Etude en vue de la réalisation de logiciels bas niveau dédiés aux réseaux de capteurs sans fil : microsystème de fichiers

Gil de Sousa

N° d'Ordre : D.U. : 1867
EDSPIC : 411

Université Blaise Pascal – Clermont II

ECOLE DOCTORALE
SCIENCES POUR L'INGENIEUR DE CLERMONT-FERRAND

Thèse

Présentée par

Gil DE SOUSA
Ingénieur en Informatique

pour obtenir le grade de

DOCTEUR D'UNIVERSITE
Spécialité : Informatique

Etude en vue de la réalisation de logiciels bas niveau dédiés aux réseaux de capteurs sans fil : microsystème de fichiers

Soutenue publiquement le 27 octobre 2008 devant le jury :

M. Alain Quilliot	Président
Mme Edwige Pissaloux	Rapporteur
M. Bernard Tourancheau	Rapporteur
M. Jean-Pierre Chanet	
M. Christophe de Vaulx	
M. Kun-Mean Hou	Directeur de thèse

A ma famille
A Anne-laure

Remerciements

Je tiens à exprimer ma gratitude aux personnes qui m'ont fait l'honneur d'être membre de mon jury de thèse : Mme Edwige Pissaloux et M. Bernard Tourancheau en tant que rapporteurs, M. Alain Quilliot en tant que président, ainsi que M. Christophe de Vaulx et Jean-Pierre Chanet. Je les remercie pour le temps qu'ils m'ont accordé ainsi que pour les remarques et autres suggestions dont ils m'ont fait part sur mon travail.

Je remercie également mon directeur de thèse M. Kun-Mean Hou, toujours prêt à prodiguer des conseils, à faire partager son savoir et à faire part de nouvelles idées avec un enthousiasme communicatif.

Beaucoup de personnes ont contribué à la réussite de cette thèse, que ce soit pour m'apporter leur aide, leur conseil ou leur soutien. Je n'aborderai pas l'exercice périlleux de les citer toutes « nominativement » de peur d'en oublier certaines.

En premier lieu, je remercie les membres des équipes SMIR et COPAIN associées respectivement au LIMOS et au Cemagref que j'ai pu côtoyer.

Je tiens en outre à remercier les différentes personnes appartenant à ces deux instituts de recherche qui ont permis l'aboutissement de cette thèse. Cela comprend les personnes en prise directe avec la recherche et celles assurant le bon fonctionnement en termes administratif, logistique et d'organisation de ces deux instituts.

J'ai également une pensée pour les nombreuses personnes avec lesquelles j'ai eu l'occasion de discuter, durant ces années de thèse, de différents sujets en rapport avec la recherche et l'enseignement. Ces échanges ont parfois été à l'origine de nouvelles idées ou de réponses à des problèmes rencontrés.

Je tiens également à remercier mes amis. Même si l'on ne se voit pas aussi souvent que l'on le souhaiterait, le lien qui nous unit est là et, je l'espère, continuera à exister durant encore de nombreuses années.

Enfin, je remercie ma famille proche pour tout. Cette thèse est, en partie, la leur.

Résumé

De nombreux travaux de recherche actuels s'intéressent aux réseaux de capteurs sans fil (RCSF) et à leurs différentes problématiques. L'une d'entre elles est la gestion des données présentes au sein du RCSF. Généralement, les deux grands types de données manipulées sont soit celles collectées à l'aide d'un dispositif de mesure, soit celles gérées par le système d'exploitation.

L'objectif de cette thèse est de proposer des solutions à cette problématique. Un microsystème de fichier a ainsi été conçu en prenant comme support un noyau temps réel au fonctionnement hybride à la fois multitâche et basé sur les événements. Ce noyau utilise un concept permettant d'offrir un niveau d'abstraction pour la gestion des processus ou des événements. Ce concept a été repris, au niveau du microsystème de fichiers, dans le cadre de l'accès aux données.

L'autre caractéristique principale de ce microsystème de fichiers, par rapport aux systèmes existants, est de réunir, au sein d'un même système, des fonctionnalités de gestion de mémoire et d'interrogation de données.

Ces deux éléments, que sont le microsystème de fichiers et le noyau temps réel, associés à un capteur sans fil multi-composant constituent une plateforme adaptative permettant la mise en place d'applications d'acquisition de données environnementales.

Mots-clefs : Réseaux de capteurs sans fil, gestion des données, noyau temps réel, microsystème de fichiers, capteur sans fil multi-composant, acquisition de données environnementales

Abstract

Many recent research works deal with wireless sensor networks and their various problems. One of them is the management of WSN data. Generally, the two main types of handled data are either those collected using a sensor that measures a physical quantity, or those generated and managed by the operating system.

The purpose of this thesis is to propose solutions to this problem. A micro-file system has then been designed from a hybrid real-time kernel which is both a native event-driven and multithreading operating system. This kernel implements a concept that offers an abstraction level for the management and the scheduling of process and events. This concept was adapted, in the micro-file system, in order to achieve data access.

The other main characteristic of this micro-file system, in comparison with existing systems, is to merge, within the same system, functionalities about memory management and data interrogation.

These two elements, namely the micro-file system and the real-time kernel, associated with a multi-component wireless sensor provide an adaptive platform dedicated to environmental data collection applications.

