République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté desSciences
Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option: *Réseaux et Systèmes Distribués (R.S.D)*

Thème

Acquisition de données à distance dans les réseaux de capteurs sans fil

Réalisé par :

- ***M^r^ BADAOUI Abdellatif***
- ***M^r^ MOSTEFAOUI Mohamed Amine***

Présenté le 02 Juillet 2013 devant le jury composé de:

- **M^me^ DIDI Fadoua** (Présidente)
- **M^r^ LEHSAINI Mohamed** (Encadreur)
- **M^me^ LABRAOUI Nabila** (Examinateur)
- **M^r^ BELHOUCINE Amine** (Examinateur)

Année universitaire:2012-2013

Remerciements

Nous remercions notre Dieu le tout puissant qui nous a accordé la volonté, la patience, et surtout la santé durant tout notre cursus.

Remercier, c'est le plaisir de se souvenir de tous ceux qui, par leurs Encouragements, leur disponibilité, leur amitié et leurs compétences, ont su créer une ambiance de travail nous ayant permis de finaliser ce mémoire.

Nos plus sincères remerciements vont :

A notre encadreur «**LEHSAINI MOAHMED**» pour ses conseils, son aide, pour son soutien moral et scientifique efficace et constant, durant toute cette durée de travail.

A tous les membres du jury qui nous fait l'honneur de prendre notre modeste travail en considération et en suite de le juger.

Nos profonds remerciements à nos parents qui nous ont encouragés, qui nous ont appris à travailler honnêtement, et nous ont réalisé tous les moyens afin d'apprendre.

Que tous ceux que nous avons cités et ceux dont on a omis les noms, trouvent ici l'expression de notre grande gratitude.

Merci Beaucoup

Dédicaces

Avec un énorme plaisir, un cœur ouvert et une Immense joie, Que nous dédions notre travail à nos très chers, Respectueux et Magnifiques parents qui nous ont soutenu tout au long de notre vie ainsi à nos frères et nos sœurs.

A toutes personnes et amis qui nous ont encouragé ou aidé au long De nos études.

En souvenir des moments passés ensemble, nous dédions ce travail en témoignage de nos amitié sincère et durable.

Nous vous souhaitons un avenir radieux et plein de réussite….

Table des matières

Introduction Générale

Toujours dans le but de faciliter la vie et de faire éliminer les difficultés autour de l'être humain, plusieurs technologies sont apparues pour garantir ces besoins, l'une de ces technologies est les réseaux de capteurs sans fil (RCSF) qui constituent un des domaines les plus actifs pour la recherche, car ils offrent des solutions économiquement intéressantes et facilement déployables à la surveillance à distance et au traitement des données dans les environnements hostiles.

Un réseau de capteurs sans fil est un ensemble de capteurs autonomes à faible coût, interconnectés par un réseau de communications sans fil, capables d'effectuer des mesures sur l'environnement pour construire une vue globale de la région contrôlée. Son but est la collecte d'un ensemble de grandeurs environnementales, physiques ou physiologiques entourant ces capteurs, telles que la température, l'humidité, glycémie, ou tension, etc.., afin de les acheminer vers des centres de contrôle.

Les RCSF sont déployés pour deux types d'applications: les applications orientées surveillance ou applications orientées événement. Dans le premier type de ces applications, les informations sont détectées et remontées aux centres de contrôle d'une manière périodique alors que dans le deuxième type d'applications, l'information devrait être remontée rapidement lors de l'occurrence d'un événement pertinent. En outre, dans les applications orientées événement, la manière d'alerter devrait être pratique et pourrait atteindre le personnel chargé de la surveillance à n'importe quel endroit. Dans ce contexte, nous avons proposé deux manières pour alerter un personnel distant de la zone de déploiement des capteurs. Dans le premier type d'alertes, nous exploitons le réseau Internet pour aviser le centre de contrôle alors que dans le deuxième type, nous utilisons le réseau téléphonique (GSM) et le personnel sera informer par SMS[1].

Notre travail consiste à visualiser les données acheminées des capteurs jusqu'au centre de contrôle en passant le réseau Internet ou le réseau téléphonique.

Pour relater les travaux réalisés dans le cadre de notre projet, notre mémoire s'articule autour des chapitres suivants:

[1]*Short Message Service* introduit par la norme GSM

Dans le premier chapitre, nous présentons les réseaux de capteurs sans fil, leurs caractéristiques et leurs concepts. Le deuxième chapitre est une description des ressources matérielles et logicielles pour le développement des applications conçues pour les RCSF en particulier nous décrivons les capteurs de type Telosb, le système d'exploitation TinyOs et le langage orienté composants NesC. Dans le troisième chapitre, nous présentons notre application. Enfin, nous terminons par une conclusion générale et nous proposons quelques perspectives.

Chapitre I.
Réseaux de capteurs sans fil : Concepts et Applications

I.1 Introduction

Les concepts des réseaux de capteurs sont bien particuliers comparativement aux réseaux sans fil puisque les RCSF sont composés d'équipements à ressources limitées en termes de calcul, de stockage et d'énergie.

Dans ce chapitre, nous présentons les réseaux de capteurs sans fils, leurs caractéristiques et leurs domaines d'applications.

I.2 Présentation des réseaux de capteurs sans fil

Un réseau de capteurs sans fil est un réseau ad hoc[2] avec un grand nombre de nœuds qui sont des micro-capteurs, aléatoirement dispersés dans une zone géographique, interconnectés entre eux par le biais d'un réseau sans fil de type Zig Bee[3], capables de récolter et de transmettre des données environnementales d'une manière autonome, telles que la température, l'humidité et luminosité …etc. Les données mesurées collectées par ces capteurs sont acheminées directement ou via les autres capteurs de proche en proche jusqu'à l'aboutissement à la station de base qui est lui-même un capteur relié directement au centre de contrôle. La station de base est considérée comme un capteur qui a plus de capacité en termes d'énergie. La figure 1 illustre un exemple de déploiement d'un réseau de capteurs.

[2] Un réseau ad-hoc est un réseau sans fil capable de s'organiser sans la présence d'infrastructure réseau.

[3] **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.

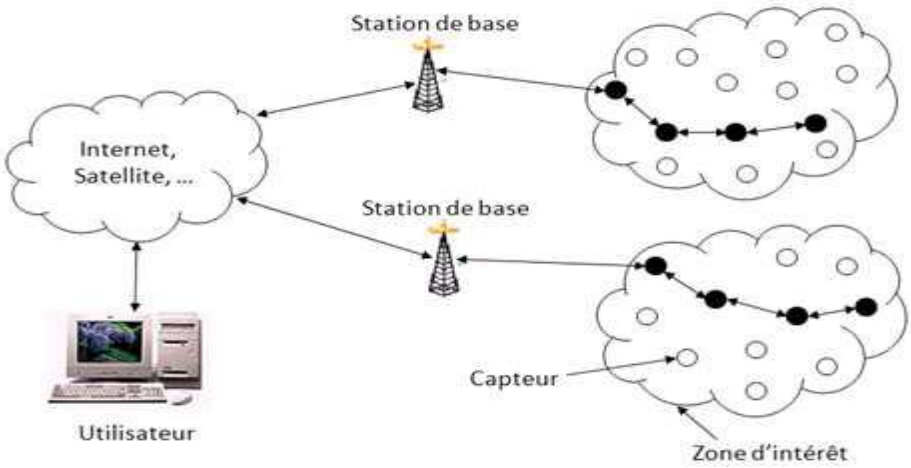

Figure 1: Exemple d'un réseau de capteurs [4]

I.3 Présentation du nœud capteur

Un nœud capteur (dit "mote" en anglais) est un nœud qui constitue l'unité de base d'un RCSF. Il est composé principalement d'une unité de calcul, une mémoire, un émetteur/récepteur radio, un ensemble de capteurs, et une pile comme s'est illustré dans la figure 2.

Il existe plusieurs modèles de capteurs qui sont commercialisés dans le marché. Parmi les plus célèbres, la famille des Mica tels que MICAZ et Mica2et la famille des Telos tels que Telosa et Telosb de Crossbow [1]. La figure 3 montre un capteur intelligent Telosb.

