Université Abou Bekr Belkaid جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
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

Développement d'une application orientée événement pour les réseaux de capteurs

Réalisé par :

- **GHORZI SAADIA.**

Présenté le 04 Juillet 2012 devant le jury composé de MM.

- *Mme. DIDI FADOUA.* *(Président)*
- *Mr. LEHSAINI MOHAMMED.* *(Encadreur)*
- *Mr. BENMAMMAR BADR.* *(Examinateur)*
- *Mme. LABRAOUI NABILA.* *(Examinateur)*

Année universitaire: 2011-2012

Remerciements

Je loue tout d'abord le bon dieu tout puissant qui a éclairé mon chemin.

Je tiens à remercier Mon encadreur Mr. LEHSAINI Mohamed pour sa disponibilité, ses idées, ses conseils et ses encouragements qu'il m'a prodigués tout au long de cet humble travail.

Je présente mes gratitudes aux membres du jury ***Mme. LABRAOUI.N***, ***Mme. DIDI*** et ***Mr. BENMAMMER*** Qui ont bien voulu examiner et évaluer notre travail et qui nous font l'honneur de participer à la soutenance.

Un grand MERCI à tous les enseignants du Département Informatique qui m'ont formée durant ces deux dernières années de Master.

J'aimerais également remercier tous mes amis et camarades de notre promotion RSD pour leur sympathie et aide qui m'a donnée la force pour continuer. Sans oublié de remercier notre équipe RCSF « Amina, Warda , Zineb et abbes » pour leurs soutien , aide et surtout leurs encouragements durant le temps où on a travaillé ensemble.

Mes remerciements vont aussi à tous ceux qui ont contribué de près ou de loin à la concrétisation de ce travail. Qu'ils trouvent tous ici l'expression de ma gratitude et ma parfaite considération.

Dédicaces

*A la mémoire de ma grande mère **Souci Rabea**,*

*A la mémoire de mon frère **Zouheir**,*

*A ma chère **MAMAN** et mon adorable **PAPA**,*

*A mes deux chères sœurs **Zoulikha** et **Amoura**,*

*A ma chère amie **GOUAOURA Ibtissem**,*

A toutes mes tantes et mes oncles,

A tout mes cousins et cousines,

A tous mes fidèles amis,

A toutes les personnes qui m'ont aidée de près ou de loin,

Je dédié ce travail

**********GHORZI Saadia**********

TABLE DES MATIERES

Table des matières

Listes des figures 5

Liste des tableaux 7

Liste des abréviations 8

Introduction Générale 9

Chapitre I: Les Réseaux de Capteurs Sans Fil 11

I.1 Introduction 11

I.2 Qu'est-ce qu'un Réseau de Capteurs Sans Fil? 11

I.3 Qu'est-ce qu'un nœud capteur ? 12

I.3.1 Unité de détection (Sensing Unit) 13

I.3.2 Unité de traitement (Processing Unit) 13

I.3.3 Unité de transmission (Transceiver Unit) 13

I.3.4 Unité d'alimentation (Power Unit) 13

I.4 Exemples de capteurs 14

I.5 Architecture d'un réseau de capteur 15

I.6 Classification des réseaux de capteurs 16

I.6.1 Par taille 16

I.6.2 Par type d'application 16

I.7 Caractéristiques des réseaux de capteurs 17

I.7.1 Densité « importante » des nœuds 17

I.7.2 Topologie dynamique 17

I.7.3 Auto-organisation 18

I.7.4 La tolérance de fautes 18

I.7.5 Passage à l'échelle 18

I.8 Domaines d'applications 18

I.8.1 Médical 19

I.8.2 Bâtiment 19

I.8.3 Armée 19
I.8.4 Applications environnementales 20
I.8.5 La domestique 20
I.8.6 Applications commerciales 21
I.9 Contraintes de conception des RCSF 21
I.9.1 La tolérance de pannes 21
I.9.2 L'échelle 21
I.9.3 Les coûts de production 22
I.9.4 La topologie de réseau 22
I.9.5 Les contraintes matérielles 22
I.9.6 Les médias de transmission 22
I.9.7 La consommation d'énergie 23
I.10 Topologies des réseaux de capteurs sans fils 24
I.10.1 Topologie Hiérarchique 24
I.10.2 Topologie plate 24
I.10.3 Topologie basée sur la localisation 25
I.11 Les protocoles de communication 25
I.11.2 Caractéristiques de ZigBee 26
I.11.3 Comparaison des technologies de communication 26
I.12 Conclusion 27
Chapitre II: Outils matériels et logiciels 28
II.1 Introduction 28
II.2 Le système d'exploitation "TinyOs" 28
II.2.1 Présentation de TinyOs 28
II.2.2 Propriétés de TinyOs 29
II.2.3 Primitives de TinyOS 30
II.2.4 Cibles de TinyOS 34
II.2.5 Tâches, évènements et applications 34
II.3. Concepts de la programmation orientée composants 35

II.4. Le langage de programmation NesC 37
II.4.1 Concepts de NesC 37
II.4.2 Les composants de NesC 38
II.4.3 Structuration d'un programme en NESC 40
II.5. Autres outils logiciels 42
II.5.1 Outils de simulation 42
II.5.2 SerialForwarder 43
II.6 Abstraction matérielle 43
II.6.1 Mote, processeur, RAM et Flash 43
II.6.2 Radio et antenne 44
II.7 Les composants du capteur TelosB 44
II.7.1 Le Microcontrôleur MSP430 45
II.7.2 Timer 45
II.7.3 ADC12 45
II.7.4 DMA 45
II.8 Conclusion 46
Chapitre III: Développement d'une application « Event/Driven » pour les RCSF 47
III.1 Introduction 47
III.2 Domaines d'application 48
III.3 Langages utilisés 48
III.4 Architecture de la plateforme 49
III.4.1 Installation logicielle 49
III.4.2 Installation matérielle 50
III.5 Etats de la plateforme 50
II.5.1 Etat stable 51
III.6 Etat d'alerte 51
III.7 Contexte d'exécution 52
III.8 Conclusion 54

Tables des Matières

Conclusion Générale 55
Annexe 56
Installation de TinyOS sur Ubuntu 11.10 56
Liste des Références 58

LISTE DES FIGURES

Listes des Figures

Figure I.1 : Schéma général d'un réseau de capteurs 12

Figure I.2: architecture d'un nœud capteur 12

Figure I.3 : Exemples de capteurs : MicaZ et TelosB 14

Figure I.4 : Architecture d'un réseau de capteur 15

Figure I.5 : Domaine d'application « médical » 19

Figure I.6 : Domaine d'application « Armé » 20

Figure I.7 : consommation d'énergie en captage, traitement et transmission 24

Figure I.8 : Positionnement de ZigBee 26

Figure II.1 : Symbole de TinyOs 29

Figure II.2 : Schéma d'allocation de la mémoire 33

Figure II.3 : Aperçu générale de TinyOS 36

Figure II.4: Schéma des interactions internes au système TinyOS 36

Figure II.5: Syntaxe d'un module 38

Figure II.6: Syntaxe d'une configuration 38

Figure II.8 : SerialForwarder 43

Figure II.9 : Les composants matériels du TelosB 44

Figure III.1: Architecture de la plateforme 49

Figure III.2 : état stable 51

Figure III.3 : état d'alerte 52

Figure III.4 : état d'alerte I 53

FigureIII.5 :état d'alerte II 53

LISTE DES TABLEAUX

Liste des tableaux

Tableau I.1 : les caractéristiques de quelques exemple de capteurs 14
Tableau I.2: Comparaison des technologies de communications 27

LISTE DES ABREVIATIONS

Liste des abréviations

RCSF: Réseau de Capteurs Sans Fil.

CAN: Convertisseur Analogique Numérique.

RF: Radio-Fréquence.

TinyOS: Tiny Microthreading Operating System.

NesC: Network Embedded Systems C.

BAL: Boîte Aux Lettres.

HAL: Hardware Adaptation Layer.

ISR: Intrrupt Service Routine.

FIFO: First In First Out.

MIG: Message Interface Generator.

ACLK: Auxillary clock.

MCLK: Master clock.

SMCLK: Sub main clock.

ADC 12: Analog to Digital Converter 12 bits.

Introduction Générale

Introduction Générale

Grâce aux progrès technologiques, le développement de la microélectronique et la communication sans fil, il y avait naissance de composants d'une taille extrêmement réduite et embarquant des unités de calcul et de communication sans fil, appelés "nœuds capteurs ou motes". Ces dispositifs ont la capacité de détecter, calculer, communiquer et coopérer entre eux pour former un réseau de capteurs.

Un réseau de capteurs est composé d'un grand nombre de nœuds qui sont déployés aléatoirement dans une zone d'intérêt pour la couvrir ou surveiller quelques cibles dans cette zone. Les nœuds capteurs présentent une certaine autonomie d'énergie car ils sont généralement dotés de batteries qui sont difficiles à les recharger ou les remplacer puisqu'ils se trouvent généralement dans des zones hostiles. En outre, ces capteurs sont déployés en grand nombre car ils sont sujets à des pannes telles que l'écrasement par un animal ou l'épuisement de batteries. En plus des contraintes énergétiques, les capteurs une capacité de calcul et de mémoire très limitée.

L'étude et le développement des réseaux de capteurs est devenu un sujet vaste et enrichissant car les RCSF ont envahit plusieurs types d'applications. Dans ce type de réseaux, à un capteur est confié la tâche de détecter, de calculer et de remonter l'occurrence d'un événement (par exemple, un changement de température, des vibrations, ...) au poste de contrôle. Ce capteur doit être donc capable de collecter des données relatives à son environnement, de les traiter, puis, si nécessaire, de les communiquer à des capteurs voisins via un médium sans fil. Le déploiement et la coopération de ce type d'appareils permettent de former alors un réseau qui peut être utilisé dans différents domaines : militaire (par exemple, le suivi de déplacement des troupes ennemies), civil (la détection de feux de forêt), médical (le suivi des patients), animalier (l'étude des migrations d'espèces), etc...

Si les perspectives d'utilisation des RCSF sont attrayantes, les problématiques qu'engendrent ces réseaux n'en sont pas moins nombreuses. A priori, ils ne dépendent d'aucune infrastructure et les capteurs n'ont aucune information

relative au réseau auquel ils appartiennent. De plus, étant construits de façon ad hoc, ces réseaux s'organisent automatiquement.

Notre objectif est la réalisation d'une application « orientée événement » portée sur les réseaux de capteurs sans fil. Dans cette application on s'intéresse en particulier aux applications qui nécessitent de remonter rapidement l'information à la station de base telles que les feux de forêts et la surveillance des patients. De ce fait, cette application permet de garantir d'aviser un centre de contrôle à temps pour qu'il intervient dans le meilleur délai. En plus, cette application permet de garantir une longue durée de vie aux réseaux de capteurs car il y aura transmission que s'il y a l'occurrence d'un événement et évite les transmissions périodiques qui sont coûteuses en termes d'énergie. En outre, la réalisation de cette application nécessite des outils logiciels bien spécifiques car les capteurs sont des dispositifs à ressources limitées.

Ce manuscrit s'articule autour de trois chapitres :

Le chapitre 1 : Présente une description générale des réseaux de capteurs sans fil ainsi que leurs caractéristiques, contraintes et spécificités.

Le chapitre 2 : Présente les outils matériels et logiciels pour la mise en place d'un réseau de capteurs.

Le chapitre 3 : est une implémentation d'une application de type "Event-Driven".

En fin de mémoire, une conclusion est donnée pour résumer les apports essentiels de notre travail ainsi que les ouvertures et les perspectives pour le futur.

CHAPITRE I

LES RESEAUX DE CAPTEURS SANS FIL

Chapitre I: Les Réseaux de Capteurs Sans Fil

I.1 Introduction

L'objectif de ce chapitre est de faire une description synthétique des réseaux de capteurs, leurs architectures, leurs caractéristiques et contraintes ainsi que leurs domaines d'applications variés.