Keywords: Wireless sensor networks, data management, real-time kernel, micro-file system, multi-component wireless sensor, environmental data collection

Table des matières

Liste des figures

Chapitre 3 : Le système de fichiers interrogatif LiveFile

Chapitre 4 : La plateforme LiveNode

Liste des tableaux

Liste des acronymes

A	
ADC	Analogic to Digital Converter
ADT	Abstract Data Type
AHA	American Heart Association
AODV	Ad hoc On-Demand Distance Vector routing
API	Application Programming Interface
ASCII	American Standard Code for Information interchange
B	
BPM	Bi-Phase Modulation
C	
(Bus) CAN	(Bus) Controller Area Network
CIV	Communication Inter-Véhicule
CIVIC	Communication Inter Vehicle Intelligent and Cooperative Communication Inter Véhicules Intelligente et Coopérative
CPU	Central Processing Unit
CRC	Cyclic Redundancy Checking
CSMA	Carrier Sense Multiple Access
CSMA/CA	CSMA / Collision Avoidance
CTS	Clear To Send
D	
DARPA	Defense Advanced Research Projects Agency
DCE	Data Centric Entity
DCS	Data Centric Scheduler
DI	Deadline Inheritance
DLM	Dynamic Loadable Module
DRAM	Dynamic Random Access Memory
DREAM	Distributed REAl-time Micro-kernel
DSDV	Destination-Sequenced Distance-Vector routing
DSN	Distributed Sensor Networks
DSR	Dynamic Source Routing protocol
DSSS	Direct Sequence Spread Spectrum
DT	Data Type
E	
E-CIVIC	Embedded-Communication Inter Vehicle Intelligent and Cooperative Communication Inter Véhicules Intelligente, Coopérative et Embarquée
EDF	Earliest Deadline First
EDFI	Earliest Deadline First Inheritance
EEPROM	Electrically Erasable Programmable Read Only Memory
ELF	Efficient Log-structured Flash File system
F	
FHSS	Frequency Hopping Spread Spectrum
FZRP	Fisheye Zone Routing Protocol
G	
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
GZIP	GNU Zip
H	
HR-DSSS	High Rate-DSSS

I

I²C	Inter Integrated Circuit (Bus)
IEEE	Institute of Electrical and Electronics Engineers

J

JPEG	Joint Photographic Experts Group

L

LAR	Location-Aided Routing
LIMOS	LIghtweight Multithreading Operating System
LiveFile	LIMOS Versatile Embedded File system
LiveNode	LIMOS Versatile Embedded Node
LZS	Lempel-Ziv-Stac
LZW	Lempel-Ziv-Welch

M

MAC	Medium Access Control
MANTIS	MultimodAl system for NeTworks of In-situ wireless Sensors
MP3	MPEG-1 Audio Layer 3
MPEG	Moving Picture Experts Group

N

NAT	Network Address Translation
NIMBE	Non Invasive MANET Bandwidth Estimator

O

O-QPSK	Offset-Quadrature Phase Shift Keying
OFDM	Orthogonal Frequency Division Multiplexing
OLAP	On-Line Analytical Processing
OLAR	Obstacle Location-Aided Routing
OLSR	Optimized Link State Routing protocol
OSI	Open Systems Interconnection
OTT	One-way Transit Time

P

PDA	Personal Digital Assistant
PNG	Portable Network Graphics
PPM	Pulse Position Modulation
PSM	Pulse Sequence Modulation

Q

QdS	Qualité de Service

R

RAM	Random Access Memory
RCSF	Réseaux de Capteurs Sans Fil
RFID	Radio Frequency IDentification
RISC	Reduced Instruction Set Computer
RLE	Run-Length Encoding
RTS	Request To Send
RTT	Round-Trip Time

S

SDREAM	Super-small Distributed REal-time Micro-kernel
SGBD	Système de Gestion de Bases de Données
SMP	Sensor Management Protocol
(S)NTP	(Simple) Network Time Protocol
SPI	Serial Peripheral Interface
SQL	Structured Query Language

SRAM	Static Random Access Memory
SSL	Secure Sockets Layer
T	
TDMA	Time Division Multiple Access
TFFS	Transactional Flash File System
TLS	Transport Layer Security
U	
UAL	Unité Arithmétique et Logique
UART	Universal Asynchronous Receiver Transmitter
USART	Universal Synchronous Asynchronous Receiver Transmitter
USB	Universal Serial Bus
UWB	Ultra-Wide Band

Introduction

Les réseaux de capteurs sans fil (RCSF) sont une thématique de recherche en pleine expansion. Durant les dix dernières années, de nombreux travaux et études ont été menés pour répondre aux différentes problématiques posées. Tout développement réalisé dans les RCSF est tenu de respecter des contraintes en rapport avec les ressources disponibles et le niveau de miniaturisation souhaitée. Ces contraintes ont eu pour conséquence soit de créer des problématiques nouvelles, soit d'amplifier certaines déjà existantes comme, par exemple, celles relatives à la communication sans fil. Ainsi, tout module de communication sans fil doit, en plus d'assurer la transmission de données, consommer un minimum d'énergie pour pouvoir envisager son utilisation dans un RCSF. De plus, les ressources d'un capteur sans fil que sont l'énergie, la mémoire et la puissance de calcul sont liées entre elles. Généralement, la préservation de l'une d'entre elles implique une consommation plus importante de l'une ou des deux autres. Le bon fonctionnement de l'application supportée passe donc par une gestion intelligente des ressources présentes au sein de chaque capteur du réseau.

Contexte de l'étude

Le nombre d'applications associées aux RCSF n'a cessé de croître au fur et à mesure des différentes avancées réalisées dans les domaines, entre autres, des systèmes embarqués ou des protocoles de routage Ad Hoc. Celles-ci peuvent être classées de différentes manières. La première consiste à les regrouper par domaines d'applications :

- les applications militaires ;
- les applications de la vie quotidienne ;
- les applications environnementales ;
- les applications industrielles ;
- les applications médicales etc.