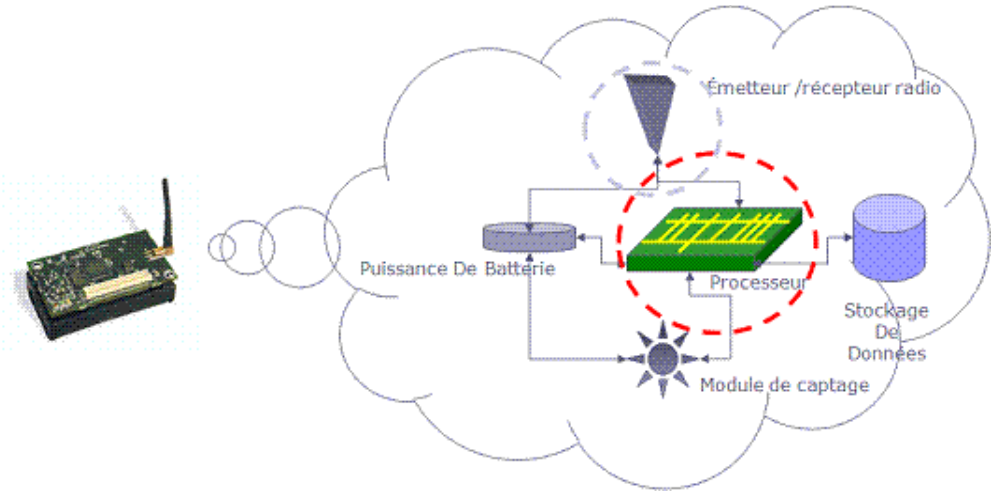

Figure 2: Anatomie d'un nœud capteur

Figure 3: Exemple de capteur intelligent Telosb de Crossbow [1]

I.4 Architecture physique d'un capteur

Un capteur est composé de trois unités [2]:

- **L'unité d'acquisition:** L'unité d'acquisition est composée d'un capteur qui va obtenir des mesures numériques sur les paramètres environnementaux et d'un convertisseur Analogique/Numérique (ADC[4]) qui permet de convertir l'information relevée à l'état brute et la transmettre à l'unité de traitement pour être interprétable soit sous forme d'une valeur numérique ou représentée par un signal.
- **L'unité de traitement:** L'unité de traitement est composée de deux interfaces, une interface pour l'acquisition de données et une autre pour leur transmission. Cette unité est également composée d'un processeur supportant un système d'exploitation spécifique. Elle récupère les informations en provenance de l'interface d'acquisition et les envoie à l'interface de transmission.
- **L'unité de transmission (ou de communication):** Cette unité est responsable de toutes les émissions et réceptions de données via un support de communication radio. Ces trois unités sont alimentées par une batterie comme le montre la figure 4 ci-dessous.

[4] ADC: Analog/Digital Converter

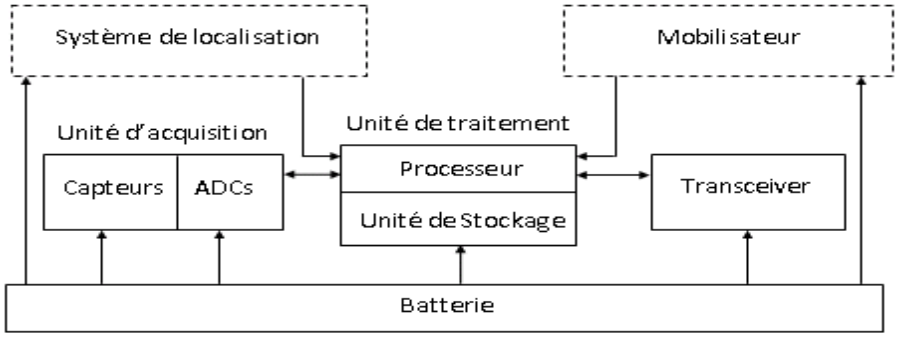

Figure 4: Architecture physique d'un capteur [4]

I.5 Caractéristiques des RCSF

Les RCSF présentent des caractéristiques particulièrement comparativement aux réseaux sans fil. Parmi les principales caractéristiques, nous citons:

- **Absence d'infrastructure:** Les réseaux de capteurs se distinguent des autres réseaux mobiles par la propriété d'absence d'infrastructure préexistante et de tout genre d'administration centralisée. Lors de la détection de l'information par un capteur, cette dernière sera communiquée aux voisins de proche en proche en utilisant un mode multi-sauts jusqu'à l'aboutissement à la station de base.
- **Déployés en grand nombre:** Les RCSF sont généralement déployés en grand nombre pour assurer la couverture de la zone d'intérêt d'une part et d'autre part pour faire face à la tolérance aux pannes puisque les capteurs sont sujets à des pannes telles que l'épuisement de leur énergie ou écrasement par des animaux. Nous pouvons avoir dans de telles applications, des réseaux de capteurs comprenant des milliers voire des millions de capteurs.
- **Contrainte d'énergie:** Le principal facteur limitant la durée de vie d'un réseau de capteurs est l'énergie [3].Dans plusieurs applications, les nœuds capteurs sont placés dans des zones hostiles là où l'accès de l'homme est difficile voire impossible. De ce fait, le rechargement et le remplacement des batteries deviennent des tâches difficiles. D'où, nous devrons mettre en place un mécanisme pour préserver l'énergie et par suite garantir une longue durée de vie pour ces réseaux.
- **Topologie dynamique:** Les capteurs peuvent être attachés à des objets mobiles qui se déplacent d'une façon libre et arbitraire rendant ainsi, la topologie du réseau fré-

quemment changeante. Par exemple, des capteurs supportés par des animaux pour les surveiller.

- **Auto-organisation du réseau:** Cette caractéristique peut être nécessaire dans plusieurs cas. Par exemple, un réseau comportant un grand nombre de nœuds placés dans des endroits hostiles où la configuration manuelle n'est pas faisable, doit être capable de s'auto-organiser. Un autre cas est celui où un nœud est inséré ou retiré (à cause d'un manque d'énergie ou de destruction physique). Ainsi le réseau doit être capable de se reconfigurer pour continuer sa fonction.
- **Sécurité physique limitée:** Les RCSF sont plus touchés par le paramètre de sécurité que les réseaux filaires classiques. Cela se justifie par les contraintes et limitations physiques qui font le contrôle des données transférées doit être minimisé.

I.6 Applications des RCSF

La miniaturisation des capteurs, le coût de plus en plus faible, la large gamme des types de capteurs disponibles ainsi que le support de communication sans fil utilisé, permettent aux réseaux de capteurs de se développer dans plusieurs domaines d'applications. La figure 5 montre quelques domaines d'applications des RCSF.

- **Le bâtiment:** L'évolution de la structure d'un ouvrage d'art, la gestion de la température et de la lumière dans un immeuble, la domotique, les interrupteurs autonomes non câblés, etc. constituent quelques exemples d'applications dans le domaine du bâtiment.
- **Les transports:** La gestion du trafic, la déformation de structure, les capteurs de pression des pneus, etc. sont des exemples d'applications de capteurs dans le domaine du transport.
- **L'environnement:** Dans le domaine de l'environnement, nous pouvons citer: la détection de polluants dans l'air ou le sol, le suivi des mouvements des oiseaux, des animaux et des insectes, la détection des incendies, la détection du niveau d'eau dans le sol, etc.
- **Le médical:** Le domaine médical peut lui aussi intégrer des applications pertinentes. Comme par exemple : l'aide à la médication et le suivi des patients à distance (rythme cardiaque, pression du sang, etc.), l'identification des allergies et des médi-

caments administrés aux patients, la localisation des docteurs et des patients dans l'hôpital, etc.

- **Le militaire:** Le domaine militaire ne sera pas épargné non plus. Il pourra utiliser les RCSF par exemple dans la détection et la collecte d'informations sur la position de l'ennemi et ses mouvements, la détection d'agents chimiques ou bactériologiques, etc.

