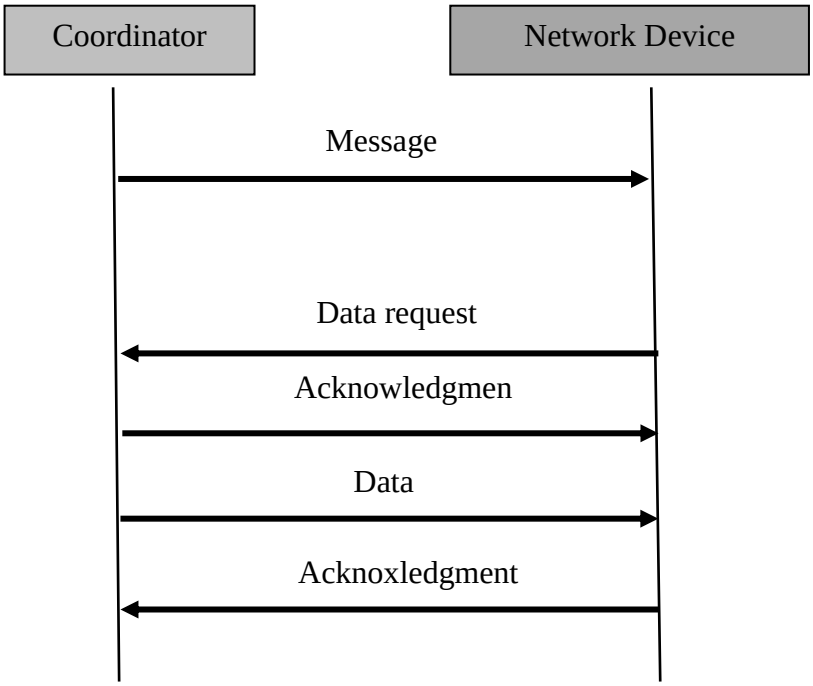
4.4 Protocole ZigBee

ZigBee est un protocole de haut niveau permettant la communication de petites radios, à consommation réduite, basée sur la norme IEEE 802.15.4 pour les réseaux à dimension personnelle (Wireless Personal Area Networks : WPANs) en utilisant le système d'échange de parquet défini par sa norme IEEE 802.15.4

Standard	Débit	Portée	Consommation en transmission	Consommation en veille	Taille de la pile protocole
ZigBee	20-250 kbit/s	100 m	20 mA	< 0.01 mA	< 32KO

Bande de fréquences ZigBee	Débit	Canaux
902-928 MHz	40 kbit/s	1-10

4.5 Communication entre la station de base et le réseau capteurs (NesC)



La communication entre la station de base et les capteurs a été gérée internement dans les capteurs par le programme codé en NesC, qui a été compilé et installé dans les capteurs, ce programme a deux parties différentes, une pour la station de base et une autre pour le nœud. Ces programmes permettent d'envoyer des messages en bidirectionnel entre les capteurs et de manipuler des données d'échanges. Par exemple le parquet de données qui a été envoyé entre les capteurs et qui a été transféré vers l'ordinateur pour pouvoir être décodé en format PPM et PGM grâce à l'application java.

Mais un problème que l'on peut se poser ici, est de savoir quel type de donné d'échange entre les capteurs. Pour cela, nous avons défini une interface de messages dans le fichier « Sendbigmsg.h » (code d'annexes). La station de base va envoyer un signal de demande pour le nœud, le nœud a un rôle de traduire ce message et faire la capture d'image, puis il renvoie le paquet données. La base station reconnait un message arrivant sur la liaison série que si ce message a été mis en file pour la livraison à la liaison radio. Évidemment, nous avons prévu les voyants LED pour reconnaitre la transmission.

4.6 Réception de donnée et affichage (Java)

Comme le protocole que l'on utilise est de type radio, la station de base va recevoir l'information dans l'ordre FIFO, et traiter, ensuite elle transfère en liaison série vers l'ordinateur. Ici, l'application Java gère le donné reçu et affiche à l'écran l'image capturée. Les codes sources java on n'a pas changé bien qu'ils avaient été fait pour la manipulation de donné en liaison série avec PC. Les données reçus vont toujours être les mêmes avec fil ou sans fil. Ces donnes sont reçus pour le port USB et aussi envoyés pour le mini USB de la carte IIB2400 qui est l'interface board du intelmote2, en sachant que cette mini USB ne travaille pas à 100% de la vitesse, bien que pour l'intelmote 2 dans le environnement de TinyOs n'est pas supporté le mini USB, donc on fait un échange de l'interface et la mini USB est pris comme une port serial, et c'est la raison du fait que l'affichage de données dans l'ordinateur est lent.

4.7 Réception de données multi-hop

Pour cette partie, en raison du manque de temps, notre travail n'est pas complétement terminé, il faut gérer la réception de donnée de la station de base surtout pour les paquets retard ou perdu de plusieurs nœuds.

5 Conclusion

Ce projet a apporté à nous, une nouvelle forme pour approfondir notre connaissance, c'est le système embarqué. La plus part de travail a été fait dans un langage de programmation NesC que nous n'avions pas connu avant. Ça nous donne de la compétence pour pouvoir nous adapter à des différents environnements de travail ainsi que la possibilité de travailler avec les matériels nous donne des expériences dans la vie professionnelle de l'avenir.

D'autre part, le sujet du projet est très intéressant parce que les réseaux de capteurs sans fils sont vraiment es pleine expression de nos jours, mais encore trop peu des personnes extérieurs à ce domaine.