I.2 Qu'est-ce qu'un Réseau de Capteurs Sans Fil?

Les réseaux de capteurs sans fil (RCSF) sont souvent composés d'un nombre important de nœuds, appelés capteurs. Ces capteurs sont de petits dispositifs électroniques qui peuvent être de simples senseurs, jusqu'à des petits calculateurs capables de capter, modifier, stocker et transmettre une information à une station de base où un autre capteur par radio afin de concentrer l'information sur un point de collecte appelé « Station de base ou Sink » [1,2,3].

La figure I.1 synthétise un réseau de capteurs dans lequel il y a un grand nombre de capteurs qui sont déployés dans une zone d'intérêt pour surveiller un tel phénomène tel que les feux de forêts. A l'occurrence d'un événement qui sera détecté par un capteur ou plusieurs capteurs l'information devrait remonter rapidement à la station base qui est reliée à un centre de contrôle. Ceci est dans le but de l'aviser rapidement afin qu'il puisse intervenir dans le bref délai pour minimiser les dégâts. Dans certains cas, la station de base est liée au poste de contrôle via une connexion Internet ou satellitaire.

En outre, les capteurs formant ce réseau doivent être équipés d'une batterie qui est considérée comme une ressource très précieuse puisqu'elle est généralement non rechargeable et difficile à la remplacer, ce qui présente la contrainte la plus critique pour la survie d'un tel réseau.

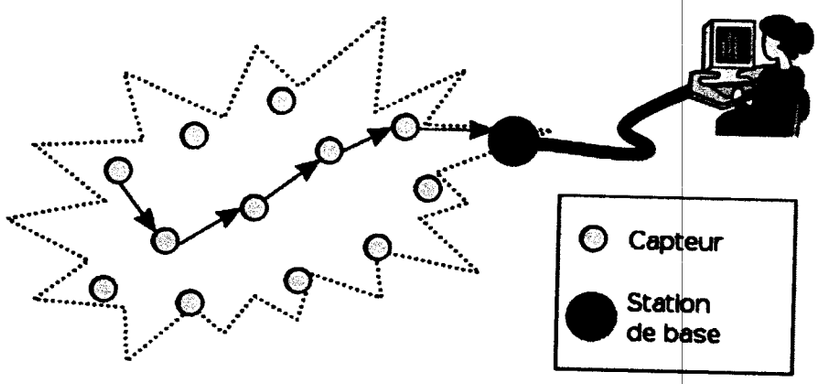


Figure I.1 : Schéma général d'un réseau de capteurs

Avant de présenter en détails les RCSF, on propose présenter les éléments formant ce type de réseaux.

I.3 Qu'est-ce qu'un nœud capteur ?

Un nœud capteur est un élément qui a la possibilité de collecter périodiquement les données sur le phénomène surveillé et envoyer les rapports de captage à un nœud spécial appelé *puits* (sink). Ce nœud capteur est composé des éléments suivants comme présente la figure I.2 :

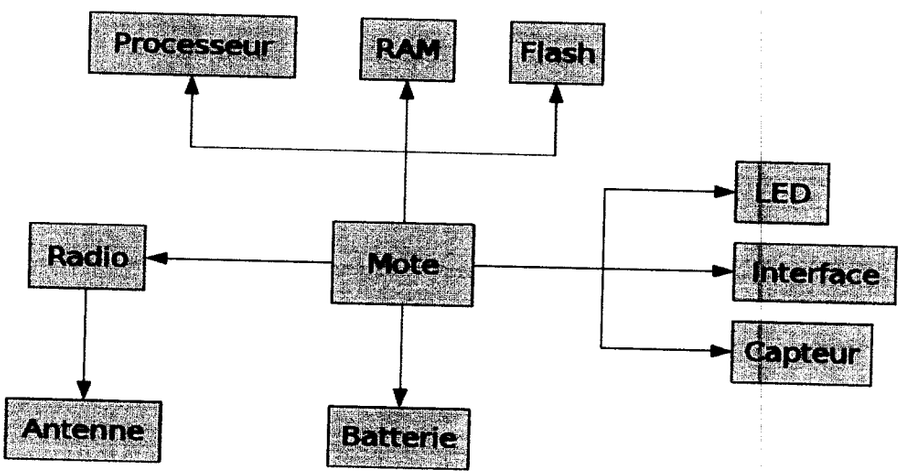


Figure I.2: architecture d'un nœud capteur [16]

I.3.1 Unité de détection (Sensing Unit)

L'unité de détection permet de détecter le phénomène observé et le convertir d'un signal analogique en un autre numérique en utilisant un convertisseur analogique numérique (ADC[1]). Ce signal sera ensuite transmis à l'unité de traitement.

I.3.2 Unité de traitement (Processing Unit)

Cette comprend un processeur avec une petite unité de stockage, une RAM pour les données et une mémoire Flash pour stocker les programmes. Elle est chargée d'exécuter les protocoles de communication qui permettent aux nœuds capteurs de collaborer avec les autres nœuds pour accomplir la requête en question.

I.3.3 Unité de transmission (Transceiver Unit)

Cette unité est composée de deux éléments: Radio et antenne. Elle est chargée d'effectuer toutes les émissions et les réceptions de données. La radio fréquence est chargée de réaliser les communications entre les capteurs. Par ailleurs, le système de transmission dans les réseaux de capteurs possède une portée de quelques dizaines de mètres. D'où pour assurer des communications entre deux nœuds distants tout en préservant l'énergie, le réseau utilise un routage multi-sauts.

I.3.4 Unité d'alimentation (Power Unit)

Les capteurs sont dotés d'une unité d'alimentation composée soit d'une batterie ou des piles. Cette unité permet d'approvisionner l'ensemble des unités du capteur en énergie. Cependant, les éléments composants cette unité sont rarement rechargeable ou remplaçable car les réseaux de capteurs sont généralement dans des zones à accès difficiles tels que les montagnes, le pôle nord, etc.

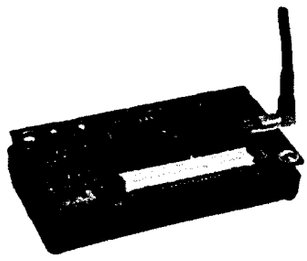
Il y a des capteurs qui sont équipés d'instruments de localisation (GPS) et dans certains cas d'un mobilisateur qui permet de déplacer un capteur d'un endroit à un autre.

[1] Analog-to-Digital Converter

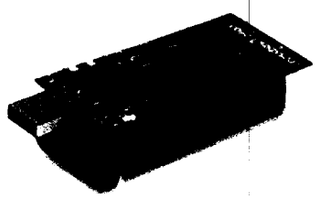
I.4 Exemples de capteurs

Il existe plusieurs modèles commercialisés des nœuds capteurs dans le marché. Parmi eux, on trouve la famille des Mica et la famille des Telos qui sont développés par l'université de Berkeley et fabriqués par Crossbow.

Pour illustrer les caractéristiques des capteurs, on présente un type de chaque famille: MicaZ de la famille Mica et TelosB de la famille Telos comme montre dans la figure I.3. Le tableau I.1 résume les caractéristiques de ces capteurs.



Capteur MicaZ



Capteur TelosB

Figure I.3 : Exemples de capteurs : MicaZ et TelosB

Tableau I.1 : les caractéristiques de quelques exemple de capteurs

Caractéristique	TelosB	MicaZ
Flash	48kb	512kb
RAM	10kb	4kb
Antenne	Radio Chipcon Wireless Transceiver	Antenne Chipcon CC2420
MCU	1MHz TI MSP430	8MHz ATmega128
Dimension	65 x 31 x 6 Mm	58 x 32 x 7 Mm
2 piles AA	radio+cpu: 75mW sleep mode: 140µW	radio+cpu mode: 63mW sleep mode: 30 µW
external flash	512kB	512kB

Ces deux capteurs sont utilisés pour la surveillance. Dans notre contexte, on a choisi des capteurs de type TelosB qu'ils sont plus pratique que les Micaz et qu'ils son programmable par liaison USB. De plus, ils supportent avec TinyOS-2.x contrairement à Micaz où la température n'est pas facilement gérable pour cette nouvelle version de TinyOs.

I.5 Architecture d'un réseau de capteur

Un réseau de capteurs est constitué d'un grand nombre de nœuds capteurs qui sont déployés dans une zone d'intérêt. Chacun de ces nœuds a la capacité de collecter des données et de les transférer à un point de collecte appelé « station de base » ou « puits » directement ou par l'intermédiaire d'un certain nombre de relais selon un mode de communication multi-sauts. Le nœud puits transmet ensuite ces données par Internet ou par satellite à l'ordinateur central «Centre de traitement des données» pour analyser ces données et prendre des décisions en conséquence [2,3]. La figure I.4 représente l'architecture d'un réseau de capteur.

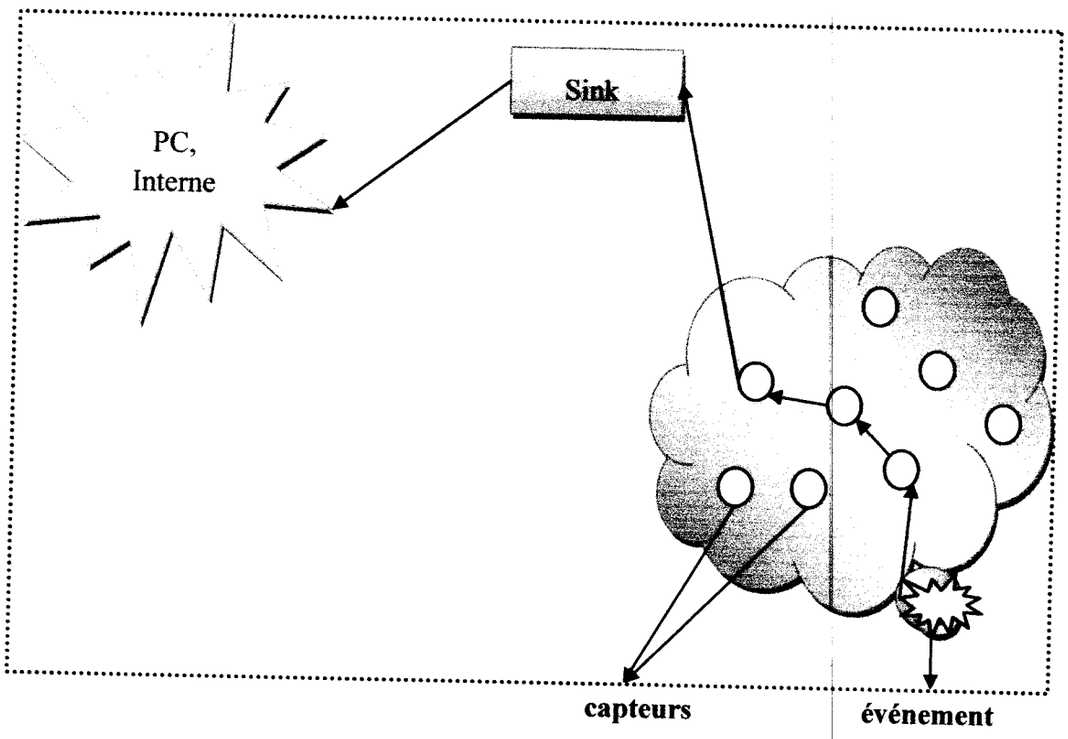


Figure I.4 : Architecture d'un réseau de capteur

I.6 Classification des réseaux de capteurs

Les réseaux de capteurs peuvent être classés selon plusieurs critères:

I.6.1 Par taille

a) Plateforme de capteurs miniaturisés

Cette plateforme est dédiée aux capteurs de taille très réduite (quelques mm3) et de faible bande passante (<50Kbps) tel que le capteur Spec.

b) Plateforme de capteurs généraux

La plus récente est basée sur MicaZ. Cette plateforme est développée pour capter et router des informations du monde ambiant.

c) Plateforme de capteurs à haute bande passante

Par exemple les capteurs de type Imote dont la communication se base sur la norme Bluetooth 1.1. Cette plateforme est utilisée pour envoyer des données ayant une taille importante.

d) Plateforme de passerelles

Cette plateforme est dédiée à transporter les informations envoyées par le réseau de capteurs vers un réseau traditionnel (Ethernet, 802.11). Les réseaux de capteurs formés de Stargate est un exemple de ce type.