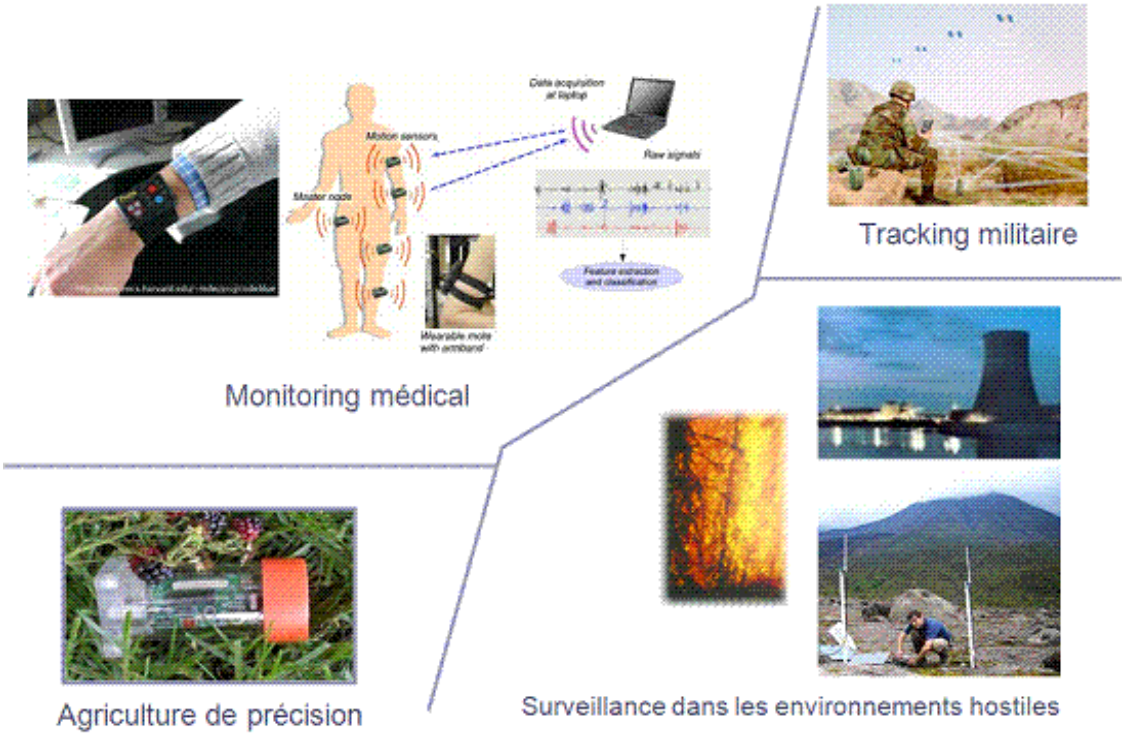

Figure 5: Applications des réseaux de capteurs sans fil [5]

I.7 Conclusion

Dans ce chapitre, nous avons présenté les réseaux de capteurs, leurs caractéristiques et leurs domaines d'applications. Ces caractéristiques nous permettent de préparer un environnement bien particulier pour développer des applications. Cet environnement contient des outils logiciels dédiés aux équipements à ressources limitées à l'instar des capteurs.

Dans le chapitre qui suit, nous présentons ces outils spécifiques au développement d'applications sur des capteurs.

Chapitre II.
Outils de développement pour les réseaux de capteurs sans fil

II.1 Introduction

Dans ce chapitre nous décrivons les outils nécessaires pour le développement des applications des RCSF, tels que les systèmes d'exploitation en particulier TinyOS et le langage de programmation Nesc.

II.2 Système d'exploitation: TinyOS

II.2.1 Présentation de TinyOS

TinyOs [6] est un système principalement développé et soutenu par l'université américaine de Berkeley. C'est un système d'exploitation open-source conçu pour des réseaux de capteur sans fil. Il respecte une architecture basée sur une association de composants permettant de réduire la taille du code nécessaire à sa mise en place. Cela s'inscrit dans le respect des contraintes de mémoire qu'observe les réseaux de capteurs.

Pour autant, la bibliothèque de composants de TinyOs est particulièrement complète puisqu'on y retrouve des protocoles réseaux, des pilotes de capteurs et des outils d'acquisition de données. L'ensemble de ces composants peuvent être utilisés tels quels, puisqu'ils peuvent être adaptés à une application précise.

En s'appuyant sur un fonctionnement événementiel, TinyOs propose à l'utilisateur une gestion très précise de la consommation d'énergie des capteurs et permet de mieux s'adapter à la nature aléatoire de la communication sans fil entre interfaces physiques.

II.2.2 Propriétés de l'OS

Le fonctionnement d'un système basé sur TinyOs s'appuie sur la gestion des évènements. Ainsi, l'activation de tâches, leur interruption ou encore la mise en veille du capteur s'effectue à l'apparition d'évènements, ceux-ci ayant la plus forte priorité. Ce

fonctionnement évènementiel (event-driven) s'oppose au fonctionnement dit temporel (time-driven) où les actions du système sont gérées par une horloge donnée.

TinyOs a été programmé en langage Nesc. Le caractère préemptif d'un système d'exploitation précise si celui-ci permet l'interruption d'une tâche en cours ou non. TinyOs ne gère pas ce mécanisme de préemption entre les tâches mais donne la priorité aux interruptions matérielles. Ainsi, les tâches entre-elles ne s'interrompent pas mais une interruption peut stopper l'exécution d'une tâche.

Lorsqu'un système est dit « temps réel » celui-ci gère des tâches caractérisées par des priorités et par des échéances à respecter dictées par l'environnement externe. Dans le cas d'un système strict, aucune échéance ne tolère de dépassement contrairement à un système temps réel mou. TinyOs se situe au-delà de ce second type car il n'est pas prévu pour avoir un fonctionnement temps réel. TinyOs a été conçu pour réduire au maximum la consommation en énergie du capteur. Ainsi, lorsqu'aucune tâche n'est pas active, il se met automatiquement en veille.

II.2.3 Caractéristiques de TinyOs

Dans cette section, on présente quelques caractéristiques de TinyOS.

- **Concurrence:** TinyOS utilise une architecture orientée événement.
- **Modularité**
  - Application composée de composants.
  - OS + Application compilées en un seul exécutable.
- **Communication**
  - Utilise un modèle event/command.
  - Ordonnancement FIFO non préemptif
- **Pas de séparation noyau/utilisateur.**

II.2.4 Structure logicielle

Le système d'exploitation TinyOS s'appuie sur le langage NesC. Celui-ci propose une architecture basée sur des composants, permettant de réduire considérablement la taille mémoire du système et de ses applications. Chaque composant correspond à un

élément matériel (Leds, Timer,…) et peut être réutilisé dans différentes applications. Ces applications sont des ensembles de composants réutilisables et portables associés dans un but précis. Chaque composant est tout d'abord constitué d'un frame qui contient l'état interne du composant. Il s'agit d'un espace mémoire de taille fixe permettant au composant de stocker les variables globales et les données qu'il utilise pour réaliser ses fonctionnalités. Il n'en existe qu'une seule par composant et celle-ci est allouée statiquement à la compilation.

L'implémentation des composants s'effectue en déclarant des tâches, des commandes ou des événements qui permettent de faire appel aux fonctionnalités d'autres composants auxquels ils sont liés, voire la figure 6.

- **Les tâches** sont utilisées pour effectuer la plupart des blocs d'instructions d'une application.
- **Une commande** permet l'exécution d'une fonctionnalité dans un autre composant.
- **Les événements** permettent de faire le lien entre les interruptions matérielles (pression d'un bouton, changement d'état d'une entrée,...) et les couches logicielles que constituent les tâches.

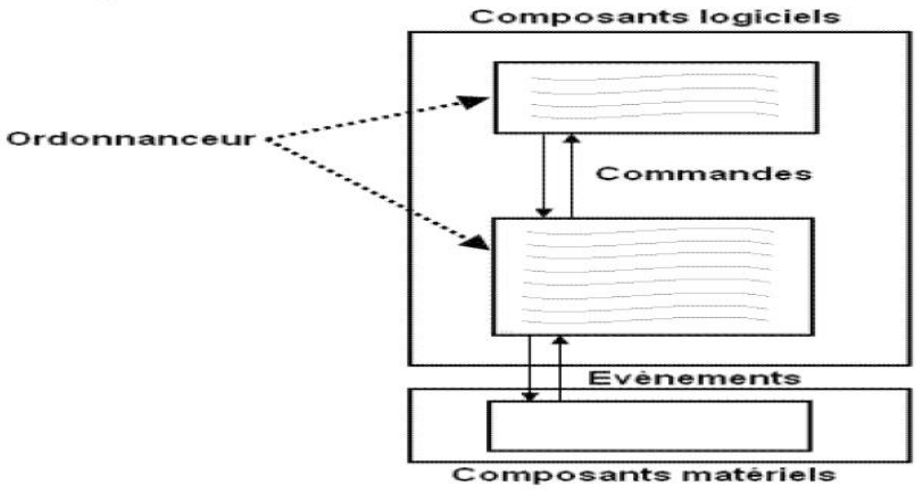

Figure 6: Schéma des interactions internes au système TinyOS

II.3 Le langage de programmation NesC

Le langage NesC (Network embedded system C) est un dialecte de C basé sur les composants [9]. Il est orienté pour satisfaire les exigences des systèmes embarqués. De plus, il supporte un modèle de programmation qui agrège l'administration des commu-

nications, les concurrences provoquant les tâches et les évènements ainsi que la capacité de réagir par rapport à ces évènements.