La seconde prend, comme critère de classification, leur mode de fonctionnement :

- les applications d'acquisition de données ;
- les applications en remplacement d'une infrastructure de réseau filaire.

Pour ne pas redévelopper, à chaque nouvelle application, l'ensemble des composants précédemment élaborés, la conception d'un dispositif complet et adaptatif a été envisagée et constitue le point de départ de l'étude présentée dans ce mémoire. Généralement, la réalisation d'un dispositif de ce type se résume à développer un nombre important de modules logiciels qui viennent se rajouter au système d'exploitation en charge de la gestion de la partie électronique du capteur sans fil. Ces deux derniers éléments ne subissent que peu de modifications d'une application à l'autre. Cette approche consiste donc à conserver une base matérielle et logicielle à peu près fixe et à développer un nombre conséquent de composants logiciels dédiés, pouvant être ajoutés ou supprimés suivant les besoins. La démarche choisie dans ce mémoire va plus loin en considérant que tous les éléments formant le capteur sans fil sont modulables et reconfigurables avec un grand degré de liberté.

Objectifs et contributions de la thèse

Pour atteindre cet objectif, la première étape a consisté à ce que le capteur sans fil LiveNode (LIMOS Versatile Embedded Node) utilisé soit construit en combinant une ou plusieurs entités matérielles simples. Chacune d'entre elles est composée d'un microcontrôleur, de modules de communication sans fil et/ou de GPS. Un dispositif réalisant le multi-support est ainsi possible en associant des entités équipées de modules de communication différents.

Pour gérer ce capteur sans fil multi-composant, un système d'exploitation hybride LIMOS (LIghtweight Multithreading Operating System) à la fois multitâches et événementiels, basé sur les composants a été développé. Ces deux éléments forment une base matérielle et logicielle adaptée à de nombreuses applications de RCSF.

La nécessité d'un système de gestion de données a été mise en évidence à partir des besoins observés dans la plupart des applications considérées. Un capteur sans fil produit différents types de données. Aux données collectées grâce à un ou plusieurs capteurs de grandeur physique et à celles provenant du transfert de fichiers entre différents nœuds, doivent être ajoutées celles liées au fonctionnement du capteur sans fil. Ces dernières sont soit générées par le système d'exploitation, ou soit par un autre composant logiciel comme celui en charge du routage. De plus, suivant la quantité et l'importance des données manipulées, leur stockage peut requérir l'utilisation d'une mémoire non volatile telle que celle de type Flash présente dans les microcontrôleurs. Les caractéristiques de ce type de mémoire imposent des traitements plus ou moins complexes pour occuper, de manière optimale, l'espace disponible et pour prolonger au maximum leur durée de vie.

La principale contribution de cette thèse est le développement du microsystème de fichiers LiveFile (LIMOS versatile Embedded File system). L'apport de ce système par rapport à ceux existants est d'offrir, au sein d'un même composant logiciel, les deux fonctionnalités suivantes :

- la gestion de la mémoire des capteurs sans fil ;
- l'interrogation des données stockées.

Le système LiveFile propose deux formats de stockage pour les données : les enregistrements et les fichiers. Les enregistrements correspondent à des structures regroupant différentes variables adaptées aux types d'informations manipulées. Des interrogations sont possibles sur ces structures en considérant chaque enregistrement comme un tuple et chaque variable associée comme une valeur d'attribut. Les fichiers sont des flux de données sans organisation particulière. Les enregistrements sont plutôt adaptés aux applications d'acquisition de données alors que les fichiers le sont plus pour celle de substitution d'une infrastructure de réseau fixe. Ainsi, un capteur sans fil qui serait temporairement dans l'impossibilité de communiquer, pourrait stocker dans sa mémoire Flash les données qu'il aurait à relayer.

Sous le système LIMOS, le concept LINDA est utilisé à la fois pour la communication et la synchronisation entre processus ou événement et pour la gestion des périphériques. Ce concept a été, une nouvelle fois, étendu pour réaliser l'intégration du système LiveFile. Au gré des différentes modifications apportées dans le cadre de cette thèse, une nouvelle version du noyau LIMOS a été progressivement élaborée.

Organisation du mémoire

Ce mémoire comporte cinq parties. Le premier chapitre est consacré à une présentation du contexte et des problématiques associées aux RCSF. Plusieurs réflexions sont menées soit en s'intéressant au comportement global du réseau soit en se focalisant sur les fonctionnalités importantes dont doit disposer un capteur sans fil. Les différents capteurs sans fil et les nombreuses applications de RCSF présentés dans cette partie confirment l'intérêt grandissant pour cette thématique de recherche.

Le deuxième chapitre a pour objet les systèmes d'exploitation dédiés au RCSF. Les différentes familles de système existantes sont illustrées par l'intermédiaire d'un système choisi pour sa représentativité. Une partie importante de ce chapitre est consacrée à la présentation du noyau temps-réel LIMOS et à son mode de fonctionnement hybride.

Le troisième chapitre est une présentation complète du système LiveFile. Quelques gestionnaires de mémoire Flash et systèmes de gestion de bases de données dédiés au RCSF sont également décrits. Leurs fonctionnalités sont, en outre, comparées à celles du système LiveFile. Les métadonnées contextuelles ainsi que la notion de propriétés font partie des concepts présentés en détails. Le rôle des métadonnées est de compresser et d'offrir un mécanisme de sécurisation simple au niveau des données stockées et transmises. Les propriétés permettent, quant à elle, d'accélérer le processus d'interrogation des données. D'autres fonctionnalités envisagées pour le système LiveFile mais non totalement développées sont introduites à la fin de ce chapitre.