6 Bibliographie

- Imote2 http://docs.tinyos.net/index.php/Imote2
- Network Types: Language Support for Messaging in the Heterogeneous Networks http://nescc.sourceforge.net/networktypes/index.html
- nesC: A Programming Language for Deeply Networked Systems http://nescc.sourceforge.net/
- Installation & Programming Guide for Imote2 (Intelmote2) on TinyOS 2.x http://web.ics.purdue.edu/~paulshin/research_Imote2.html
- Imote2 Hardware Reference Manual http://www.memsic.com/support/documentation/wireless-sensor-networks/category/6-user-manuals.html?download=56%3Aimote2-hardware-reference-manual

7 Annexes.

Code en NesC

- Caméra nœud

```
module cameraNodeM {
  uses {
    interface Boot;
    interface Leds;
    interface Timer<TMilli> as Timer0;
    interface XbowCam;
    interface Init as CameraInit;

    interface SendBigMsg;

    // Radio interfaces
    interface SplitControl as RadioControl;
    interface Packet;
    interface Receive as CmdReceive;
    interface AMSend as ImgStatSend;

  }
}
```

```
typedef nx_struct img_stat{
nx_uint8_t type;
nx_uint16_t width;
nx_uint16_t height;
nx_uint32_t data_size;
nx_uint32_t timeAcq;
nx_uint32_t timeProc;
nx_uint32_t tmp1;
nx_uint32_t tmp2;
nx_uint32_t tmp3;
nx_uint32_t tmp4;
} img_stat_t;

typedef nx_struct cmd_msg{
nx_uint8_t cmd;
nx_uint16_t val1;
nx_uint16_t val2;
} cmd_msg_t;
```

- Station de base

```
module BaseStationM{

  uses {
    interface Boot;
    interface SplitControl as SerialControl;
    interface SplitControl as RadioControl;

// Interfaces MOTE-PC
    interface AMSend as UartSend[am_id_t id];
    interface Receive as UartReceive[am_id_t id];
    interface Packet as UartPacket;
    interface AMPacket as UartAMPacket;

    // Interfaces MOTE-MOTE
    interface AMSend as RadioSend[am_id_t id];
    interface Receive as RadioReceive[am_id_t id];
    interface Packet as RadioPacket;
    interface AMPacket as RadioAMPacket;

    interface Leds;
  }
}
```

- BigMessage

```
typedef nx_struct bigmsg_frame_part{
nx_uint16_t part_id;
nx_uint8_t buf[BIGMSG_DATA_LENGTH];
} bigmsg_frame_part_t;

typedef nx_struct bigmsg_frame_request{
nx_uint16_t part_id;
nx_uint16_t send_next_n_parts;
} bigmsg_frame_request_t;
```

```
module SendBigMsgM
{
  provides
  {
    interface SendBigMsg;
    interface Init;
  }

  uses
  {
    interface Boot;
    interface SplitControl as CollectionControl;
    interface AMSend as FrameSend;
    interface Leds;
    interface Timer<TMilli> as Timer1;
  }
}
```

Diagramme de PERT (1)

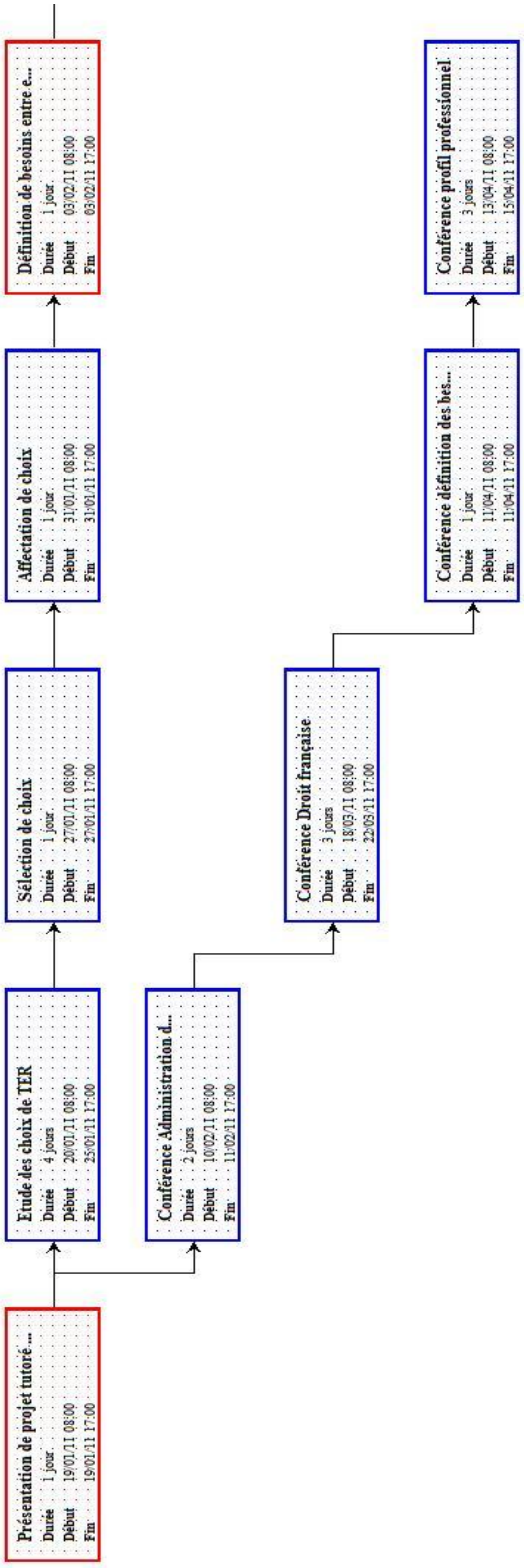


Diagramme de PERT (2)