I.6.2 Par type d'application

La transmission de données dans les réseaux de capteurs de ce type peut se faire suivant plusieurs modèles. On distingue trois principaux modèles:

a) Le modèle Driven-event

Dans ce type de modèle, il y a envoi de l'information à la station de base que s'il y a occurrence d'un événement pertinent. Dans ce modèle, on s'intéresse essentiellement à la détection d'un événement et à le faire remonter rapidement à un centre de contrôle. Il est commode aux applications qui ont l'aspect temps réel. Cependant, l'inconvénient majeur de ce modèle est la redondance des données, car les nœuds qui détectent par le même événement envoient la même information au « sink » [6].

b) Le modèle Query-Driven

Ce modèle est semblable au modèle event-driven sauf que la collecte des informations sur l'état de l'environnement est initiée par des interrogations envoyées par le « sink ». On peut utiliser ce modèle pour contrôler et reconfigurer les nœuds. Par exemple, le « sink » peut envoyer des commandes au lieu des interrogations pour modifier le programme d'un nœud capteur, son taux de trafic et son rôle. Seul le nœud capteur jouant le rôle de «sink» est autorisé à émettre des demandes d'interrogations ou des commandes.

c) Le modèle continu (périodique)

Dans ce modèle, les nœuds capteurs envoient les informations d'une manière continue au nœud « sink » suivant un volume de trafic prédéterminé [7]. On trouve ce modèle dans les applications orientées surveillance qui doivent envoyer périodiquement de l'information à la station de base. Il consomme beaucoup d'énergie puisqu'il y a un grand nombre de transmissions.

I.7 Caractéristiques des réseaux de capteurs

Les principales caractéristiques des réseaux de capteurs se résument dans ce qui suit :

I.7.1 Densité « importante » des nœuds

Les réseaux de capteurs se composent généralement d'un nombre très important de nœuds pour garantir une couverture totale de la zone surveillée. Ceci engendre un niveau de surveillance élevé et assure une transmission plus fiable des données sur l'état du champ de capteur.

I.7.2 Topologie dynamique

La topologie des réseaux de capteurs est caractérisée par son instabilité. Ceci est le résultat des facteurs suivants:

a) La mobilité des nœuds

Les nœuds capteurs peuvent être attachés à des objets mobiles qui se déplacent librement et arbitrairement, introduisant ainsi une topologie instable du réseau.

b) La défaillance des nœuds

Les capteurs sont sujets de plusieurs pannes par exemple, ils peuvent être écrasés par des animaux ou ils peuvent épuiser leurs batteries puisqu'ils sont déployés généralement dans des zones hostiles. Les capteurs qui sont atteints par ce type de pannes, sont considérés comme des capteurs supprimés.

c) L'ajout de nouveaux nœuds

De nouveaux nœuds capteurs peuvent facilement être rajoutés. Il suffit de placer un nouveau capteur qui soit dans la portée de communication d'au moins un autre nœud capteur du réseau déjà existant.

I.7.3 Auto-organisation

L'auto-organisation s'avère très nécessaire pour ce type de réseau afin de garantir son déploiement dans des zones hostiles ou sa maintenance. Vu les différentes raisons résultant d'une topologie instable du réseau de capteurs, ce dernier devra être capable de s'auto-organiser pour continuer ses applications.

I.7.4 La tolérance de fautes

Le réseau doit être capable de maintenir ses fonctionnalités sans interruptions en cas de défaillance d'un ou plusieurs de ses capteurs. Cette défaillance peut être causée par une perte d'énergie, ou par dommage physique ou interférence de l'environnement. Le degré de tolérance dépend du degré de criticité de l'application et des données échangées [4].

I.7.5 Passage à l'échelle

Les réseaux de capteurs peuvent contenir des centaines voire des milliers de nœuds capteurs [4]. Un nombre aussi important engendre beaucoup de transmissions inter-nodales et nécessite que le nœud « *Sink* » soit équipé d'une mémoire importante pour stocker les informations reçues.

I.8 Domaines d'applications

Les capteurs sont utilisés dans de nombreux domaines [8,9] :

I.8.1 Médical

Les micro-caméras sont déjà présentes dans certains hôpitaux, cela permet d'obtenir des images de l'intérieur du patient.

Des études visant à implanter des capteurs dans le corps d'une personne pour surveiller ses fonctions vitales et alerter les secours en cas de problème sont en cours.

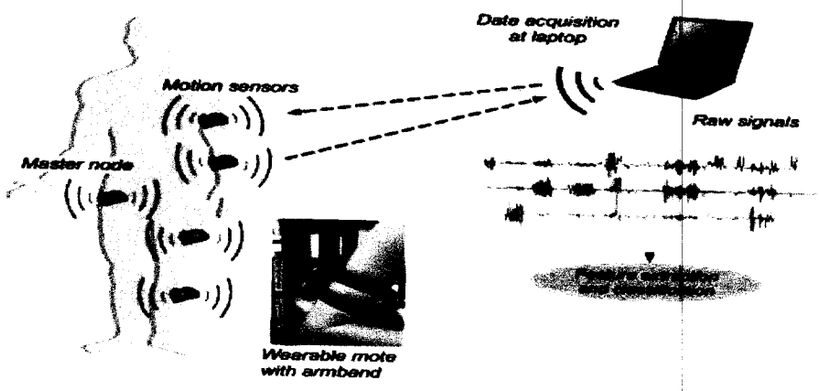


Figure I.5 : Domaine d'application « médical »

Ce type d'applications nécessite de remonter l'information rapidement au centre de contrôle dans le cas de l'occurrence d'un événement pertinent chez le patient. Par exemple, une glycémie ou une tension qui atteint une valeur seuil.

I.8.2 Bâtiment

De nombreuses applications existent dans ce domaine, on utilise des capteurs pour détecter une fuite d'eau ou de gaz ou la présence de fissures dangereuses dans les mûrs. En cas de l'occurrence de ce type d'événements, l'information doit être remontée rapidement au centre de contrôle pour alerter les occupants de ces bâtiments.

I.8.3 Armée

Le déploiement d'un réseau de capteurs permet une analyse du terrain même inaccessible. On peut ainsi détecter une activité ennemie, la présence de radiations ou de produits chimiques dangereux sans y être exposé.

Figure I.6 : Domaine d'application « Armé »

Quand un événement de ce type survient, une alerte est envoyée au centre de contrôle pour l'aviser et pour prendre les mesures nécessaires.

I.8.4 Applications environnementales

La présence de capteurs thermiques dans une forêt peut signaler un début d'incendie ce qui permet une intervention rapide. Les capteurs de produits chimiques permettent une détection d'une éventuelle fuite toxique pour l'environnement. Les animaux peuvent aussi être surveillés grâce à des capteurs (implantés sur leur corps ou disséminés dans la nature) ce qui permet de les étudier et les protéger sans avoir à les déranger.

I.8.5 La domestique

Avec le développement technologique, les capteurs peuvent être embarqués dans des appareils, tels que les aspirateurs, les fours à micro-ondes, les réfrigérateurs,.... Ces capteurs embarqués peuvent interagir entre eux et avec un réseau externe via une passerelle résidentielle pour permettre à un utilisateur de contrôler les appareils domestiques localement ou à distance. En outre, le déploiement des capteurs de mouvement et de température dans les futures maisons dites intelligentes permet d'automatiser plusieurs opérations domestiques telles que: la lumière s'éteint et la musique se met en état d'arrêt quand la chambre est vide, la climatisation et le chauffage s'ajustent selon les points multiples de mesure, le déclenchement d'une alarme par le capteur anti-intrusion quand un intrus veut accéder à la maison.

I.8.6 Applications commerciales

Des nœuds capteurs pourraient améliorer le processus de stockage et de livraison. Le réseau ainsi formé, pourra être utilisé pour connaître la position, l'état et la direction d'un paquet ou d'une cargaison. Un client attendant un paquet peut alors avoir un avis de livraison en temps réel et connaître la position du paquet. Des entreprises manufacturières, via des réseaux de capteurs pourraient suivre le procédé de production à partir des matières premières jusqu'au produit final livré. Grâce aux réseaux de capteurs, les entreprises pourraient offrir une meilleure qualité de service tout en réduisant leurs coûts. Les produits en fin de vie pourraient être mieux démontés et recyclés ou réutilisés si les micro-capteurs en garantissent le bon état.

I.9 Contraintes de conception des RCSF

La conception et la réalisation des RCSF est influencée par plusieurs paramètres, parmi lesquels nous citons la tolérance aux pannes, la scalabilité, le coût de production, l'environnement d'exploitation, la topologie du réseau, les contraintes matérielles, le support de transmission et la consommation d'énergie. Ces facteurs importants servent comme directives pour le développent des algorithmes et protocoles utilisés dans les réseaux de capteurs, ils sont considérés également comme métriques de comparaison de performances entre les différents travaux dans le domaine.

Les principaux facteurs et contraintes influençant l'architecture des réseaux de capteurs peuvent être résumés comme suit [10]:

I.9.1 La tolérance de pannes

Le réseau doit être capable de maintenir ses fonctionnalités sans interruption en cas de défaillance d'un de ses capteurs. Ce premier défi consiste à identifier et modéliser formellement les modes de défaillances des capteurs, puis de repenser aux techniques de tolérance aux pannes à mettre en œuvre sur le terrain.

I.9.2 L'échelle

Le nombre de capteurs déployés pour un projet peut atteindre milliers. Un nombre aussi important de capteurs engendre beaucoup de transmissions inter-nodales et nécessite que le puits "sink" soit équipé de beaucoup de mémoire pour stocker les

informations reçues. Par ailleurs, la plupart des techniques et des algorithmes sont conçus au début à des réseaux de capteurs de petite taille et ils perdent leurs performances lorsqu'on passe à l'échelle.

I.9.3 Les coûts de production

Le coût de production d'un seul capteur est encore un peu important. De ce fait, la réduction du coût de production pourrait élargir l'utilisation de cette nouvelle technologie dans plusieurs domaines.

I.9.4 La topologie de réseau

Le déploiement d'un grand nombre de capteurs nécessite une maintenance de la topologie. Cette maintenance consiste en trois phases : déploiement, post-déploiement (les capteurs peuvent se déplacer, ne plus fonctionner,...), et redéploiement.

I.9.5 Les contraintes matérielles

Un nœud doit être placé dans une petite surface n'excédant pas, généralement, un centimètre cube ($1cm^3$). En outre de cette contrainte de surface, un ensemble de conditions doit être satisfait :

Un nœud capteur doit :

- consommer le minimum d'énergie,
- opérer dans une haute densité,
- être autonome et pouvoir opérer sans assistance,
- être adaptatif à l'environnement

I.9.6 Les médias de transmission

Dans un réseau de capteurs, les capteurs sont reliés par une architecture sans fil. Pour permettre des opérations sur ces réseaux dans le monde entier, le média de transmission doit être normé. On utilise le plus souvent la technologie Zigbee et Bluetooth.

I.9.7 La consommation d'énergie

Comme les nœuds capteurs sont des composants de petite taille, ils ne peuvent être équipés que par des sources limitées d'énergie (<5 AH, 1.2 V). De plus, dans certaines applications, ces nœuds ne peuvent pas être dotés de mécanismes de rechargement d'énergie, par conséquent, la durée de vie d'un nœud capteur dépend fortement de la durée de vie de la batterie associée. Sachant que les réseaux de capteurs sont basés sur la communication multi-sauts, chaque nœud joue à la fois un rôle de source de données et de routeur également. Le mal fonctionnement d'un certain nombre de nœuds entraîne un changement significatif sur la topologie globale du réseau, et peut nécessiter un routage de paquets différent et une réorganisation totale du réseau. C'est pour cela que le facteur de consommation d'énergie est d'une importance primordiale dans les réseaux de capteurs.