Le quatrième et dernier chapitre fournit des exemples d'utilisation qui sont issus de projets soit réalisés, soit envisagés, de l'ensemble du dispositif adaptatif dans son état actuel. Les différentes versions de capteurs sans fil élaborées au sein de l'équipe SMIR du LIMOS sont d'abord présentées, le capteur sans fil LiveNode étant celui présenté avec le plus de détails. Puis, une plate-forme réalisée dans le cadre d'un projet est présentée en donnant des informations sur la configuration des différents éléments. Enfin, des cas d'utilisation de l'ensemble du dispositif non encore rencontrés dans le cadre de projet sont exposés.

La dernière partie vient conclure ce mémoire en synthétisant les différents points abordés et en fournissant différentes perspectives sur les travaux réalisés.

Chapitre 1

Les réseaux de capteurs sans fil

Les avancées technologiques de ces dernières années ont permis l'amélioration et la miniaturisation de différents dispositifs utilisés pour les applications embarquées et l'apparition de la notion de « capteur sans fil » telle qu'elle est employée aujourd'hui. Pour illustrer les progrès réalisés, on peut considérer les évolutions observées dans le domaine de la communication sans fil. L'utilisation d'onde radio pour communiquer date du début du XX^{e} siècle mais l'un des premiers protocoles de communication entre ordinateurs n'est apparu que dans les années 70 avec le protocole ALOHA. Beaucoup de chemin a été parcouru depuis pour arriver à des protocoles tels que ceux de la famille des IEEE 802.11 (Wi-Fi) et IEEE 802.15 (Bluetooth et ZigBee).

L'objectif de ce chapitre est de donner une vision d'ensemble sur les réseaux de capteurs sans fil (RCSF). Le point de départ est le capteur en lui-même qui sera ensuite intégré dans un réseau de manière à répondre à une application donnée.

1.1 Les capteurs sans fil

Un capteur sans fil reste un capteur c'est-à-dire un instrument qui transforme une grandeur physique en une grandeur quantifiable comme peut l'être une tension. Mais si la phase de détection ou de récolte des données reste identique à celle d'un capteur classique, les traitements qui en découlent ont considérablement évolué. Les changements sont de deux types. Le premier concerne l'intégration dans le capteur d'une partie des traitements de post-acquisition (voir Figure 1.1). L'augmentation des capacités en terme de mémoire et de puissance de calcul font des capteurs actuels l'équivalent d'ordinateurs de bureau datant d'une vingtaine d'années. Une partie du conditionnement des données peut donc avoir lieu au sein même du capteur. Le second changement est l'acheminement simplifié et facilité des données du lieu d'observation vers un autre lieu. Les données peuvent être transmises naturellement vers l'utilisateur mais également vers d'autres capteurs sans fil dans un mode de fonctionnement coopératif. L'apparition d'équipements de communication sans fil performants et nécessitant moins d'énergie a grandement bénéficié au développement des capteurs sans fil.

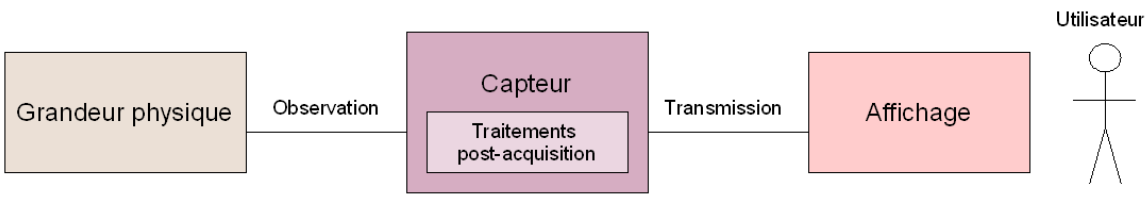

Figure 1.1 – Schématisation de la chaîne de collecte de données

Les quatre principaux éléments d'un capteur sans fil sont :

- une unité de calcul ;
- un module de communication sans fil ;
- un gestionnaire des alimentations ;
- des dispositifs de mesure de la grandeur observée ou « capteur » au sens classique du terme.

L'unité de calcul est, en général, un microcontrôleur constitué d'un processeur, d'une certaine quantité de mémoire, de convertisseurs de signaux de type ADC (Analogic to Digital Converter) et d'un ensemble d'interfaces présentes par exemple sous forme d'un port série UART (Universal Asynchronous Receiver Transmitter) ou d'un bus de type SPI (Serial Peripheral Interface) (voir Figure 1.2.).