NesC réalise aussi une optimisation dans la compilation du programme, en détectant les carrières possibles de données qui peuvent produire des modifications concurrentes à une même section de mémoire (concurrence d'accès mémoire entre threads), et quand au moins l'un des accès est en écriture. NesC simplifie aussi le développement d'applications et réduit la taille du code.

II.3.1 Les caractéristique de NesC

NesC est constitué d'interfaces et de composants. Une application développée en NesC, est un ensemble de composants, regroupés et reliés entre eux comme s'est illustré par la figure 7.

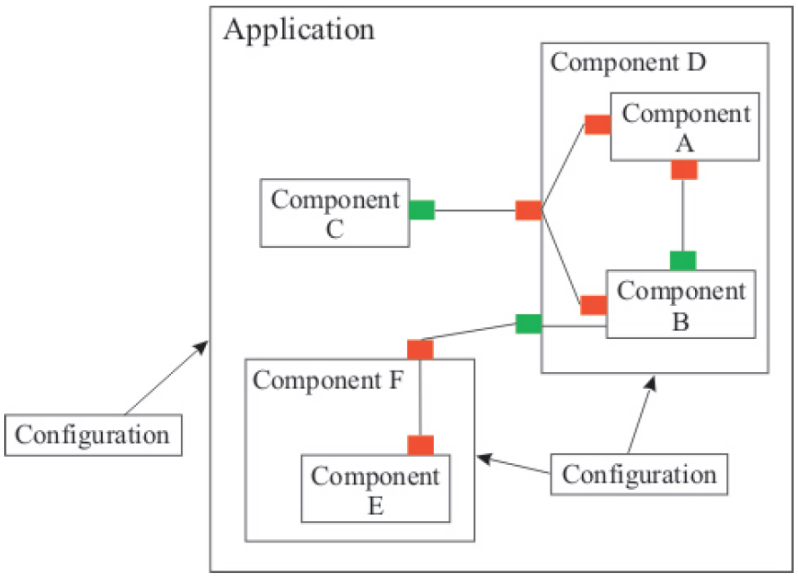

Figure 7: Architecture d'une application NesC

Les interfaces sont utilisées pour les opérations qui décrivent l'interaction bidirectionnelle. Le fournisseur de l'interface doit mettre en application des commandes, alors que l'usager de l'interface doit mettre en application des évènements.

Il existe deux types de composants:

- **Les modules**, qui mettent en application des spécifications d'un composant.
- **Les configurations**, qui se chargent d'unir différents composants en fonction des interfaces (commandes ou évènements).

La figure8 montre un diagramme de blocs dans lequel est décrit le processus de compilation pour une application TinyOS écrite en NesC.

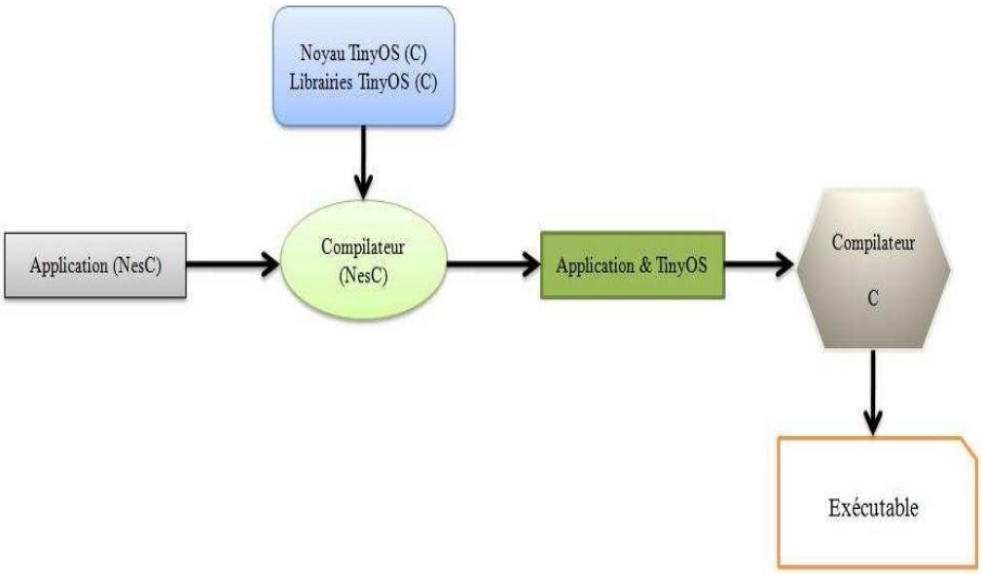

Figure 8: Processus de Compilation d'une application sous TinyOS [7]

II.3.2 Les Fichiers dans NesC

Les fichiers dans NesC sont classés en trois types: Interfaces, Modules et Configurations.

- **Interface:** Ce type de fichiers déclare les services fournis et les services qui seront utilisés. Elle définit les interactions entre deux composants [5]. Ils se trouvent dans le répertoire /tos/interface.

  Exemple: StdControl.nc présenté dans l'annexe II.

- **Module:** Le type Module contient le code de l'application, en mettant en œuvre une ou plusieurs interfaces.

  Exemple : BlinkM.nc présenté dans l'annexe II.

- **Configuration:** Dans ces fichiers est déclarée la manière d'unir les différents composants et comment effectuer le contrôle des flux.
  Exemple: Blink.nc présenté dans l'annexe II.

II.3.3 Eléments d'un programme NesC

a. Composants

TinyOS définit un nombre important de concepts qui sont exprimés dans NesC. D'abord, les applications NesC sont construites par des composants avec des interfaces bidirectionnelles définies [8]. Aussi, NesC définit un modèle basé sur les tâches et les captures d'évènements matériels, et détecte des éclatements d'information pendant la compilation. Un Composant implémente des interfaces utilisées par d'autres composants pour communiquer avec ce composant [5].

Un composant, du point de vue programmation, est constitué de plusieurs sections et l'ensemble de toutes ces sections donne lieu à la création de ce composant.

b. Implémentations

Cette section définit les connexions entre les différents composants qu'utilise l'application. Dans une implémentation sont donc principalement définis les composants qui fournissent les interfaces à notre application (ce sera généralement des composants primitifs). Habituellement, nous devons utiliser les interfaces que nous fournissent d'autres composants, primitifs ou non primitifs, et en définitive pour chacune de ces interfaces, que nous utiliserons dans la création de notre composant. Nous devons obligatoirement définir des relations avec les composants qui fournissent ces interfaces. Le processus définissant ces relations s'appelle (wiring) [8].

Exemple:

```
Implémentation {
components  Main, MonAppliM ;
Main.StdControl - >MonAppliM.StdControl ; }
```

c. Configuration

C'est à cet endroit que l'on déclare les autres composants dont se servira l'application. Cette possibilité offerte par le langage permet de faire de la programmation modulaire et de réutiliser des composants préalablement définis.

La structure de la partie Configurations est la même que celle de la partie Implémentation.

Exemple:

```
Configuration MonAppli {
implementation {
components GenericComm ,TimerC ;
TimerC . StdControl ->GenericComm . StdControl ;
}}
```

d. Module

Cette partie du code est généralement plus étendue et c'est dans celle-ci que l'on programme réellement le comportement qu'on souhaite voir réalisé par l'application. Elle est divisée en trois sous-sections: Uses, Providers et Implémentation.

Exemple:

```
Module MonAppliM {
provides {...}
uses {...}}
implementation{...}
```

La première sous-section, (provides), indique au compilateur les interfaces fournies par le composant. Par exemple, si le composant est une application, on doit fournir aunions l'interface StdControl.

```
provides {
interface interfacequenousfournissons;}
```

La sous-section (uses) informe le compilateur que nous allons faire usage d'une interface (on pourra donc effectuer des appels aux méthodes de cette interface). Pour faire cela, nous avons besoin de respecter quelques règles. Ainsi, si nous utilisons une interface, il nous faut avoir dans la section (implémentation) un lien (”wiring”) reliant cette interface avec un composant qui la fournit. Finalement, utiliser une interface oblige implicitement à gérer les évènements pouvant se produire du fait d'avoir utilisé cette interface précise.

```
uses {
interface interfacequenousutilisons ;
}
```

La sous-section (implementation), est celle qui contiendra toutes les méthodes nécessaires pour fournir le comportement souhaité à ce composant ou à une application. Cette sous-section doit contenir au moins:

- Les variables globales qui sont utilisées par l'application
- Les fonctions que doivent mettre en œuvre pour les interfaces que nous fournissons.
- Les évènements que doivent mettre en œuvre venant des interfaces que nous utilisons.