Les étapes de consommation d'énergie par ce nœud peuvent être, dès lors, divisées en trois phases: le captage, la communication et le traitement de données.

a) Phase de détection:

L'énergie consommée au moment de la détection varie suivant la nature de l'application. Une détection sporadique consomme moins d'énergie qu'un contrôle d'événement constant.

b) Phase de communication:

L'énergie de communication représente la plus grande proportion de l'énergie totale consommée au niveau d'un nœud. Cette communication est assurée dans la plupart des RCSF par le support de transmission radio. La consommation d'énergie de ce dernier est affectée par plusieurs facteurs : le type du système de modulation, la quantité des données à communiquer, la puissance de transmission, etc.

La minimisation d'énergie pendant la communication est principalement liée aux protocoles développés pour la couche MAC et la couche réseau. Le but des protocoles de cette dernière est de trouver les routes optimales en termes de consommation d'énergie. En effet, la perte d'énergie due à un mauvais acheminement des paquets de données à un impact sur la durée de vie du réseau. La figure I.10 résume la variation de consommation de l'énergie par les différentes opérations d'un capteur.

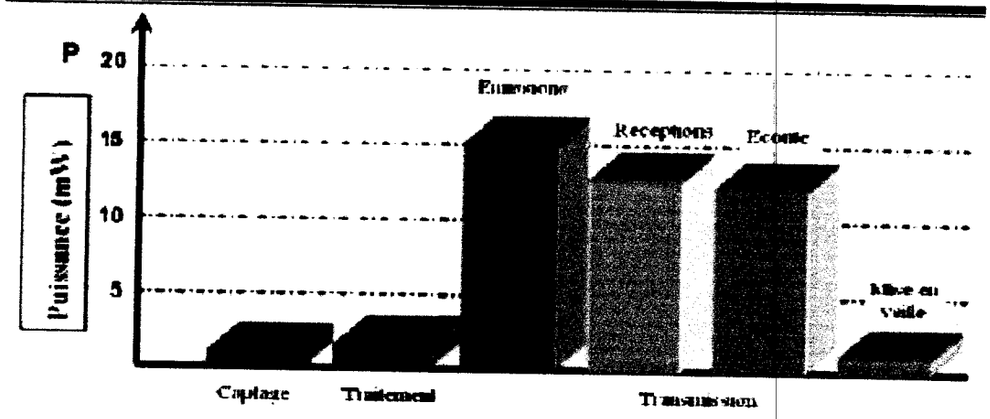


Figure I.7 : consommation d'énergie en captage, traitement et transmission

I.10 Topologies des réseaux de capteurs sans fils

Les topologies des RCSF sont déterminées à partir des protocoles de routage utilisés pour l'acheminement des données entre les nœuds et le sink. Ces protocoles peuvent être hiérarchiques, plat ou basée sur la localisation.

I.10.1 Topologie Hiérarchique

Les protocoles à topologie hiérarchique forment des réseaux dans lesquels un nœud central *sink* (le niveau supérieur de la hiérarchie) est relié à un ou plusieurs autres nœuds qui appartiennent à un niveau plus bas dans la hiérarchie (deuxième niveau) avec une liaison point à point. Aussi, chacun des nœuds du deuxième niveau aura également un ou plusieurs autres nœuds de niveau plus bas dans la hiérarchie (troisième niveau) reliées à lui avec une liaison point à point. Chaque ensemble de nœuds forme un groupe appelé aussi cluster. Le nœud central n'a aucun autre nœud au-dessus de lui dans la hiérarchie sauf le centre de traitement des données ou la passerelle si elle existe. Les nœuds du deuxième niveau jouent le rôle des passerelles entre ceux du troisième niveau et le sink. Dans ce cas, le routage devient plus simple, puisqu'il s'agit de passer par les passerelles pour atteindre le nœud destination [6,11].

I.10.2 Topologie plate

Les protocoles à topologie plate considèrent que tous les nœuds sont égaux, ont les mêmes fonctions, et peuvent communiquer entre eux sans devoir passer par un nœud particulier ou une passerelle. Seul un nœud particulier, le sink, est chargé de la collecte

des données issues des différents nœuds capteurs afin de les transmettre vers les centres de traitement [6].

I.10.3 Topologie basée sur la localisation

Les protocoles à topologie basée sur la localisation suppose que :

- Le réseau est partitionné en plusieurs zones de localisation.
- Chaque zone a son identifiant.
- Chaque nœud a un identifiant EUI (*End-system Unique Identifier*) et enregistre dynamiquement l'identifiant de la zone à laquelle il appartient temporairement.

L'information temporaire de localisation appelée LDA (*Location Dependent Address*) qui est un triplet de coordonnées géographiques (longitude, latitude, altitude) obtenues, par exemple, au moyen d'un GPS avec une précision dépendant du type de l'application. Une telle topologie exige l'implémentation d'un algorithme de gestion de localisation qui permet aux nœuds de déterminer les endroits approximatifs des autres nœuds.

Ce type de topologie est mieux adapté aux réseaux avec une forte mobilité Avant d'envoyer ces données à un nœud destination, le nœud source utilise un mécanisme pour déterminer la localisation de la destination puis inclut l'identifiant de zone de localisation et du nœud destination dans l'en-tête du paquet à envoyer [6].

I.11 Les protocoles de communication

Dans les réseaux de capteurs, on utilise plusieurs technologies de communications telles ZigBee et Bluetooth. Cependant, ZigBee est le plus utilisé dans les RCSF.

Dans cette section, on présente ZigBee et on le compare à d'autres technologies usuellement utilisées.

I.11.1 ZigBee/IEEE 802.15.4

ZigBee / IEEE 802.15.4 [12] est un protocole qui permet de mettre en place une communication entre des petites radios, avec une consommation énergétique réduite comparativement à d'autres technologies de communications telles que Bluetooth. Ce

paramètre est important car ces radios sont souvent implantées sur des objets mobiles de équipés de capteurs. Or, les capteurs possèdent une durée de vie limitée, d'où l'optimisation des coûts de communication est primordiale. La figure I.11 suivante illustre le positionnement de ZigBee par rapport à d'autres standards.

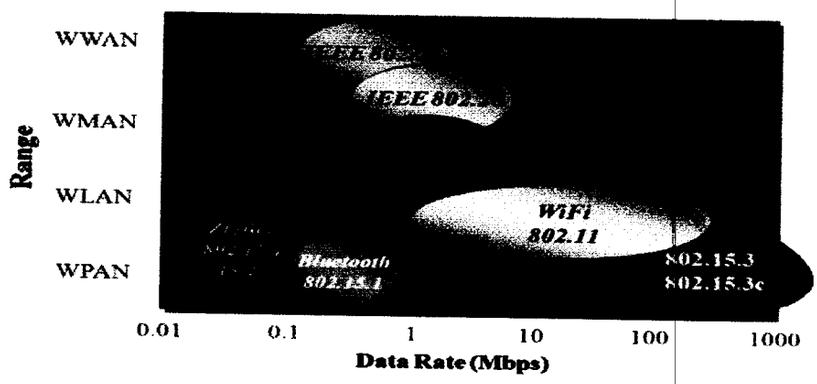


Figure I.8 : Positionnement de ZigBee

I.11.2 Caractéristiques de ZigBee

- Usage sans restrictions géographiques
- Pénétration à travers les murs et plafonds
- Installation automatique/semi-automatique
- Possibilité de rajouter/retirer des dispositifs
- Coût avantageux
- **Débit :** 10kbps-115.2kbps
- **Portée radio:** 10-75m
- Jusqu'à 65k nœuds par réseau
- Jusqu'à 100 réseaux co-localisés
- Jusqu'à 2 ans de durée de vie de batterie standards Alkaline

I.11.3 Comparaison des technologies de communication

Dans cette section, on présente une comparaison entre les technologies de communication les plus utilisées. La table illustre cette comparaison en termes de débit, consommation d'énergie et portée.

Tableau I.2: Comparaison des technologies de communications

Technologie	Débit Max	Consommation	Temps de Démarrage	Portée
Zigbee (IEEE 802.15.4)	250 KB/s	Faible	Court	100-160 m
Bluetooth (IEEE802.15.1)	1 MB/s	Moyenne	Moyen	10-100 m
WI-FI (IEEE 802.11)	11- 54 Mb/s	Elevée	Long	300 m

I.12 Conclusion

Dans ce chapitre, on a présenté tout d'abord les réseaux de capteurs sans fil, leurs architectures, leurs contraintes et leurs classifications. Ensuite, on a présenté quelques domaines d'applications en se focalisant sur les applications de type « Event-Driven ». Enfin, on a exposé le protocole ZigBee qui est une technologie dédié pour les systèmes à ressources limitées à l'instar des réseaux de capteurs.

Dans le prochain chapitre on va présenter les outils logiciels qui sont nécessaires pour développer une application de type « Event-Driven ». Ces outils sont conçus spécialement pour les dispositifs à ressources limitées.

Chapitre II :

Outils matériels et logiciels

Chapitre II: Outils matériels et logiciels

II.1 Introduction

Les réseaux de capteurs sont considérés parmi les systèmes à ressources limitées. De ce fait, les outils logiciels conçus aux ordinateurs ne sont plus adaptables à ce type de systèmes. Il y a des outils logiciels légers qui sont dédiés spécialement aux réseaux de capteurs que ce soient des systèmes d'exploitation ou des langages de programmation. Dans cette optique, plusieurs systèmes d'exploitation et des langages ont vu le jour pour exploiter les réseaux de capteurs.

Dans ce chapitre, on présente les systèmes d'exploitation pour les réseaux de capteurs en se focalisant sur TinyOs qui est considéré comme un système d'exploitation complet et réputé. Puis, on présente en détails un langage de programmation orienté composants « NesC » et d'autres outils logiciels tels que SerialForwarder, MIG, etc.

En outre, dans le cadre de notre projet, on va utiliser des capteurs TelosB, une plateforme de "Crossbow Technology" qui est beaucoup utilisée en recherche

II.2 Le système d'exploitation "TinyOs"

Il existe plusieurs systèmes d'exploitation qui sont conçus pour les réseaux de capteurs, mais TinyOs est le plus réputé et le plus complet.

Dans cette section, on va décrire en détails le mode de fonctionnement de la plateforme TinyOs. Cette description va nous permettre par la suite de valider un exemple d'une application orientée événement pour les réseaux de capteurs sans fil.

II.2.1 Présentation de TinyOs

TinyOs (**Tiny Microthreading Operating System**) est un système d'exploitation open-source spécialement conçu pour les applications embarquées fonctionnant en réseaux et, en particulier, pour les réseaux de capteurs sans fil. Il a été développé au sein de l'université de Berkeley en Californie et il est supporté par plusieurs plateformes (« Mica », « Telos »,«Iris »,….Etc).

TinyOs respecte une architecture basée sur une association de composants, réduisant la taille du code nécessaire à sa mise en place. Cela s'inscrit dans le respect des contraintes de mémoires qu'observent les réseaux de capteurs. En outre, il utilise une architecture orientée événement et il est écrit entièrement en NesC, un langage d'ANSI C, qui signifie Network Embedded Systems C en anglais.

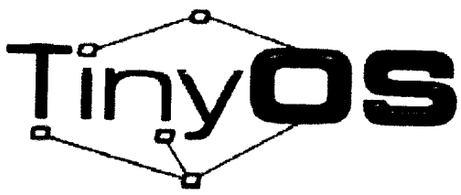


Figure II.1 : Symbole de TinyOs

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 peut être utilisé tel quel, il peut aussi être adapté à une application précise [13].