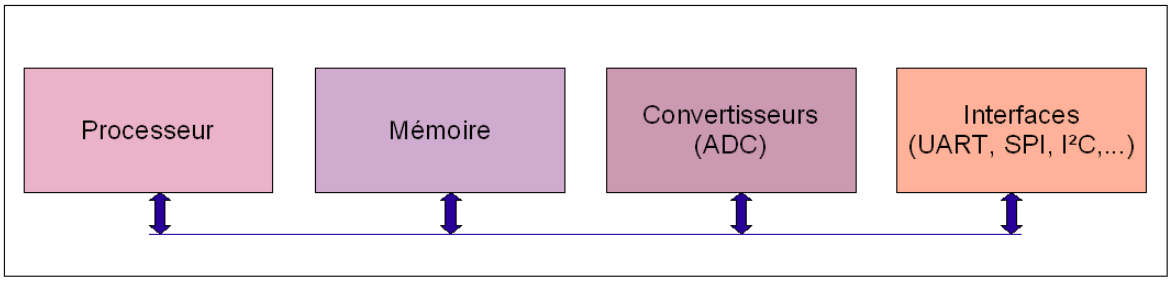

Figure 1.2 – Structure d'un microcontrôleur

Le choix du microcontrôleur utilisé s'effectue suivant la puissance de calcul mais également le niveau de consommation d'énergie. Dans un capteur sans fil, ces deux critères ne peuvent être dissociés. En effet, le microcontrôleur doit assurer sa tâche et exécuter correctement les traitements qui lui sont soumis tout en ayant une consommation d'énergie la plus basse possible. Le fait de disposer de plusieurs modes dont un de type veille est un avantage sauf si le coût énergétique de passage d'un mode à l'autre est trop important. Ce dernier point dépend de l'application : si le capteur doit effectuer un relevé par semaine ou par mois, malgré la surconsommation ponctuelle d'un changement de mode, on préconisera quand même le passage en mode veille. Actuellement, les capteurs sans fil sont équipés majoritairement de microcontrôleur de type MSP de chez Texas Instruments [MSP 2008] ou ARM et Atmega d'Atmel Corporation [Atmega 2008].

Les modules de communication implémentent soit des protocoles de la famille des IEEE 802.11 ou IEEE 802.15 soit une technologie utilisant la bande de fréquences libre. Si l'on prend l'exemple des « mica motes » de l'Université de Berkeley, la communication sans fil est obtenue grâce à un module RFM TR1000 opérant à une fréquence de 916,5MHz. L'infrarouge est également utilisé mais a pour principal inconvénient de nécessiter une ligne de communication dégagée. Pour des applications sous-marines, les ultra-sons sont une alternative.

Les recherches actuelles s'orientent vers l'utilisation des systèmes radio de type UWB (Ultra-Wide Band) (voir Table 1.1). Ces systèmes peuvent être aussi employés pour la localisation. A l'inverse, ils pâtissent pour l'instant d'une portée limitée, de problèmes techniques affectant leur performance (problème de synchronisation émetteur/récepteur) et d'un manque de standardisation.

Au départ, les capteurs intégraient des équipements de communication réalisés pour d'autres types d'applications. Désormais, avec l'intérêt grandissant autour de cette technologie, de plus en plus de travaux de recherche comme ceux sur les UWB visent à développer des dispositifs dédiés aux applications de capteurs sans fil c'est-à-dire performants au niveau de la communication et peu consommateur d'énergie.

Technologie	Fréquence	Débit maximal	Modulation
IEEE 802.11a	5GHz	54Mb/s	OFDM
IEEE 802.11b	2.4GHz	11Mb/s	DSSS / HR-DSSS
IEEE 802.11g	2.4GHz	54Mb/s	DSSS / OFDM
Bluetooth IEEE 802.15.1	2.4GHz	1Mb/s	FHSS
ZigBee IEEE 802.15.4	2.4GHz	250Kb/s	O-QPSK
IrDA	Infrarouge	4Mb/s	PPM / PSM
HDR UWB (Hotspot)	3.1-10.6GHz	100-500Mb/s	PPM / BPM
LDR/MDR UWB	3.1-10.6GHz	<1Mb/s	

Table 1.1 – Technologies de communication sans fil

L'alimentation du capteur est un point très important. La première idée consiste à s'orienter vers une source d'énergie naturelle et inépuisable comme celle fournie par les panneaux solaires. Cependant, dans la majorité des cas, cela n'est pas possible. Si l'on reste sur les panneaux solaires, d'une part ils ont un certain encombrement lié à leur relatif faible rendement et d'autre part ils sont assez fragiles. Leur utilisation n'est, par exemple, pas possible au milieu de désert tel que le Sahara. Bien qu'ils y offrent le meilleur rendement, ils peuvent être recouvert entièrement par une dune ou détériorés par une tempête de sable.

L'autre alimentation la plus répandue est l'utilisation de piles (rechargeables ou non) standards de type soit bouton soit AA dans la majorité des cas. Il s'agit d'une solution assez économique vis-à-vis de l'utilisation d'un bloc de batterie moins répandu et donc plus cher à l'achat.

Le choix de l'alimentation se fait en fonction des caractéristiques matérielles du capteur mais également des traitements qu'il aura à effectuer. Une bonne estimation de l'énergie nécessaire pour mettre en place une application est très primordiale ; un capteur sans fil pouvant avoir pour vocation de rester en place pour une durée de 6 mois voire 1 an sans intervention extérieure.

Grâce aux efforts de miniaturisation entrevus ces dernières années, les capteurs sans fil devraient, à l'horizon 2010, avoir la taille d'une poussière [Chong 2003]. Actuellement, différentes familles de capteurs peuvent être considérées, allant des capteurs de très petite taille qui effectuent des traitements simples aux capteurs plus riches en ressources, proches

d'un assistant personnel de type PDA (Personal Digital Assistant). Les différents capteurs développés au sein de l'Université de Berkeley illustrent ce point (voir Table 1.2).