II.3.4 Types de données utilisés par NesC

Les types de données qui peuvent être utilisés en NesC sont tous ceux que fournit le langage C standard plus quelques autres qui n'apportent pas de puissance de calcul mais qui sont très utiles pour la construction de paquets puisqu'ils fournissent à l'utilisateur le nombre de bits qu'ils occupent. Ceci est important au moment de la transmission des informations par l'intermédiaire des ondes radio.

Quelques exemples de ces types additionnels :

- **uint16_t** : entier non signé sur 16 bits.
- **uint8_t** : entier non signé sur 8 bits.

- **Result_t** : utilisé pour savoir si une fonction a été exécutée avec succès ou non, c'est comme un booléen mais avec les valeurs SUCCESS et FAIL (retour de fonction).
- **bool** : valeur booléenne qui peut être TRUE ou FALSE.

En NesC, il est possible de faire une utilisation dynamique de la mémoire mais ce n'est pas très recommandé à moins que cela ne soit absolument nécessaire. Pour pouvoir l'utiliser il existe un composant particulier appelé MemAlloc qui permet une gestion dynamique de la mémoire.

II.3.5 Types de fonctions en NesC

En NesC les fonctions peuvent être de types très variés. Il y a d'abord les fonctions classiques avec la même sémantique qu'en C et la façon de les invoquer est aussi la même qu'en C. Il y a aussi des types supplémentaires de fonctions: task, event, command. Les fonctions (command) sont principalement des fonctions qui sont exécutées de manière synchrone, c'est-à-dire que lorsqu'elles sont appelées elles sont exécutées immédiatement. La manière d'appeler ce genre de fonction est:

```
Call interface.nomFonction
```

Les fonctions (task) sont des fonctions qui sont exécutées dans l'application, utilisant la même philosophie que les threads. C'est principalement une fonction normale qui est invoquée de la manière suivante:

```
post interface .nomTache
```

Immédiatement après son invocation, l'exécution du programme qui l'a invoqué se poursuit. Les fonctions (event) sont des fonctions qui sont appelées quand on relèvera un signal dans le système. Ces fonctions possèdent principalement la même philosophie que la programmation orientée événements, de sorte que, lorsque le composant reçoit un événement, on effectue l'invocation de cette fonction. Il existe une méthode pour pouvoir invoquer manuellement ce type de fonctions:

```
signal interface .nomEvenement
```

On utilise donc le mot clef **call** pour déclarer les évènements et le mot clef **post** pour déclarer les tâches.

II.4 Conclusion

Dans ce chapitre, nous avons présenté les outils logiciels nécessaires pour développer une application dans les RCSF. Nous avons détaillé les fonctionnalités et les caractéristiques du système d'exploitation TinyOS et du langage NesC. Ces outils ont des spécificités bien particulières leurs permettant d'être utiles aux systèmes embarqués et aux équipements à ressources limitées tels que les capteurs.

Dans le chapitre suivant, nous montrons comment nous exploitons ces outils pour développer une application orientée événement.

Chapitre III.
Implémentation

III.1 Introduction

Dans ce chapitre, nous détaillons le fonctionnement de l'application que nous avons développée. Nous commençons dans la première partie par préciser l'environnement et les outils utilisés dans le développement. Ensuite, nous exprimons une vue globale sur le fonctionnement de l'application. Finalement, nous présentons les différentes parties du code en détails.

III.2 Vue d'ensemble des fonctions

Ce projet consiste à réaliser une application orientée évènement permettant la détection de la température qui dépasse un certain seuil dans un réseau de capteurs sans fil déployée dans une zone d'intérêt. Dans notre cas, la zone d'intérêt est une maison composée de plusieurs pièces et la température seuil est de l'ordre de 30°.

Dans notre application, l'alerte est représentée de deux manières différentes. Dans la première, il s'agit d'alerter le centre de contrôle distant en passant par le réseau Internet et dans la seconde, l'utilisateur est avisé par SMS en passant par le réseau téléphonique GSM.

Le processus d'exécution de notre application suit la démarche suivante:

- Détecter le paramètre de l'environnement (température dans notre cas) grâce à un nœud capteur, nommé Sender.
- Envoyer la donnée captée par le nœud Sender au nœud Receiver via l'interface radio qui est chargée lui-même de l'envoyer au nœud central appelé station de base.
- Dans le cas où la température dépasse la valeur seuil (30°), une alerte est déclenchée. Cette alerte est représentée par une interface graphique qui indique le capteur qui a détecté cet événement pertinent. Dans ce cas, une alerte est diffusée aux autres

centres de contrôle par une diffusion sur Internet. En outre, quand l'alerte est un message SMS, plusieurs copies de ce message sont envoyés aux personnels pour les informer cet événement pertinent.

III.3 Environnement de travail et outils de développement

Notre but étant l'implémentation d'une application de surveillance de l'environnement à base d'un réseau de capteur sans fil. De point de vu matériel, nous avons utilisé la plateforme Telosb et pour l'autre logiciel le système d'exploitation TinyOs, ainsi que le langage de programmation NesC pour les parties de l'application qui sont embarquées sur la station de base et les nœuds capteurs Senders et Receivers. Pour la partie installée sur PC, nous avons utilisé le langage de programmation JAVA. La figure 9 présente une architecture générique de l'application développée.

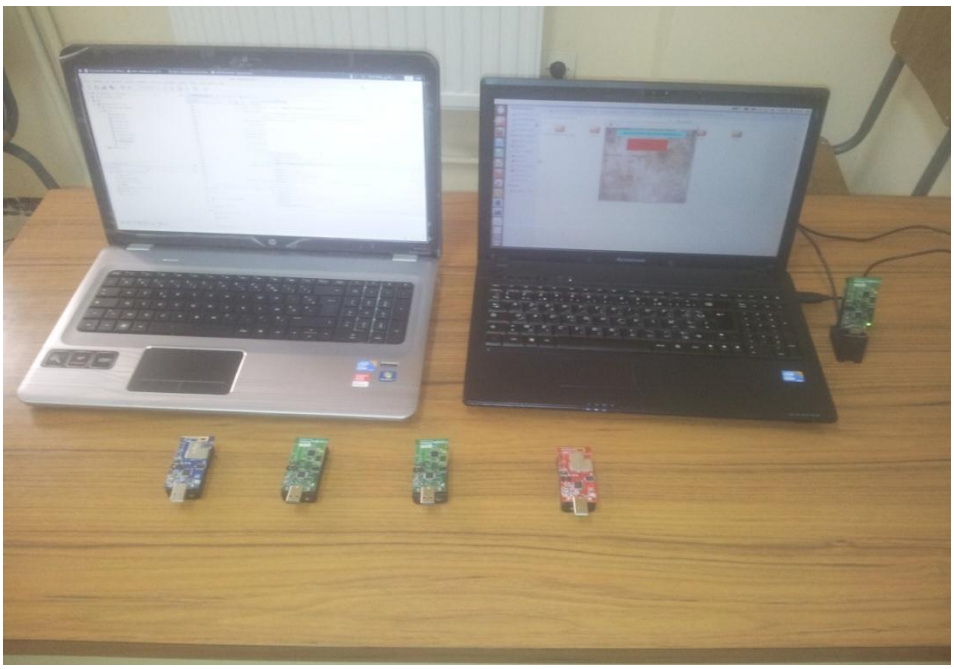

Figure 9: Architecture du Système

III.4 Les différentes parties de l'application

Dans cette section, nous détaillons l'implémentation des principales fonctionnalités offertes par l'application que nous avons développée. Notre application est composée de quatre parties: une partie embarquée sur des nœuds capteurs (Senders) du réseau,

une deuxième partie embarquée sur les nœuds relais (Receivers), une autre partie embarquée sur la station de base et une dernière partie installée sur un PC. Cette dernière partie assure la synchronisation des données avec l'application nœud capteur.

III.4.1 Partie Sender

C'est une partie de l'application qui est embarquée sur des nœuds capteurs et qui contient les différentes fonctionnalités permettant la capture de paramètre de l'environnement et leur envoie au nœud Receiver. Le code de cette partie est divisé en deux fichiers distincts: le premier contient la définition des composants qui seront utilisés par l'application déployée sur le capteur alors que le second fichier comprend les modules, les interfaces utilisées ainsi que le code qui sera exécuté.