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

II.2.2 Propriétés de TinyOs

TinyOs est caractérisé par quatre propriétés qui font de lui un système d'exploitation qui s'adapte bien aux systèmes à faible ressources:

a) Evénementiel

Le fonctionnement d'un système basé sur TinyOs tel que les réseaux de capteurs s'appuie sur la gestion des événements qui se produisent lors de son déploiement. Ainsi, l'activation de tâches, leur interruption ou encore la mise en veille des capteurs s'effectue à l'apparition d'événements. Ces derniers ont la plus forte priorité.

Par ailleurs, ce fonctionnement événementiel connu sous le nom "Event/Driven" s'oppose au fonctionnement dit temporel "Time/Driven" où les actions du système sont gérées par une horloge donnée.

b) Non préemptif

Le caractère préemptif d'un système d'exploitation précise si celui-ci permet l'interruption d'une tâche en cours. 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.

c) Pas de temps réel

Lorsqu'un système est dit temps réel ; celui-ci gère des niveaux de priorité dans ses tâches permettant de respecter des échéances (date de fin de tâche) données par son environnement. Dans le cas d'un système strict, aucune échéance ne tolère de dépassement contrairement à un système temps réel. TinyOs se situe au-delà de ce second type car il n'est pas prévu pour avoir un fonctionnement temps réel.

d) Consommation

TinyOs a été conçu pour réduire au maximum la consommation en énergie du capteur. Ainsi, lorsqu'aucune tâche n'est pas active, elle se met automatiquement en mode veille [10,12].

II.2.3 Primitives de TinyOS

TinyOs est un système d'exploitation dédié pour les réseaux de capteurs qui sont caractérisés par une limitation considérable des ressources c'est pourquoi qu'il implémente des primitives aussi fine pour pouvoir gérer et fonctionner correctement sur ce type de réseaux :

a) L'ordonnanceur TinyOs

L'ordonnanceur TinyOs est défini au cœur de la gestion des tâches et des événements du système. Son choix détermine le fonctionnement global du système et permet de le doter de propriétés précises telles que la capacité à fonctionner en événementiel. En outre, l'ordonnanceur TinyOs dispose d'une file d'attente dont la capacité est sept et comporte deux niveaux de priorité: bas pour les tâches et haut pour les événements.

Par ailleurs, entre les tâches, un niveau de priorité est défini permettant de classer les tâches, tout en respectant la priorité des interruptions (ou évènements). De ce fait, lors de l'arrivée d'une nouvelle tâche, celle-ci sera placée dans la file d'attente en fonction de sa priorité (plus elle est grande, plus le placement est proche de la tête de la file).

Dans le cas où la file d'attente est pleine, la tâche dont la priorité est la plus faible sera automatiquement enlevée de la file [11].

b) Communication et Synchronisation

Les notions de synchronisation et de communication sont étroitement liées et l'existence de l'une ne peut s'envisager sans faire référence à l'autre. En général, les processus d'une application n'évoluent pas de façon indépendante. La spécification de l'application fixe les relations logiques et temporelles entre ses processus.

Dans le but de synchroniser ces processus, on fait communiquer ces processus à l'aide de mécanismes offerts par le système. Parmi les mécanismes existants pour les systèmes spécialement pour TinyOs, on peut citer:

- **Communications par Messages :** Pour éviter les problèmes liés à l'exécution en parallèle des processus et résoudre les problèmes que pose la synchronisation, le mécanisme le plus simple est celui de la boîte aux lettres qui est généralement implémentée avec une seule case mémoire ou avec plusieurs sous forme de file de messages. Les échanges s'effectuent de façon indirecte où un message est placé en mémoire par l'émetteur sera récupéré depuis cette case par le récepteur.
- **Communications par zone commune :** La façon la plus simple, la plus économique en termes d'énergie et la plus rapide pour transmettre des données est de ne pas les transmettre directement. Il suffit de disposer d'une zone de mémoire commune dans laquelle seront disposées les données en libre accès. Le problème que pose ce type de communication est surtout la synchronisation entre les processus. Par ailleurs, la synchronisation dans TinyOs fournit plusieurs outils, parmi lesquels on cite:
  - **Synchronisation par Rendez-vous :** on fait distinguer deux types de synchronisation : *Rendez-vous unilatéral* et *Rendez-vous bilatéral*. Un rendez-vous unilatéral est un point où un processus se synchronise avec élément externe, sans que l'agent extérieur ait besoin de se synchroniser avec ce processus. Un rendez-vous bilatéral peut être considéré comme un double rendez-vous unilatéral où les deux processus doivent se synchroniser avant de continuer.

- **Synchronisation par événements :** TinyOs utilise le mécanisme de *drapeaux d'événements* (*event flags*) pour signaler aux processus qu'un *événement* asynchrone s'est produit. Le principe d'attente sur un événement est relativement simple. Si le drapeau correspondant est positionné lorsque la requête d'attente est évoquée, la tâche continue son exécution. Si le drapeau est à zéro, le processus est suspendu jusqu'à ce que le drapeau soit positionné. Un processus peut attendre sur plusieurs événements. Pour cela, il existe deux manières d'attendre: une *synchronisation conjonctive* qui fait reprendre le processus lorsque tous les événements signalés ont eu lieu et une *synchronisation disjonctive* qui fait reprendre le processus lorsque n'importe lequel des événements signalés a eu lieu.

- **Synchronisation par échange de messages :** Deux tâches synchronisées par un message dont le contenu est important (résultat d'un traitement par exemple).

- **Exclusion mutuelle** : Utilisée pour le contrôle des ressources partagées du système. L'exclusion mutuelle empêche l'accès à une donnée, à un composant, ou à un périphérique si celui-ci est utilisé par un autre processus. Il existe plusieurs méthodes pour assurer l'exclusion mutuelle: Masquage des interruptions, Variables verrous, Sémaphores, etc.

c) Allocation de la mémoire

Il est important de préciser de quelle façon un système d'exploitation aborde la gestion de la mémoire, d'autant plus lorsque ce système travaille dans un environnement aussi restreint comme celui des capteurs.

TinyOs occupe un espace mémoire très faible puisqu'il ne prend que 300 à 400 octets dans sa distribution minimale. En plus de cela, il est nécessaire d'avoir 4 Ko de mémoire libre qui se repartit de la façon suivante comme montre la figure II.2:

- **La pile (Stack) :** Sert de mémoire temporaire au fonctionnement du système notamment pour l'empilement et le dépilement des variables locales.
- **Les variables globales :** Réservent un espace mémoire pour le stockage de valeurs pouvant être accessibles depuis des applications différentes
- **La mémoire libre :** Pour le reste du stockage temporaire.

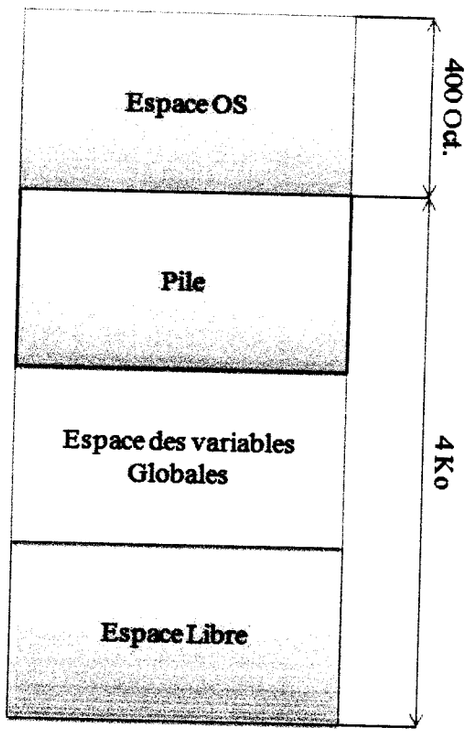


Figure II.2 : Schéma d'allocation de la mémoire

La gestion de la mémoire possède de plus quelques propriétés. Ainsi, il n'y a pas d'allocation dynamique de mémoire et pas de pointeurs de fonctions. Bien sur cela simplifie grandement l'implémentation. Par ailleurs, il n'existe pas de mécanisme de protection de la mémoire sous TinyOs, ce qui rend le système particulièrement vulnérable aux crashs et corruptions de la mémoire.

d) Gestion des Entrées/Sorties

Dans un système embarqué, le matériel et le logiciel sont liés, ainsi le matériel et le logiciel ne sont pas aussi facilement discernables. Comme le système embarqué TinyOs est conçu de façon à fonctionner sur un grand nombre d'architectures, une couche appelée HAL (*Hardware Adaptation Layer*) a été conçue pour encapsuler toutes les fonctions dépendantes de l'architecture.

e) Gestion de la connectivité (Networking)

Le support de la connectivité dans les systèmes embarqués est très important car il leur permet de communiquer facilement avec l'extérieur mais aussi de les gérer, les mettre à jour, etc. à distance! Sans oublier que parfois, la connectivité est la fonction principale du système tel que les réseaux de capteurs, routeur, téléphone portable, PDA,

etc. Dans cette optique, TinyOs implémente de nombreux protocoles pour la communication réseaux et offre aux utilisateurs une palette de protocoles de routage car plus de 70% des opérations exécutées par un capteur sont des routages de paquets.

f) Gestion des Interruptions

Dans TinyOs, la gestion des interruptions est d'une grande importance car l'interaction avec l'environnement se fait à travers des interruptions. Le traitement d'une interruption se fait à l'aide d'une routine correspondante (ISR *Intrrupt Service Routine*). TinyOS est aussi appelé système piloté par interruptions (*Interrupt Driven Systems*) car la grande partie des traitements se fait durant les ISRs et le système passe la plupart de son temps dans un mode basse-énergie. Pour réduire le temps de latence du aux changements de contexte (tâche courante => ISR => ancienne tâche), le nombre d'interruptions à gérer a été limité et aussi d'utiliser des masques d'interruption pour ne prendre en compte que les interruptions urgentes, et laisser les autres interruptions pour plus tard, dans le cas où plusieurs interruptions ont lieu.

II.2.4 Cibles de TinyOS

Il existe de nombreuses cibles possibles pour ce système d'exploitation embarqué. Malgré leurs différences, elles respectent toutes globalement la même architecture qui est basée sur un noyau central autour duquel s'articulent les différentes interfaces d'entrée-sortie, de communication et d'alimentation.

II.2.5 Tâches, évènements et applications

TinyOs est basé sur la gestion de tâches et d'évènements. Une tâche est un bloc d'instruction alors qu'un évènement est l'équivalent logiciel d'une interruption matérielle et a priorité sur les tâches. Chaque tâche est activée ou interrompue en fonction de l'apparition d'un évènement et TinyOs n'étant pas préemptif donc les tâches ne peuvent pas s'interrompre entre elles, mais elles peuvent l'être par un évènement.

a) Gestion des tâches

Chaque tâche activée est mise en attente dans une file d'attente de type FIFO. Lorsque la file des tâches est vide le système se mettra en veille en attendant le prochain évènement. Ce mécanisme de tâches a pour avantage d'empêcher une tâche d'en interrompre une autre, pouvant bloquer le système, mais il a aussi pour inconvénient de ne pas permettre une gestion en temps réel.

Pour les tâches de longue durée TinyOs possède un mécanisme permettant de fragmenter l'exécution d'une tâche nommée "split-phase" qui permet de ne pas bloquer le système. Ce mécanisme est utilisé dans l'initialisation de composants qui demandent du temps au démarrage, comme la radio par exemple.

b) Les Événements

Lorsqu'une interruption matérielle aura lieu, l'évènement correspondant reçoit un signal et prend la main de manière asynchrone, c'est à dire qu'il n'attend pas la fin de la tâche courante pour s'exécuter. Des évènements peuvent être signalés ne correspondent pas directement à une interruption matérielle.