			
Identification	**Mica2Dot (MPR500CA)**	**Mica2 (MPR400CB)**	**Tmote Sky**
Microcontrôleur	**ATmega128L**	**ATmega128L**	**MSP430F**
Architecture	8-Bit	8-Bit	16-Bit
Fréquence	4MHz	7.3728MHz	8MHz
Mémoire de programmation	128Ko	128Ko	48Ko
Mémoire de données	4Ko	4Ko	10Ko
Mémoire de stockage	512Ko	512Ko	1024Ko
Module radio	**CC1000**	**CC1000**	**CC2420**
Bande de fréquence	315-916MHz	315-916MHz	2.4GHz
Débit maximal	38.4Kb/s	38.4Kb/s	250Kb/s

Table 1.2 – Capteurs sans fil développés à l'Université de Californie, Berkeley

1.2 Du capteur au réseau de capteurs sans fil

Les RCSF intègrent des technologies issues des capteurs, de la communication et de l'informatique au sens large, avec le matériel, le logiciel associé à l'algorithmique. Les évolutions techniques déjà mises en évidence au début du chapitre ont permis de former des réseaux de capteurs qui ont donné naissance à de nombreuses perspectives et développements. En effet, là où un capteur seul a un rayon d'action limité à sa propre portée, un réseau de capteurs permet d'acheminer les données à un utilisateur plus éloigné et de couvrir, au même moment, une surface plus importante. L'élément important est la coopération qui doit exister entre les capteurs ou nœuds du réseau.

Un RCSF est différent d'un réseau sans fil Ad Hoc. On peut dire qu'un réseau de capteurs est un réseau Ad Hoc mais pas l'inverse. La différence se situe au niveau du nombre et des caractéristiques des nœuds qui forment le réseau. Un RCSF peut être composé d'une centaine voire d'un millier de nœuds. La gestion du réseau qui en découle est donc très particulière. Dans un avenir plus ou moins proche, un RCSF pourrait être formé de millions de nœuds microscopiques disséminés à travers un lieu à observer (voir Figure 1.3). La présence d'un nombre conséquent de nœuds au mètre carré est un facteur d'interférence important.

Toutefois, si un nœud tombe en panne, la probabilité de l'existence d'un ou plusieurs nœuds voisins pouvant le remplacer est assez élevée. Plus précisément, si la répartition des nœuds sur une surface donnée est uniforme, les risques de partition ou séparation du réseau sont moindres par rapport à un réseau sans fil Ad Hoc classique.

En général, les données collectées par les capteurs sont acheminées à travers le réseau vers une station centrale de traitement appelée « sink » en anglais. Cette station récupère donc les données, les stocke puis les met à disposition des utilisateurs (voir Figure 1.3). Dans un mode de fonctionnement plus évolué, elle sert d'interface avec le réseau en envoyant des requêtes d'interrogation ou des ordres de reconfiguration aux capteurs. Si l'on intègre le réseau de capteurs dans un réseau plus global comme celui d'une entreprise ou Internet, son rôle est celui de passerelle. En termes de dispositif, la station centrale est, soit un capteur évolué, soit un ordinateur personnel, portable ou non.

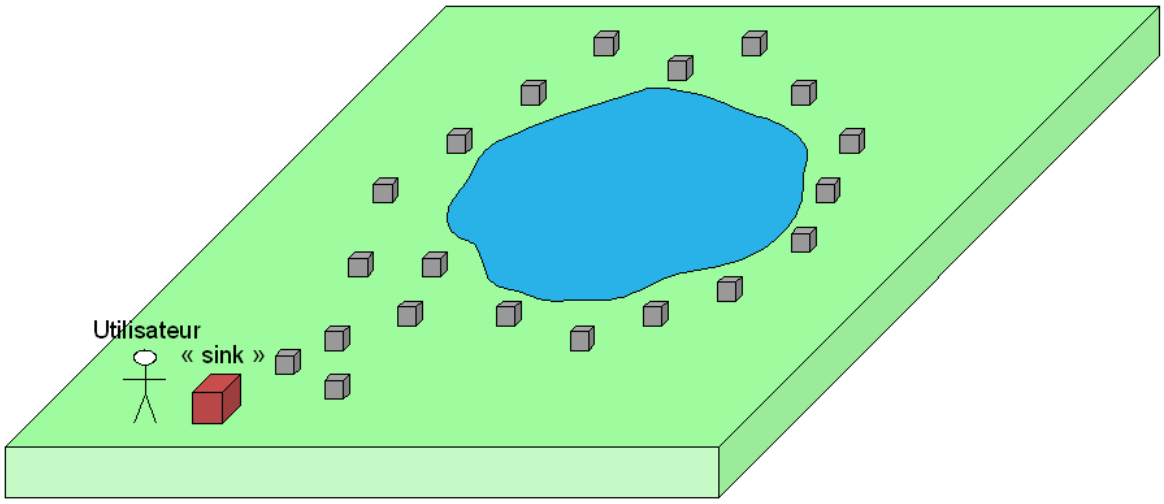

Figure 1.3 – Exemple d'organisation d'un réseau de capteurs sans fil

Le déploiement des capteurs est la première étape dans la formation du réseau [Römer 2002]. Il existe différents modes de déploiement :

- la dissémination aléatoire sur une zone donnée ;
- l'association entre un capteur et l'entité observée.

Le premier déploiement a déjà été abordé ci-dessus. Son domaine d'application est l'observation de phénomènes qui ne sont pas localisés précisément. Plusieurs points de mesure ou d'observation sont donc positionnés et les renseignements obtenus sont le résultat de la fusion des données provenant de chaque capteur.