Dans la partie Sender, plusieurs fichiers ont été définis:

- **SenderAppC.nc:**Dans ce fichier, les composants suivants sont déclarés:
  - MainC**:** Interface du système avec la séquence de boot TinyOS.
  - LedsC : Permet l'utilisation des leds.
  - SensirionSht11C () as Sense: Capteur de température auquel on associe le nom Sensor.
  - SenderC: Composant défini dans le fichier SenderC.nc ;
  - TimerMillicC () as Timer : Temporisateur en millisecondes nommé Timer ;
  - ActiveMessageC.

Il faut ensuite associer ces composants au composant SenderC afin de pouvoir les utiliser dans le code de celui-ci. Cela est effectué de la manière suivante:

```
App.Boot ->MainC.Boot;

App.Leds ->LedsC.Leds;

App.Packet ->SerialActiveMessageC;

App.AMSend ->SerialActiveMessageC.AMSend[AM_TEMPERATURE_MSG];

App.Receive ->ActiveMessageC.Receive[AM_TEMPERATURE_MSG];

App.RadioControl -> ActiveMessageC;

App.SerialControl ->SerialActiveMessageC;
```

De ce fait, avec cette déclaration et ces définitions, nous aurons ainsi accès à toutes les fonctionnalités proposés par ces composants.

- **SenderC.nc:**Ce fichier contient la définition et l'implémentation du module SenderAppC. Dans ce fichier, nous spécifions toutes les interfaces qui seront utilisées.

```
Uses interface Boot;
 Uses interface Timer<TMilli>;
 Uses interface Leds;
 Uses interface Read<uint16_t>;
 Uses interface Packet;
 Uses interface AMSend;
 Uses interface SplitControl as RadioControl;
```

Donc, le programme que nous développons, sera un ensemble de fonctions réagissant chacune à certains évènements.

Le premier événement à déclencher correspond au démarrage du capteur et l'action associée se nomme Boot.booted. Il est représenté par les instructions suivantes:

```
event void Boot.booted(){
busy = FALSE ; call RadioControl.start();
}
```

Voir aussi la fonction RadioControl qui démarre le timer qui permet de contrôler la lecture de température et l'envoie de données une fois l'antenne est allumée.

```
event void RadioControl.startDone(error_t err){
if (err == SUCCESS) {
callTimer.startPeriodic(TIMER_PERIOD);
}
else{
```

```
call RadioControl.start();
}
}
```

Pour récupérer le paramètre de l'environnement (Température), nous effectuons une lecture ce qui déclenchera l'événement Read. La fonction qui mesure la température est développée de cette manière:

```
event void Read.readDone (error_t result, uint16_t val){
Temperature_Msg* payload;
double valeur ; uint32_t b;
if (result == SUCCESS)  {
/* Si l'antenne n'est pas occupée, on récupère le vrai contenu du paquet. */
If (busy == FALSE){
payload = (Temperature_Msg*) (call Packet.getPayload(&pkt,
sizeof(Temperature_Msg)));
if (payload == NULL) {
Return;
}
valeur = -39.6 + 0.01 * val;
```

```
b=(uint32_t) valeur;
if (b > 30) {payload->temperature=b;
// payload->temperature=b;
If ((call AMSend.send (AM_BROADCAST_ADDR, &pkt,
sizeof(Temperature_Msg)) == SUCCESS)) {
busy = TRUE;}
} } } }
```

III.4.2 Partie Receiver

C'est une deuxième partie de l'application embarquée sur des nœuds capteurs permettant la collection des informations captées depuis les Senders.

Comme la partie nœud capteur Sender, la partie nœud capteur Receiver est constituée aussi de deux fichiers distincts: ReceiverAppC.nc qui contient la définition de tous les composants qui seront utilisés par l'application Receiver et ReceiverC.nc qui contient l'implémentation du module ReceiverAppc.

Quand le capteur Receiver reçoit un paquet via la liaison radio, une méthode nommée Receive.receive extrait le contenu du paquet et la valeur de la température stockée dans ce paquet.

```
event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
Temperature_Msg* rcvPayload; /* Contenu du paquet reçu */
Temperature_Msg* sndPayload; /* Contenu du paquet à envoyer */
call Leds.led1Toggle(); /* On allume led 1. */
if(len != sizeof(Temperature_Msg)){
return NULL;
   }
rcvPayload = (Temperature_Msg*)payload;
sndPayload = (Temperature_Msg*)call Packet.getPayload(&pkt,
sizeof(Temperature_Msg));
if(sndPayload == NULL){
return NULL;
   }
sndPayload->temperature = rcvPayload->temperature;
if(call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(Temperature_Msg)) ==
SUCCESS){
```

```
busy = TRUE;
    }
returnmsg;
  }
```

III.4.3 Partie station de base

C'est une autre partie de l'application embarquée sur un nœud capteur permettant la collection des informations envoyées par les Receivers et ensuite les envoyer via la liaison série.

Cette partie de l'application permettant de récolter les messages envoyés par les nœuds capteurs de type Receiver du réseau et les acheminer vers le PC à travers la liaison série. Nous utilisons aussi deux fichiers comme dans les parties nœuds capteurs Sender et Receiver, dans lesquelles nous avons ajouté deux fonctions afin d'avoir la possibilité d'envoyer et de recevoir les paquets via la liaison série. Dans ce qui suit il y'a deux instructions permettant l'envoi et la réception à travers la liaison série.

```
/*Envoi du message qui se trouve dans le buffer « uartQueue » provenant des noeuds
capteurs du réseau sur la liaison série */

if (call UartSend.send[id](addr, uartQueue[uartOut], len) == SUCCESS)
/*Si l'envoi ce fait sans problème la led jaune du noeud capteur sera allumé*/

   call Leds.led1Toggle(); }

/*Réception du message provenant de la liaison série */

event message_t *UartReceive.receive[am_id_t id](message_t *msg,
void *payload,  uint8_t len) {

/*Enregistrer le message reçu dans un buffer «radioQueue » qui sera transmet par la
suite vers les noeuds capteurs via les ondes radio*/

radioQueue [radioIn] = msg;  }
```

III.4.4 Partie PC

C'est la partie de l'application qui est installée sur un PC à partir de laquelle nous surveillons l'environnement et nous alertons les autres centres de contrôles et les per-

sonnels en cas de danger. De ce fait, quand la température dépasse un certain seuil (30° dans notre cas), une alerte est envoyée localement et une autre au centre de contrôle distant. Ceci est illustré dans l'application développée par une interface simple dont le but est de lancer une alerte locale et une autre externe sur les réseaux Internet et/ou téléphonique.

a. Alerte via le réseau Internet

Cette alerte est envoyée via le réseau Internet quand la température dépasse le seuil (30°). Elle est faite par une communication entre processus en utilisant des sockets. La température et l'emplacement du capteur qui a détecté cet événement sont affichés sur l'interface de l'application résidée dans le PC du poste de contrôle. Ce dernier est relié directement avec la station de base qui joue le rôle d'une Socket de type SocketServer qui permet d'envoyer les données vers les autres centres de contrôle du réseau qui sont des sockets clients qui demandent et reçoivent les données envoyées de la part du serveur. Les figures 10 et 11 illustrent respectivement le PC local et un centre de contrôle avec les données relevées de l'environnement.

Figure 10: Interface du client

Figure 11: Interface du serveur

Nous montrons dans ce qui suit comment la communication est faite entre le client et le serveur:

- **Machine Serveur**

Cette machine est reliée directement avec la station de base. Elle contient une application qui affiche les données nécessaires pour savoir la valeur de la température et l'emplacement du capteur dans l'environnement. Cette application joue le rôle d'un serveur de type ServerSocket auquel il faut spécifier un numéro de port pour ce serveur, pour que les autres clients puissent le contacter.

Le code qui permet de réaliser ce mécanisme est le suivant:

```
ServerSocket  server = new ServerSocket(7777);
```

Le serveur attend et accepte les connexions avec les clients avec le code suivant:

```
Socket sock = server.accept();
```

Lorsque la connexion est établie avec les autres clients qui sont des centres de contrôle, le serveur envoie les données vers ces clients en utilisant des flux d'écriture.