Il existe également des évènements synchronisés qui sont mis en attente dans la liste des tâches, avec une priorité supérieure aux tâches en attente mais n'interrompent pas la tâche courante comme dans le cas des Timers.

c) Les applications

Les applications basées sur TinyOs sont formées de composants réutilisables et portables, (comme les Timers, les convertisseurs de signal ou la radio) qui sont reliés entre eux. Les composants sont implémentés en utilisant les tâches, les évènements et des commandes qui permettent de faire appel aux fonctionnalités d'autres composants auxquels ils sont liés.

II.3. Concepts de la programmation orientée composants

TinyOs peut fonctionner sur une multitude de plateformes, disponibles dès l'installation. En effet, TinyOs peut être installé à partir d'un environnement Windows ou bien GNU/Linux. Il s'appuie sur le langage NesC qui propose une architecture basée sur des composants comme montre la figure II.3, 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 réutilise dans différentes applications.

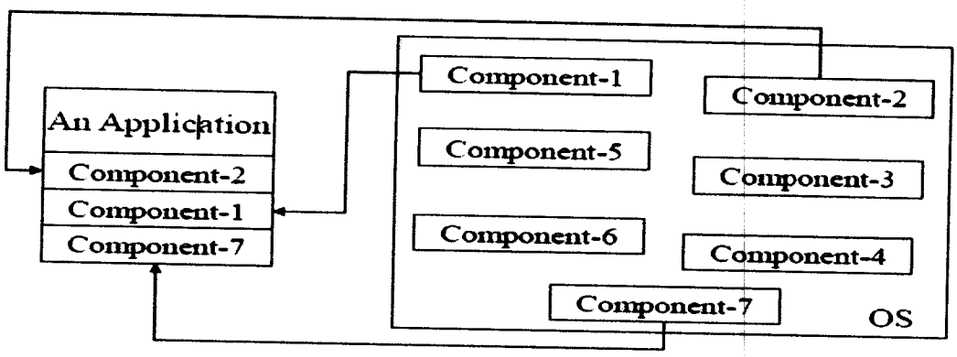


Figure II.3 : Aperçu générale de TinyOS

Chaque composant est constitué d'un frame qui reflète l'état interne du composant. Il s'agit d'un espace mémoire réservé 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 de 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 liées :

- Les tâches sont utilisées pour effectuer la plupart des blocs d'instruction 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.

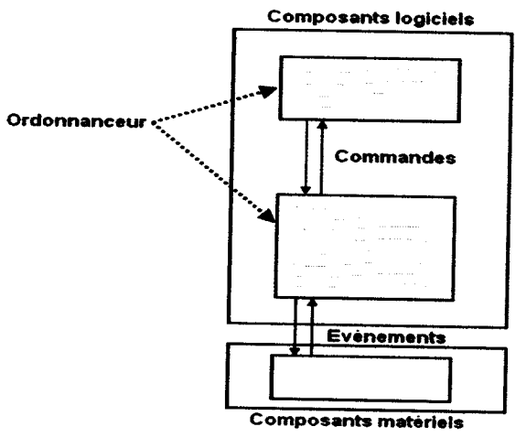


Figure II.4: Schéma des interactions internes au système TinyOS

II.4. Le langage de programmation NesC

NesC est un langage conçu pour incarner les concepts structurant le modèle d'exécution de TinyOS. C'est une extension du langage C orientée composant et il est compilé vers le langage C avant sa compilation en binaire [12].

II.4.1 Concepts de NesC

Le NesC se caractérise des éléments suivants :

- **Séparation de construction et de composition:** les programmes sont construits à partir de composants, qui sont assemblés pour former des programmes complets. Les composants sont divisés en deux blocs : un pour leur spécification (contenant les instances des interfaces), et un pour leur implémentation. Les composants sont en concurrence interne sous la forme de tâches.

  Il existe deux groupes d'interfaces: les interfaces fournies et les interfaces utilisées par le composant. Les interfaces fournies sont prévues pour représenter la fonctionnalité que le composant fournit à son utilisateur, alors que les interfaces utilisées représentent la fonctionnalité que le composant a besoin pour réaliser son travail.

- **Les interfaces sont bidirectionnelles:** les interfaces spécifient un ensemble de fonctions à implémenter par le fournisseur de l'interface (commandes) et un ensemble à implémenter par l'utilisateur de l'interface (événements). Ceci permet à une simple interface de représenter une interaction complexe entre les composants. C'est crucial car toutes les très longues commandes dans TinyOs sont non bloquantes et leur accomplissement est signalé par un événement. Donc un composant qui fait appel à une commande doit implémenter une interface qui reçoit l'événement d'accomplissement de la commande appelée. Typiquement les commandes appellent vers les niveaux bas, c.-à-d., depuis des composants d'application vers ceux plus près du matériel, alors que les événements appellent vers les couches supérieures. Certains événements proviennent d'interruptions matérielles.

- Des composants sont statiquement liés entre eux par l'intermédiaire de leurs interfaces. Ceci augmente la vitesse d'exécution, encourage une conception robuste, et tient compte d'une meilleure analyse statique des programmes.

II.4.2 Les composants de NesC

Il existe deux types de composants [11] :

a) Module

Un module est un composant qui implémente une ou plusieurs interfaces et peut utiliser une ou plusieurs interfaces, comme montre la figure II.5.

```
module nomModule{
provides{
        //liste des interfaces fournies
        interface nomInterface ;
    }
    uses{
        //liste des interfaces requises
        interface nomInterface ;
    }
}
implementation{
    //declaration des variables
    uint8 x ;
    //implementation des fonctions décrites par les interfaces fournies
}
```

Figure II.5: Syntaxe d'un module

b) Configuration

Composants reliés ensemble pour former un autre composant. 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.

```
configuration nomConfig{}
implementation{
    //liste des modules et configurations utilisées
    components Main, Module1, ..., ModuleN, Config1, ..., ConfigM

    //description des liaisons
    //interface requise -> interface fournie
    Main.StdControl -> Module1.StdControl
}
```

Figure II.6: Syntaxe d'une configuration

c) Interface

- Une interface permet de définir 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).

Exemple

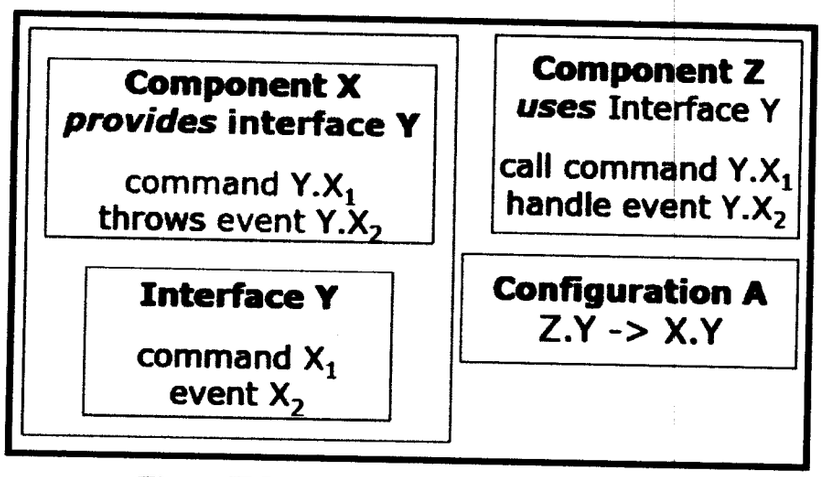


Figure II.7: Exemple d'une application TinyOS

d) Autres principes

- Un composant est constitué d'au moins un module utilisant et fournissant des interfaces. S'il contient plusieurs modules les liens entres eux sont décrits par un fichier de configuration.
- Une application complète est un composant contenant plusieurs modules liés entre eux dont un module Main qui permet de la démarrer.
- Le système d'exploitation offre une centaine de composants que l'on peut utiliser pour écrire des applications. Quand le programme est généré par le compilateur, seuls les composants utilisés (y compris ceux du système) sont présents.
- Main est lui-même connecté à certains composants du système (comme l'ordonnanceur par exemple) qui seront donc chargés en mémoire puis lancés par Main.

- On va décrire dans des fichiers portant le suffixe **.nc** les modules, les interfaces et les configurations.
- Une application comporte donc plusieurs de ces fichiers.
- Les modules offrent et utilisent des interfaces.
- Une interface déclare un ensemble de fonctions appelées commands qui peuvent être utilisées par les autres composants et un ensemble de fonctions appelées events qui réagissent à des événements en provenance d'autres composants.
- Toutes ces fonctions doivent être implémentées.

II.4.3 Structuration d'un programme en NESC

Une application est un assemblage dans lequel apparaît le composant système "Main" qui sert à démarrer. Elle est décrite par une configuration "de premier niveau". La partie "configuration" d'un fichier d'une application est vide car elle n'offre et ne requiert aucune interface, la partie implémentation décrit le schéma de câblage de plus haut niveau (assemblage des composants de l'application).

L'ensemble des configurations permet de savoir quels composants sont utiles pour que l'application fonctionne. Dans ce cas, il suffit de regarder le fichier de configuration de l'application pour savoir quels composants elle utilise (components) puis de faire la même chose récursivement pour chacun de ces composants.

a) Modèle d'exécution

TinyOs n'exécute qu'une application à la fois. Il y a deux types de processus d'exécution :

- les tâches
- les pilotes d'interruptions.

Les tâches s'exécutent l'une après l'autre sans préemption alors que les pilotes d'interruptions sont exécutés en réponse à une interruption mais peuvent préempter les tâches et les autres pilotes d'interruptions.

Les commandes et les événements qui sont exécutés par un pilote d'IT doivent être définis avec le mot clé "async".

b) 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

c) Problème de temps réel

Les commandes écrites dans les composants doivent être de courte durée car TinyOs ne fait pas de préemption donc si une commande peut durer longtemps le programmeur doit utiliser une tâche pour l'exécuter et cette tâche retournera un événement pour indiquer qu'elle est terminée. C'est par exemple le cas des commandes d'envoi sur liaison radio. L'exécution de cette tâche est différée puisqu'elle sera mise en file d'attente et exécutée à son tour quand les tâches précédentes seront terminées. En outre, les traitements d'événements sont, par nature, de courte durée.

d) Concurrence

Comme les tâches et les pilotes d'interruptions peuvent être préemptés par des pilotes d'interruption, il peut se produire des problèmes de concurrence. Traditionnellement, l'utilisation de sémaphores permet de résoudre le problème de la concurrence. NesC utilise des parties de code atomiques (mot clé **atomic**).

Le compilateur NesC détecte les problèmes de concurrence et les indique au programmeur par une erreur de compilation. Lorsqu'il est sûr que le problème ne se pose pas, le programmeur peut demander au compilateur (par le mot clé **norace**) d'accepter une construction apparemment douteuse. Bien entendu il faut utiliser cela avec circonspection.

e) Tâches

TinyOS connaît les tâches et les pilotes d'interruptions.

- Les commandes associées aux pilotes d'interruptions sont déclarées **async** pour pouvoir être exécutées en préemption des autres. Elles doivent être rapides.
- Les tâches représentent le reste de l'application et peuvent durer longtemps dans ce cas, la règle est de créer une tâche qui retourne un signal lorsqu'elle est terminée.

- Les tâches sont mises dans une file d'attente et exécutées l'une après l'autre sans temps partagé. Une tâche est déclarée par : **task void nomdeTâche() { ... }** elle n'a pas de paramètres et ne retourne rien. Elle sera mise dans la file d'attente des tâches à exécuter par : **post nomdeTâche().** Le lancement de cette tâche peut se faire dans n'importe quel code (commande ou événement) du composant.

f) Bloc atomic

Le bloc **atomic** permet de définir du code qui ne sera pas interrompu par un pilote d'interruptions :

```
atomic
{
for (i=0; i < size; i++) sum = sum + rdata[i];
}
```

Toute la boucle ci-dessus est exécutée sans interruption possible. Il ne faut pas en abuser et y mettre le minimum nécessaire sinon les IT ne seront pas prises en compte à temps.