Dans le second mode, le capteur est lié parfois physiquement à l'objet à surveiller. Les capteurs sont positionnés un à un soit manuellement soit par un robot préprogrammé. La mise en place est plus longue et plus complexe mais est à l'inverse très adaptée à l'environnement étudié et les chances d'obtenir des informations précises sont élevées. Ce mode de déploiement est également plus économique car tous les capteurs sont utiles et utilisés.

1.3 Quelques exemples d'applications des réseaux de capteurs

Les RCSF peuvent être classés selon leur domaine d'application ou la tâche qui leur est allouée. A chaque application, un scénario est défini et comprend le nombre et le type de capteurs utilisés, le rôle de chacun, la topologie du réseau (regroupement en cluster, etc.).

1.3.1 Les applications militaires

Historiquement, de nombreux progrès technologiques proviennent de la recherche militaire, qu'ils s'agissent des télécommunications, des systèmes embarqués ou dans bien d'autres domaines. Les RCSF ne dérogent pas à cette règle. Dans cette thématique, l'armée est très active et a un rôle, soit de précurseur, soit d'utilisateur [Chong 2003]. Elle étudie et teste, comme ce fut le cas pour le protocole de routage OLSR, les travaux réalisés au sein d'autres laboratoires de recherche [Plesse 2005]

Durant la guerre froide, l'armée américaine a développé un système appelé SOSUS (SOund SUrveillance System) qui permettait de sonder les océans. Par la suite, sous l'impulsion du DARPA (Defense Advanced Research Projects Agency), des recherches centrées sur les réseaux de capteurs ont été initiées dans le cadre d'un programme nommé DSN (Distributed Sensor Networks).

Les applications militaires des RCSF sont multiples. Certaines ont pour but la protection des soldats engagés sur un champ de bataille. Une première application est de fournir au soldat des dispositifs destinés à surveiller ses signes vitaux. Equipé d'un module de repérage, il est possible de localiser la position de l'ensemble des membres d'une unité et d'établir ensuite une stratégie de défense ou d'attaque.

La prévention liée aux attaques chimiques, biologiques voire nucléaires par une détection anticipée fait partie des applications envisagées pour les RCSF. Des capteurs mobiles sont envoyés en repérage sur un champ de bataille et donnent des indications sur les risques présents.

Une zone délimitée autour d'un camp de base peut être surveillée à l'aide d'un RCSF. Une telle application repose sur des dispositifs spécifiques de détection présents sur les capteurs. Ces derniers sont positionnés dans des endroits précis et en charge de l'observation d'une zone donnée. En cas d'intrusion, un message est transmis à travers le réseau en direction des entités à alerter (voir Figure 1.4).

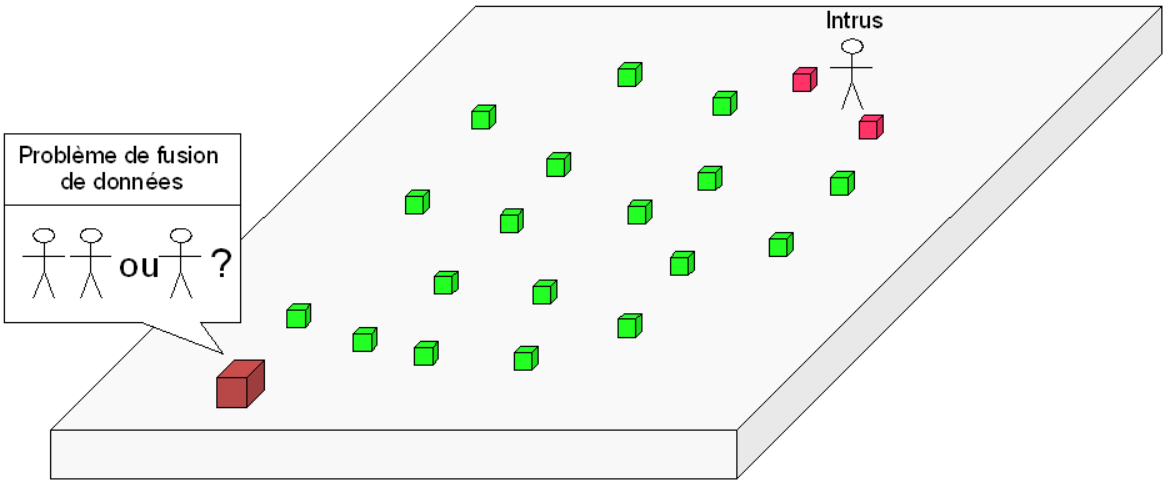

Figure 1.4 – Détection d'intrusion par un réseau de capteurs sans fil

Ce type d'application est complexe à mettre en place. Une trop grande proximité des capteurs peut entraîner des erreurs d'appréciation. En effet, si deux capteurs sont trop proches l'un de l'autre, ils vont surveiller une même zone. Si les traitements effectués après un événement ne sont pas corrects, il se peut que l'on indique deux intrusions au lieu d'une ou l'inverse.

Le suivi à la trace (ou « mobile tracking » en anglais) accompagne parfois la détection d'une intrusion (voir Figure 1.5). Localiser avec précision les forces ennemies et suivre leurs déplacements est un avantage important durant une bataille. Cette opération est obtenue en plaçant à l'avance un réseau de capteurs tout au long d'un parcours qui pourrait être emprunté par l'ennemi. Les problèmes de cohérence entre le phénomène réel et l'observation qui en découle, soulevés pour l'application précédente restent valables. Les techniques de détection utilisent des capteurs vidéo, acoustiques et de chaleur.