- **Machine client**

Cette machine joue le rôle d'un centre de contrôle distant et elle récupère les données envoyées par le serveur en utilisant le type de socket Socket dont il faut spécifier l'adresse du serveur et son numéro de port comme suit:

```
Socket client = new Socket(″adresse du serveur″,numero de port)
```

Nous avons ajouté le principe des threads en cas où il y a plusieurs clients (centres de contrôle distants) qui sont connectés au serveur pour récupérer les données. Ainsi, quand le client récupère les données depuis le serveur, il les affiche sur une interface comme il est illustré dans la figure 10.

b. Alerte par SMS via un modem 3G

Une autre partie de ce code se concentre aussi sur l'envoi des SMS quand la température dépasse un seuil déterminé afin d'alerter certaines personnes quelques soient leur emplacement géographique. Pour se faire, il s'agit d'utiliser une API[5] d'envoi des SMS développée sous licence publique GNU[6] [3].D'où, lorsque la température dépasse le seuil défini on instancie la classe ComputeSmsData. Cette classe contient la méthode Send (String message) permettant d'envoyer des SMS avec un argument de type chaîne de caractères qui représentent une alerte contenant la valeur de température et son emplacement dans l'environnement.

La méthode Send est représentée comme suite:

```
public void send (String a){
//declare new objects and variables
ComputeSmsData sms = new ComputeSmsData();
SerialToGsmstg = new SerialToGsm("/dev/ttyUSB0");
    String retStr = new String("");
    String sss = new String();
```

[5] Application Programming Interface

[6]GNU's Not UNIX.

```
String alarmNumber = new String("+213552400192");  // a real phone number here
// running code
// check for messages
retStr = stg.checkSms();
if (retStr.indexOf("ERROR") == -1) {
System.out.println("Phone # of sender: " + stg.readSmsSender());
System.out.println("Recv'd SMS message: " + stg.readSms());
}
// send a message
sss = stg.sendSms(alarmNumber, "Attention la temperature maintenant est " +a+ " c'est trop dangeureuse : fait appel au pompier ");
}
```

III.5 Conclusion

Dans ce chapitre nous avons montré la démarche de notre application qui permet de surveiller des endroits cibles dans l'environnement de déploiement des capteurs. Nous avons montré aussi comment alerter un centre de contrôle en passant par le réseau Internet et le réseau téléphonique quand un événement pertinent survient.

Conclusion Générale

Les réseaux de capteurs sans fil sont une nouvelle technologie qui a surgi après les grands progrès technologiques concernant le développement des capteurs intelligents, des processeurs puissants et des protocoles de communication sans fil. Ce type de réseau composé de centaines ou de milliers d'éléments, a pour but la collecte de données de l'environnement, leur traitement et leur dissémination vers le monde extérieur.

Notre travail consiste à mettre en œuvre une application orientée évènement basée sur les RCSF. Cette application est constituée de quatre parties: une partie qui est embarquée sur des nœuds capteurs (Senders) du réseau permettant la mesure des paramètres de l'environnement, une deuxième qui embarquée sur le reste des nœuds (Receivers) chargée de recevoir les messages envoyés par les Senders et les acheminer vers la troisième partie qui est embarquée sur la station de base .Cette dernière permet la collection des messages envoyés et leur acheminement vers le PC, et une quatrième partie qui est installée sur le PC qui est illustre les alertes locales et distantes quand une grandeur dépasse un certain seuil. Cette alerte est présentée en deux façons: la première est faite pour alerter les autres centres de contrôle à travers le réseau, alors que la deuxième est faite pour signaler un danger par l'envoie des messages SMS au personnels chargés de la surveillance de ce RCSF.

En perspectives, nous proposons de généraliser cette application sur d'autres domaines tels que la santé pour garantir une surveillance distante des personnes dépendantes (âgées et handicapées), la surveillance des forêts, etc…

Bibliographies

[1] Crossbow TELOSB Data sheet. http://www.willow.co.uk/TelosB_Datasheet.pdf

[2] AKYILDIZ (I. F.), SU (W.), SANKARASUBRAMANIAM (Y.) et CAYIRCI. (E.), « Wireless Sensor networks: a survey. », *IEEE Communications Magazine*, vol. 40, no 8, p. 102–114, August2002

[3] Abdallah MAKHOUL, Réseaux de capteurs : localisation, couverture et fusion de donnée, thèse de doctorat, Université de Franche-Comté (LIFC) ,2008.

[4] Mohamed LEHSAINI, Diffusion et couverture basées sur le clustering dans les réseaux de capteurs : application à la domotique, thèse de doctorat, Université A.B Tlemcen && Université de Franche-Comté, 2009.

[5] Yacine CHALLAL, Réseaux de capteurs sans fils, article scientifique, 2008.

[6] TinyOS Mission Statement. UC Berkeley, 2004, http://www.tinyos.net/special/mission.

[7] Khaled MEDJHOUM, Les Système Embarqué TinyOS, article scientifique, Université de Brest.

[8] David GAY, Philip LEVIS, David CULLER, Eric BEWER ; "nesC 1.1 Language Reference Manual", 2003.

[9] B. W. Kernighan and D. M. Ritchie. The C Programming Language, Second Edition. Prentice Hall, 1988.

ANNEXE I. Installation de TinyOs 2.1.2

Cette annexe est un guide d'installation de **TinyOS 2.1.2** sur **Ubuntu12.04**.Cette installation est testée avec succès en suivant ces démarches:

1. Suppression de la version précédente de **TinyOS** si elle existe:

Pour installer la dernière version de **TinyOS** il faut supprimer la version précédente et même sa version du compilateur **gcc-msp430**, en fait ça par les deux commandes suivantes:

```
sudo apt-get remove tinyos-2.1.1

sudo apt-get autoremove --purge msp430*
```

2. Ajouter le lien du package de TinyOS dans le fichier qui contient la liste des package d'Ubuntu. Premièrement ouvrir le fichier des packages avec:

```
sudogedit /etc/apt/sources.list
```

Ensuite, ajouter ces lignes à la fin du fichier:

```
#TinyOS Repository

deb http://tinyos.stanford.edu/tinyos/dists/ubuntu kramic main
```

Sauvegarder le fichier (**CTRL+S**) après fermer le.

3. Maintenant en va installer le TinyOS 2.1.2 avec les commandes suivants:

```
sudo apt-get update

sudo apt-get install tinyos-2.1.2
```

4. Après la terminaison de l'installation, il faut pointer sur le dossier **TinyOS 2.1.2** avec:

```
cd /opt/tinyos-2.1.2
```

Et créer le fichier tinyos.sh avec le contenu suivant :

```
#! /usr/bin/env bash
# www.ElectronicsPub.com
# TinyOS 2.1.2 Configuration Guide
# Here we setup the environment
# variables needed by the tinyos
# make system
echo "Setting up for TinyOS 2.1.2"
export TOSROOT=
export TOSDIR=
export MAKERULES=
TOSROOT="/opt/tinyos-2.1.2"
TOSDIR="$TOSROOT/tos"
CLASSPATH=$CLASSPATH:$TOSROOT/support/sdk/java
MAKERULES="$TOSROOT/support/make/Makerules"
export TOSROOT
export TOSDIR
export CLASSPATH
export MAKERULES
```

Si le fichier déjà existe, il faut garantir que son contenu soit le même avec le précédent.

5. Après, il faut définir les variables d'environnement sur le fichier .bashrc. Ouvrir ce fichier par la commande suivante:

```
sudogedit ~/.bashrc
```

Maintenant, ajouter les lignes suivantes :

```
# Start TinyOS environment pathing
```

```
export TOSROOT=/opt/tinyos-2.1.2
export TOSDIR=$TOSROOT/tos
export CLASSPATH=$TOSROOT/support/sdk/java/tinyos.jar:.$CLASSPATH
export MAKERULES=$TOSROOT/support/make/Makerules
export PATH=/opt/msp430/bin:$PATH
source /opt/tinyos-2.1.2/tinyos.sh
# End TinyOS pathing
```

En suit, sauvegarder les modifications par:

```
source  ~/.bashrc
```

6. Finalement, il reste de tester si l'installation est bien faite. Pour faire ce teste, brancher le capteur sur votre pc et tapez la commende qui visualise le capteur branché :

```
motelist
```

Le résultat est comme suite :

```
Reference  Device           Description
---------- ---------------- ----------------------------------------
---
XBTIQYGZ   /dev/ttyUSB1     XBOW Crossbow Telos Rev.B
```

Ensuite, accéder au répertoire /etc/tinyos2.1.2/apps/Blink et taper sur la ligne de commande:

```
make telosb install bsl,/dev/ttyUSB1
```

Si tout ça marche bien, le résultat est affiché comme suit:

```
……
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
   writing TOS image
cp build/telosb/main.ihex build/telosb/main.ihex.out
   found mote on /dev/ttyUSB1 (using bsl,auto)
```

```
    installing telosb binary using bsl
tos-bsl --telosb -c /dev/ttyUSB1 -r -e -I -p build/telosb/main.ihex.out
MSP430 Bootstrap Loader Version: 1.39-goodfet-8
Mass Erase...
Transmit default password ...
Invoking BSL...
Transmit default password ...
Current bootstrap loader version: 1.61 (Device ID: f16c)
Changing baudrate to 38400 ...
Program ...
2482 bytes programmed.
Reset device ...
rm -f build/telosb/main.exe.out build/telosb/main.ihex.out
```

Quelques liens qui vous seront utiles:

- Divers tutoriels sur le site de documentation de TinyOS :http://docs.tinyos.net/index.php/TinyOS_Tutorials**.**
- TinyOS installation guide on Ubuntu : http://www.eetutorials.com/article/28/1/TinyOS-installation-guide-on-Ubuntu.html

ANNEXE II. Exemple des fichiers écrient en langage NesC

Dans cette annexe, nous présentons des fichiers écrient en Nesc qui sont fournis par défaut avec **TinyOS-2.1.2** après son installation.