II.5. Autres outils logiciels

Il existe d'autres outils logiciels qui permettent d'exploiter les réseaux de capteurs

II.5.1 Outils de simulation

a) TOSSIM

TOSSIM [4] est un simulateur basé sur TinyOS. Il permet de simuler le comportement d'un capteur (envoi/réception de messages via les ondes radios, traitement de l'information, ...) au sein d'un réseau de capteurs.

b) PowerTossim

L'outil PowerTossim permet de faire des simulations de la même manière que TOSSIM sauf que celui-ci prend en considération la consommation d'énergie, ainsi le

nœud qui ne possède plus d'énergie s'arrête de fonctionner, ce qui nous permet d'exécuter la simulation jusqu'à la mort du réseau.

c) TinyViz

TinyViz est une application graphique qui donne un aperçu sur un réseau de capteurs à tout instant, ainsi que des divers messages qu'ils émettent. Il permet de déterminer un délai entre chaque itération des capteurs afin de permettre une analyse pas à pas du bon déroulement des actions. Il possède aussi des options afin de pouvoir simuler la consommation d'énergie.

II.5.2 SerialForwarder

TinyOs-2.x qui est la deuxième version de TinyOs, inclut un outil de liaison entre la station de base et une application Java, nommée SerialForwarder présenté par la figure II-8. Celui-ci récupère les messages reçus par la station de base et les transmet à l'application Java. L'inverse est aussi possible, l'application Java peut envoyer des messages à la station de base (qui va les transmettre aux capteurs) via cet outil.

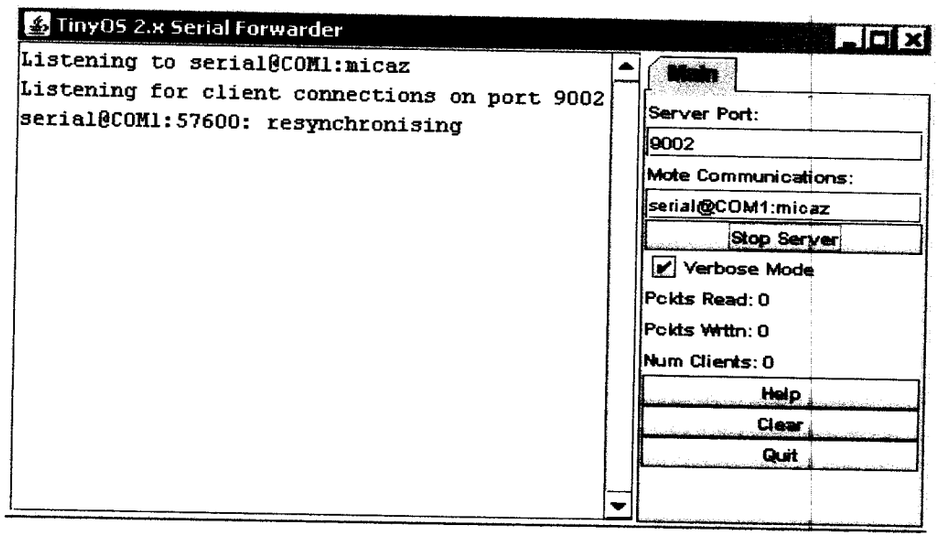


Figure II.8 : SerialForwarder

II.6 Abstraction matérielle

II.6.1 Mote, processeur, RAM et Flash

On appelle généralement Mote la carte physique utilisant TinyOs pour fonctionner. Celle-ci a pour cœur le bloc constitué du processeur et des mémoires RAM et Flash. Cet

ensemble est à la base du calcul binaire et du stockage, à la fois temporaire pour les données et définitif pour le système TinyOs.

II.6.2 Radio et antenne

TinyOs est prévu pour mettre en place des réseaux sans fil où les équipements étudiés sont donc généralement équipés d'une radio ainsi que d'une antenne afin de se connecter à la couche physique.

II.6.3 LED, interface, capteur

TinyOs est prévu pour mettre en place des réseaux de capteurs, on retrouve donc des équipements dotés de différents types de détecteurs et autres entrées.

II.6.4 Batterie

Comme tout dispositif embarqué, ceux utilisant TinyOs sont pourvus d'une alimentation autonome telle qu'une batterie.

II.7 Les composants du capteur TelosB

Dans cette section, on présente les caractéristiques des capteurs qu'on a utilisés dans l'expérimentation. Il s'agit des capteurs de la famille Telos (TelosB). Ces capteurs ont été développés par l'université de Berkeley aux USA, et fabriqués par Crossbow (figure II.9). Ils comportent les composants matériels suivants:

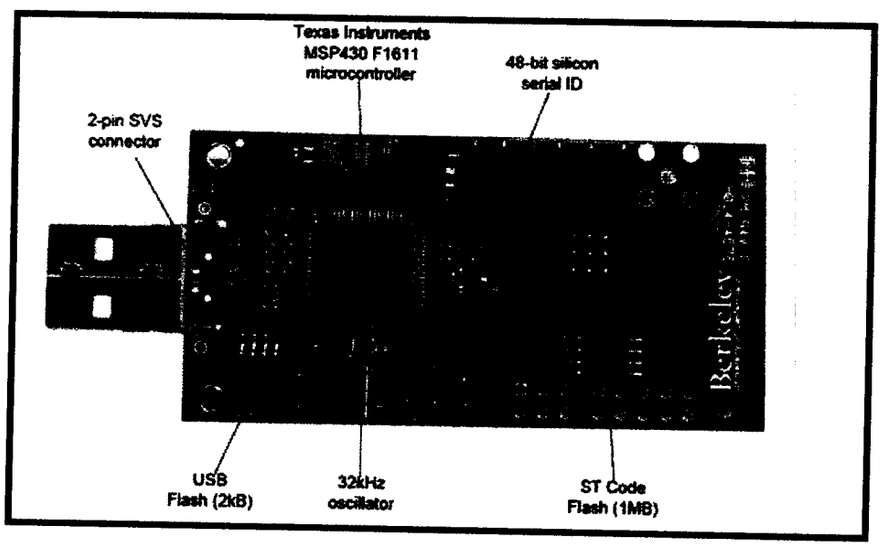


Figure II.9 : Les composants matériels du TelosB [17]

III.2 Domaines d'application

Il existe plusieurs applications qui nécessitent une surveillance continue et dans lesquelles l'information doit remontée au centre de contrôle si un événement pertinent se produit. Par exemple, le contrôle des grandeurs physiologiques d'un patient, dans ce cas si une grandeur dépasse une certaine valeur seuil telles que le taux de glycémie ou la tension artérielle, une alerte est envoyée à son médecin ou la surveillance d'un habitat si on la température dépasse une certaine valeur seuil, une alerte est remontée à un centre de contrôle pour l'aviser. L'alerte peut prendre plusieurs formes : sonore, SMS, etc.

Dans ce contexte, on pourra tirer profit des avantages des réseaux de capteurs pour développer une application "Event/Driven" pour contrôler la température que ce soit dans un habitat ou dans une forêt.

III.3 Langages utilisés

L'implémentation de cette application en impliquant les réseaux de capteurs fait appel à des outils matériels et logiciels bien spécifiques tels que NesC et Java comme langages, TinyOs comme système d'exploitation, et une plateforme de capteurs, .etc. Par exemple, dans notre cas, on a utilisé des capteurs de type TelosB.

Dans cette implémentation, on utilise deux langages différents : NesC et Java. NesC sert pour programmer les capteurs et Java permet d'utiliser ses atouts pour bien exploiter l'application au niveau du poste de contrôle par exemple la création d'interfaces graphique grâce à leur librairie Swing et aussi la gestion des communications réseau. Ce langage est supporté par Windows, Unix et Mac et dispose d'outils pour communiquer et exploiter les données envoyées par les capteurs tels SerialForwarder et MIG.

Le choix du langage s'est donc fait par affinité. Java contenant les librairies nécessaires au développement d'interfaces homme/machine et étant le seul dont le code exécutable est portable, c'est ce langage qui a été choisi pour l'application. En outre, NesC est un langage orienté composant conçu pour programmer les dispositifs à ressources limités.

III.4 Architecture de la plateforme

L'architecture de la plateforme s'articule autour de deux parties : l'installation logicielle et l'installation matérielle. La figure III.1 résume la plateforme développée.

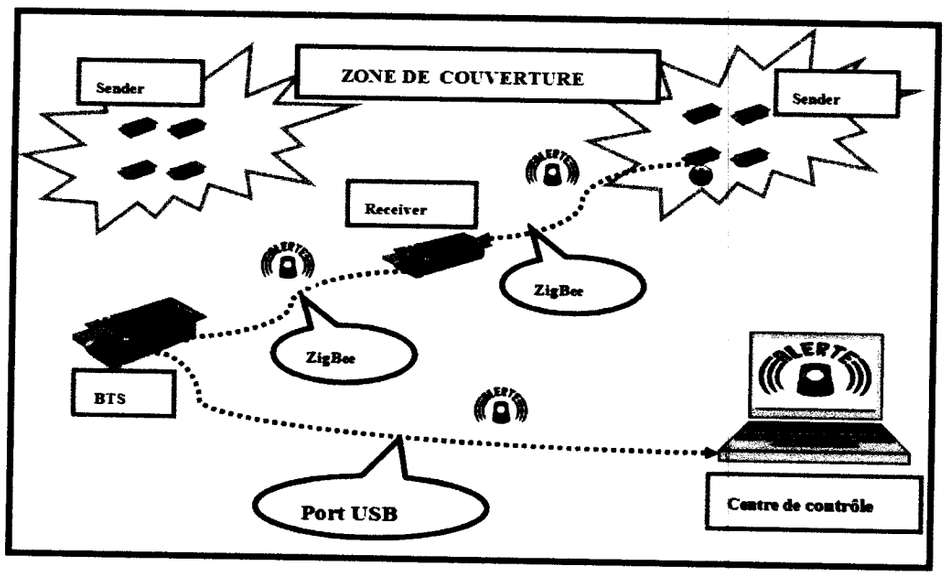


Figure III.1: Architecture de la plateforme

III.4.1 Installation logicielle

Durant cette phase, on a installé le système d'exploitation NesC, le langage NesC et le compilateur MSP430 qui est conçu spécialement pour les capteurs de type TelosB.

Pour se faire, on a choisi l'environnement Linux (Ubuntu) pour la mise en place cette application. Il est possible de l'exploiter sous cygwin (Windows) mais il y a des soucis avec la deuxième version de TinyOs (TinyOs-2.x) sous cet environnement.

La phase d'installation a été un peu délicate car il existe plusieurs versions de TinyOs et chacune d'elle nécessite son propre environnement. On a commencé par installer TinyOS 2.2.1, le langage NesC et le micro-contrôleur MSP430 pour les capteurs de type TelosB. La procédure d'installation de TinyOs se déroule en plusieurs étapes et elle est décrite dans l'annexe. Puis on a installé NetBeans au niveau du poste de contrôle pour communiquer avec les capteurs et faire visualiser les alertes sur l'écran du poste de contrôle.

III.4.2 Installation matérielle

Une fois l'installation logicielle est terminée, il a fallu installer le matériel : une station de base reliée à l'ordinateur via un câble USB, différents capteurs TelosB (Sender/Receiver) :

- Chacun des Sender mesure la température et la communique au capteur Receiver via une liaison sans fil si cette température dépasse une certaine valeur seuil.
- Un capteur Receiver joue le rôle de nœud relai et permet d'hiérarchiser la plateforme pour éviter les transmissions redondantes. Il communique avec la station de base via une liaison sans fil.
- La station de base est la destination finale de tous les nœuds capteurs et elle communique avec l'ordinateur via le câble USB.

La figure III.1 résume l'architecture de cette plateforme dans laquelle on trouve des capteurs « Sender » qui sont sensés de surveiller la zone d'intérêt. Ils sont en mode veille pour économiser leur énergie et garantir par la suite une longue longévité du réseau de capteurs. Quand un événement pertinent survient par exemple une température qui dépasse un certain seuil, le capteur qui détecte cet événement, passe en mode actif et remonte rapidement une alerte au centre de contrôle via un capteur nommé « Receiver ». Ce dernier joue le rôle de nœud relai ou un nœud de collecte. Ceci est dans le but d'éviter les messages redondants. Puis, le Receiver à son tour remonte l'information à la station de base qui est connectée au poste de contrôle par le biais du port USB.

Au niveau de la station de base le paquet reçu sera transmis au poste de contrôle après avoir fait appel à l'outil MIG qui est un générateur du code. Ceci est dans le but est d'analyser automatiquement chacun des champs du paquet pour les visualiser. Par ailleurs, le poste de contrôle est représenté par une machine puissante en termes de calcul et de stockage.

III.5 Etats de la plateforme

Dans l'architecture proposée, on définit deux états différents: « état stable » et « état d'alerte ».

II.5.1 Etat stable

Dans l'état stable, les capteurs sont en mode veille pour économiser leurs énergies. La figure III.2 illustre cet état où les capteurs sont en mode veille et il n' y pas d'alerte au niveau du poste de contrôle.

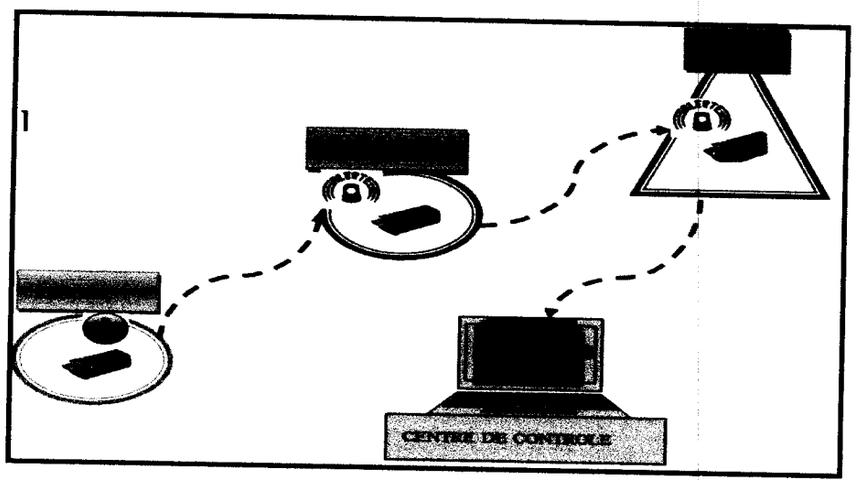


Figure III.3 : état d'alerte

III.7 Contexte d'exécution

On met le programme qui teste si la température dépasse un certain seuil dans le capteur Sender. La partie qui détecte et envoie les alertes est illustré comme suit :

```
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;
}
/* Stocker la temperature dans le paquet. */
valeur = -39.6 + 0.01 * val;
temp=(uint32_t)valeur;
if (temp > 32){

payload->temperature=b;
}

if((call AMSend.send(AM_BROADCAST_ADDR, &pkt,
sizeof(Temperature_Msg)) == SUCCESS)){
busy = TRUE;
}
}
}
}
Fin de l'envoi.
```


Après le test de la température, si sa valeur dépasse la valeur seuil un paquet qui contient cette température va être envoyé au Receiver qui va le transmettre lui-même à

III.8 Conclusion

Dans ce chapitre, on a présenté les outils logiciels et matériels ainsi que la démarche à suivre pour réaliser une application orientée événement.

Cette implémentation basée sur les réseaux de capteurs exige des outils bien spécifiques qui sont développés pour exploiter les systèmes à ressources limitées tels qu'un système d'exploitation léger « TinyOs », un langage orienté composant « NesC ».

Cette application permet de détecter les événements pertinents dans une zone d'intérêt. Dans notre contexte, on a programmé deux niveaux d'alertes.

Conclusion Générale

Conclusion Générale

L'aspect miniaturisé et les ressources limitées en termes d'énergie, calcul et stockage des réseaux de capteurs permettent l'utilisation d'outils bien spécifiques pour les exploiter.

Les réseaux de capteurs traitent deux types d'applications: les applications orientées événements et les applications orientées surveillance. Dans le premier type, on s'intéresse principalement aux événements pertinents qui peuvent survenir dans la zone de déploiement alors que dans le deuxième type on s'intéresse à une surveillance de longue durée d'une zone d'intérêt.

Dans le cadre de notre projet, on s'est intéressé aux applications « Event/Driven ». Ces applications consistent à envoyer des alertes au un centre de contrôle distant quand un événement pertinent survient. Par exemple, le suivi d'un patient, le suivi d'un animal, contrôle d'un bâtiment, etc.

Après avoir étudié les spécificités des réseaux de capteurs, on a implémenté une application orientée événement en tenant compte des contraintes matérielles des réseaux de capteurs : puissance de calcul limitée, mémoire de stockage réduite, portée de transmission courte, et une autonomie de l'énergie. Pour réaliser cette application, on a utilisé des outils logiciels spéciaux qui ont été développés pour les dispositifs à ressources limitées : TinyOs comme système d'application et NesC comme langage de programmation des capteurs.

Annexe A

Annexe

Installation de TinyOS sur Ubuntu 11.10

Installation en deux étapes sur votre système d'exploitation hôte avec les paquets Debian :

1. Taper la commande suivante dans le terminal pour ouvrir le fichier « sources.list »: **$ sudo gedit /etc/apt/sources.list**

   Ajouter la ligne suivante à la fin du fichier :

   deb http://tinyos.stanford.edu/tinyos/dists/ubuntu karmic main

2. Mettez à jour votre cache de dépôts :

   sudo apt-get update

3. Exécutez la commande suivante pour installer la dernière version de TinyOs et tous ses outils pris en charge :

   $ sudo apt-get install tinyos-2.1.1

4. ouvrer le fichier ".bashc" :

   $ gedit ~/.bashrc

Puis coller les lignes suivantes :

Export TOSROOT=/opt/tinyos-2.1.1
export TOSDIR=$TOSROOT/tos
exportCLASSPATH=$TOSROOT/support/sdk/java/tinyos.jar:.:$CLASSPATH
export MAKERULES=$TOSROOT/support/make/Makerules
export PATH=/opt/msp430/bin:$PATH
#Sourcing the tinyos environment variable setup script source /opt/tinyos-2.1.1/tinyos.sh

sudo chown yourname:yourname -R /opt/tinyos-2.1.1/

5. Finalement, taper la commande suivante pour vérifier si l'installation est terminé

   $ tos-check-env

Liste des Références

[1] Mr. Clément SAAD, « Quelques contributions dans les réseaux de capteurs sans fil: Localisation et Routage », thèse pour obtenir un diplôme de DOCTORAT, l'Université d'Avignon et des Pays de Vaucluse ,2009.

[2] Bounegta Nadia, « Approche Décentralisée pour la sécurité d'un Réseau de Capteurs Sans Fil (RCSF)», Mémoire de Fin d'étude Pour l'obtention du diplôme d'ingénieur d'état en informatique, l'Université de Bechar, juin 2010.

[3] Mr. fares Abdelfattah, « Développement d'une bibliothèque de capteur sans fil », diplôme de master en informatique, université Montpellier 2, avril 2008

[4] Wassim ZNAIDI, « Modélisation formelle de réseaux de capteurs à partir de TinyOS », projet fin d'étude, école polytechnique de Tunisie, 2006.

[5] H. Alatrista, S. Aliaga, K. Gouaich, J. Mathieu, « Implémentation de protocole sur une plateforme de réseaux de capteurs sans-fils », mémoire master, Université de Montpellier 2,25.5.2008.

[6] TinyOS Team, "TinyOS." [Online].Disponible: www.tinyos.net

[7] David Gay, David Culler, Philip Levis, "NesC Language Reference Manual", 2002, [Online] Disponible : http://NesCc.sourceforge.net/papers/NesC-ref.pdf.

[8] P.Levis and N.Lee, TOSSIM: A Simulator for TinyOS Networks, Incluse avec le software TinyOS 1.1.0, Septembre 2003

[9] CAYIRCI, E. (2004). "Wireless sensor networks". In : D. Katsaros et al. (Éd), Wireless information highways (pp. 273-301). Hershey: Idea group Inc.

[10] C.Y. Chong and S.P. Kumar. Sensor Networks: Evolution, Opportunities, and Challenges. In Proceedings of the IEEE, vol.91, no.8, pp. 1247-1256, 2003.

[11] Yazeed Al-Obaisat, Robin Braun "On Wireless Sensor Networks: Architectures, Protocols, Applications, and Management" Institute of Information and Communication Technologies University of Technology, Sydney, Australia.

[12] POTTIE, G. J. et KAISER, W. J. (2000). "Wireless integrated network sensors". ACM Communications, 43, 51-58.

[13] PANTAZIS, N. et VERGADOS, D. (2007). "A survey on power control issues in wireless sensor networks". IEEE Communications Surveys & Tutorials, 9, 86-107.

[14] AKAN, O. B. et AKYILDIZ, I. F. (2005). "Event-to-sink reliable transport in wireless sensor networks " . IEEE/ACM Transactions on Networking, 13, 1003-1016.

[15] TIAN, D. et GEORGANAS, N. (2002). "A coverage-preserving node scheduling scheme for large wireless sensor networks". Proc. 1st ACM international workshop on Wireless sensor networks and applications (WSNA). ACM, New York, NY, USA, 32-41.

[16] Adel CHOUHA, « Traitement et Transfert d'images Par Réseau de Capteurs sans Fil ». Mémoire Magistère, Université Hadj Lakhder – Batna, Mars 2011.

[17] Nicolas ESTEVES, « Capture de Mouvement par Réseau de Capteurs Sans Fil ». Juillet 2007.

Implémentation d'une application orienté événement pour les RCSF

L'objectif principal de ce mémoire est de développer une application de type Event-Driven pour les Réseaux de Capteurs Sans Fil dédiés aux applications critiques. Ce type d'application permet de minimiser l'énergie consommée par les capteurs et garantir par la suite une longue durée de vie à ces réseaux car il y aura transmission que s'il y a un événement pertinent qui survient et évite les transmissions périodiques qui sont coûteuses en termes d'énergie.

La réalisation de cette application nécessite l'utilisation d'outils logiciels conçus spécialement aux dispositifs à ressources limitées. Ces outils concernent le système d'exploitation « TinyOs » et un langage orienté composant NesC.

Mots clés: Réseaux de capteurs Sans Fil, event-driven, TinyOs, NesC.

implementation of an event-driven application for RCSF

The main objective of this work is to develop an "Event-Driven" application for critical sensor network applications. This kind of applications enables to minimize consumed energy and consequently guarantee a long network lifetime because there will be transmission only if there is a specific event and avoids the periodic transmissions which consume more energy.

The realization of this application requires the use of specific software tools such as TinyOs and NesC which are designed for equipments with limited resources as sensor networks.

Keywords: " Wireless Networks Sensor ", "event-driven", "TinyOs", "NesC".

انجاز تطبيق الموجه للحدث لشبكات الاستشعار اللاسلكية

ان الهدف المتجلي من هده المذكرة هو تطوير هدا النوع من التطبيقات "الموجه للحدث" على أجهزة شبكات الاستشعار اللاسلكية وهذا من أجل التقليل من الطاقة المستهلكة من قبل عقدة المصدر وبالتالي زيادة في متوسط عمر الشبكة حيث انه لن يتم اي ارسال إلا إذا كان هناك حالة محددة وسيتم تجنب الارسال الدوري الذي هو مكلف من حيث الطاقة.

و تحقيق هذا التطبيق يتطلب استخدام لنظام التشغيل "TinyOs" و للغة البرمجة "NesC" المخصصين لشبكات الاستشعار.

كلمات المفتاحية: "شبكات الاستشعار اللاسلكية"، "الموجه للحدث"، "TinyOS"،"NesC".