- **BlinkC.nc**

```
/**
 * Implementation for Blink application.  Toggle the red LED when a
 * Timer fires.
 **/
#include "Timer.h"
moduleBlinkC @safe()
{
uses interface Timer<TMilli> as Timer0;
uses interface Timer<TMilli> as Timer1;
uses interface Timer<TMilli> as Timer2;
uses interface Leds;
uses interface Boot;
}
implementation
{
event void Boot.booted()
  {
call Timer0.startPeriodic( 250 );
call Timer1.startPeriodic( 500 );
call Timer2.startPeriodic( 1000 );
  }
event void Timer0.fired()
  {
dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
call Leds.led0Toggle();
  }
```

```
event void Timer1.fired()
  {
dbg("BlinkC", "Timer 1 fired @ %s \n", sim_time_string());
call Leds.led1Toggle();
  }
event void Timer2.fired()
  {
dbg("BlinkC", "Timer 2 fired @ %s.\n", sim_time_string());
call Leds.led2Toggle();
  }
}
```

- **BlinkAppC.nc**

```
// $Id: BlinkAppC.nc,v 1.6 2010-06-29 22:07:14 scipioExp $
/**
 * Blink is a basic application that toggles a mote's LED periodically.
 * It does so by starting a Timer that fires every second. It uses the
 * OSKI TimerMilli service to achieve this goal.
 *
 * @author tinyos-help@millennium.berkeley.edu
 **/
configurationBlinkAppC
{
}
implementation
{
components MainC, BlinkC, LedsC;
components new TimerMilliC() as Timer0;
components new TimerMilliC() as Timer1;
components new TimerMilliC() as Timer2;
BlinkC ->MainC.Boot;
  BlinkC.Timer0 -> Timer0;
```

```
  BlinkC.Timer1 -> Timer1;
  BlinkC.Timer2 -> Timer2;
BlinkC.Leds -> LedsC;
}
```

- **StdControl**

```
/* This interface is used to switch between
 * the on and off power states of the component providing it.  A call to the
 * <code>start()</code> command is a request to switch a component into the
 * on state, and a call to the <code>stop()</code> is a request to switch a
 * component into the off state.
*/
interface StdControl
{
  /**
   * Start this component and all of its subcomponents.
   *
   * @return SUCCESS if the component was either already on or was
   *         successfully turned on<br>
   *         FAIL otherwise
   */
commanderror_t start();
  /**
   * Stop the component and any pertinent subcomponents (not all
   * subcomponents may be turned off due to wakeup timers, etc.).
   *
   * @return SUCCESS if the component was either already off or was
   *         successfully turned off<br>
   *         FAIL otherwise
commanderror_t stop();}
```

ANNEXE III. Installation de l'API SMSApi

Cette annexe montre comment réussir un programme java d'envoyer des **SMS** depuis un téléphone branché avec une liaison **USB** ou bien on utilisant un modem **3G**. Tout ça est basé sur l'api SMSApi. Ce travail est réussi avec **Ubuntu12.04LTS 32bits** et on suivant les étapes suivant :

a) Il faut télécharger l'api SMSApi depuis ce lien:

   http://sourceforge.net/projects/java-sms-api/.

b) Une autre étape, télécharger le fichier javax.comm.properties sur cette page :http://www.oracle.com/technetwork/java/index-jsp-141752.html, il suffit de le mettre sous le répertoire 'votrejdk'/jre/lib.

c) Pour garantir la communication entre le pc et le téléphone portable via le port serial, le package RXTXcomm.jar est charger de garantir ce genre de communication,

   Le fichier est disponible dans l'URL suivant:

   http://rxtx.qbang.org/wiki/index.php/Download.

   Après avoir télécharger le fichier, en doit le mettre sous le répertoire

   'votreJdk'/jre/lib/ext/.

d) La dernière étape de téléchargement est de télécharger le fichier librxtxserial.so qui

   Représente le driver, en suivant ce lien pour le télécharger:

   http://jlog.org/java_lin.htm.

   Placer ce fichier dans le chemin suivant 'votrejdk'/jre/lib/i386.

e) Finalement, si tout est marché bien, en va envoyer des sms à travers cette dernière étape.

   Décompresser l'api SMSApi télécharger et suivre les étapes mentionnées ci-dessus. Il faut créer un nouveau projet sous un IDE (nous avons testé ce travail avec l'IDE Netbeans6.9.1) et lui ajouter les deux classes ComputerSmsData.java et Serial-ToGsm.java. Ensuite, ajouter les lignes suivantes dans votre propre code :

```
//declare new objects and variables
        ComputeSmsData sms = new ComputeSmsData();
SerialToGsmstg = new SerialToGsm("serial0");
        String retStr = new String("");
        String sss = new String();
        String alarmNumber = new String("+393351234567");   // a real phone number here
        ....
        // running code

        // check for messages
retStr = stg.checkSms();
if (retStr.indexOf("ERROR") == -1) {
System.out.println("Phone # of sender: " + stg.readSmsSender());
System.out.println("Recv'd SMS message: " + stg.readSms());
        }

        // send a message
sss = stg.sendSms(alarmNumber,"Hello GSM World");
```

Liste des figures

ملخص:

في العقود الأخيرة، كانت الحاجة إلى مراقبة الظواهر الفيزيائية مثل الحرارة والضغط والسطوع أمر بالغ الأهمية بالنسبة للعديد من التطبيقات العلمية والصناعية. و تحقيقا لهذه الغاية، فإن شبكات الاستشعار اللاسلكية هي تطور تكنولوجي هدفه رصد منطقة جغرافية و الإنذار عند وقوع حدث خطير.

في هذه المذكرة، قمنا بإنشاء تطبيق هدفه رصد ومراقبة درجة الحرارة في منطقة معينة بالاعتماد على شبكة من أجهزة الاستشعار اللاسلكية. في حالة ما إذا زادت درجة الحرارة عن حدها الطبيعي، يتم إطلاق إنذارين، الأول لإبلاغ مراكز التحكم مرورا بشبكة الأنترنت، و الثاني إرسال رسائل قصيرة إلى الجوالات.

الكلمات الرئيسية: شبكة الإستشعار اللاسلكي, التطبيق الموجه بحدث, نظام التشغيل TinyOS, لغة البرمجة NesC.

Résumé

Depuis quelques décennies, le besoin d'observer et de contrôler des phénomènes physiques tels que la température, la pression ou encore la luminosité est essentiel pour de nombreuses applications scientifiques et industrielles. Pour cela, les réseaux de capteurs sans fil sont des progrès technologiques réalisés pour surveiller une zone géographique et de remonter une alarme en cas de détection d'un événement redouté.

Dans ce mémoire, nous avons développé une application de surveillance de température dans l'environnement à base d'un réseau de capteurs sans fil. Quand la température dépasse un certain seuil, deux alertes sont déclenchées, une pour informer les centres de contrôle à travers le réseau Internet et l'autre via des messages SMS.

Mots clés: Réseau de capteurs sans fil, Application orientée évènement, TinyOS, NesC.

Abstract

Since a few decades, the need to observe and to check physical phenomena such as the temperature, the pressure and the luminosity is essential for many scientific and industrial applications. For this purpose, the wireless sensor networks are technological progress made to control a geographical zone and to go back up an alarm in case of detection of a dreaded event.

In this report, we developed an application of temperature surveillance in the environment with wireless sensor networks. When the temperature exceeds a threshold value, two alarms are activated; one to inform distant control centers through Internet networks and the other one via SMS messages.

Key words: Wireless sensor network, Application Event-Driven, TinyOS, NesC.