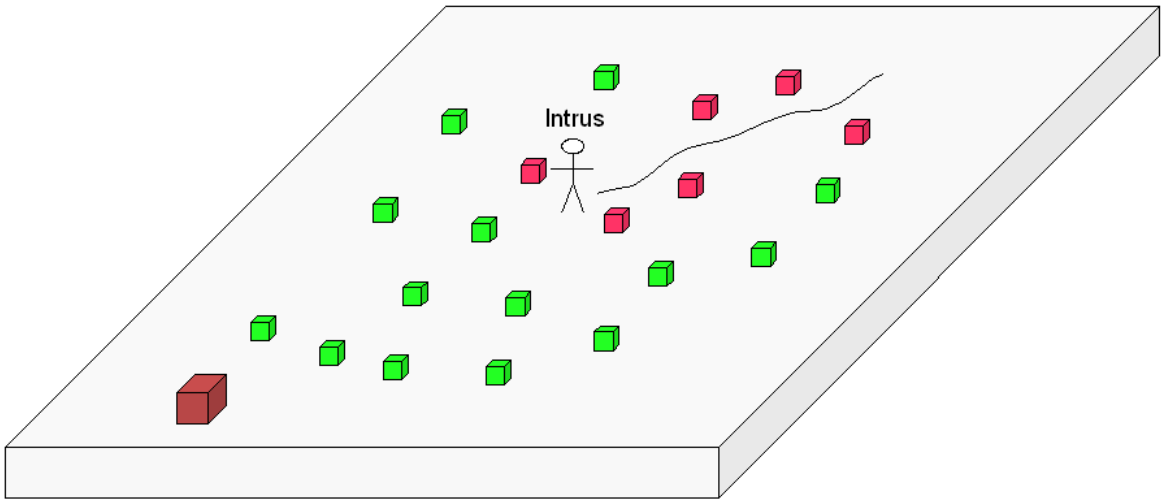

Figure 1.5 – Suivi à la trace par un réseau de capteurs sans fil

1.3.2 Les applications environnementales

Les RCSF donnent la possibilité d'étudier des phénomènes plus près que ne le permettent des dispositifs classiques. L'accessibilité à certains endroits est possible ou du moins simplifiée grâce aux RCSF. On peut prendre l'exemple d'une forêt tropicale comme la forêt amazonienne. Cette forêt est tellement dense et par certains points hostile à l'homme que l'utilisation d'un réseau de capteurs pour l'étudier pourrait être envisagée. Les capteurs pourraient être ainsi disséminés par voie aérienne sur un périmètre donné et renvoyer les données collectées par transmission sans fil. Le problème majeur dans ce mode de fonctionnement, en plus du développement de capteurs petits et robustes, est la récupération de ces mêmes capteurs après l'accomplissement de leur tâche.

Au-delà de l'exemple précédent, l'architecture d'un capteur sans fil avec un module de communication sans fil, une puissance de calcul raisonnable et un ensemble de capteurs dédiés à l'étude de grandeurs physiques rend possible son intégration dans un environnement physique donné [Karl 2003]. Les grandeurs physiques mesurées sont la température, l'humidité, la tension du sol [Jacquot 2007], le degré de luminosité, le niveau sonore.

L'observation des animaux dans leur habitat prend une place importante dans les applications environnementales [Stojmenovic 2005]. Les capteurs sans fil déployés dans une réserve naturelle [Lalooses 2007] donnent des informations de localisation sur les animaux, leur état de santé, sur leur intégration dans un nouvel habitat. Les RCSF ont permis l'observation d'oiseaux sans troubler leur habitude en évitant une intervention humaine jugée inappropriée [Mainwaring 2002].

Le projet PODS [Biagioni 2002], lui, vise à comprendre le processus et les mécanismes de croissance de certaines plantes devenues rares et en danger d'extinction. L'objectif de cette étude est la réintroduction de celles-ci dans de nouveaux habitats propices à leur développement.

L'agriculture est un secteur où les RCSF sont de plus en plus utilisés [Chanet 2007]. Les capteurs collectent des données sur les cultures et la qualité du sol et les transmettent à une station centrale présente dans la ferme. Si la station est connectée à Internet, il est possible, sous couvert d'une autorisation, d'accéder aux informations sur l'ensemble de l'exploitation.

La prévention des risques d'incendie ou d'inondation fait partie des domaines où les RCSF apportent les plus grandes perspectives. Dans ce type d'applications, les capteurs sans fil sont en charge de la détection de tous phénomènes anormaux observés dans leur périmètre d'action. Il peut s'agir d'une brusque augmentation de la température ou du taux d'humidité caractéristique d'un début d'incendie ou d'inondation. Chaque capteur est soit placé à un endroit connu, soit muni d'un système de localisation de type GPS [Kara 2007]. Ainsi, à l'autre bout du réseau, la station centrale peut fournir des renseignements précis permettant une intervention rapide et localisée (voir Figure 1.6).

Dans la surveillance des risques d'inondation, les capteurs sont obligatoirement étanches. Pour les incendies, ils doivent être assez robustes pour supporter pendant un temps suffisant de très fortes températures. Le capteur doit pouvoir détecter le départ du feu et transmettre le plus de données possibles avant sa détérioration.

Les RCSF aident également à l'étude de phénomènes complexes tels que les tremblements de terre, les éruptions volcaniques, les ouragans et les tsunamis. Ils fournissent des données permettant d'établir des modèles de prévision. Pour quelques applications présentées dans cette section, des systèmes filaires existaient déjà mais étaient plus difficiles à déployer et n'offraient parfois pas autant de fonctionnalités.

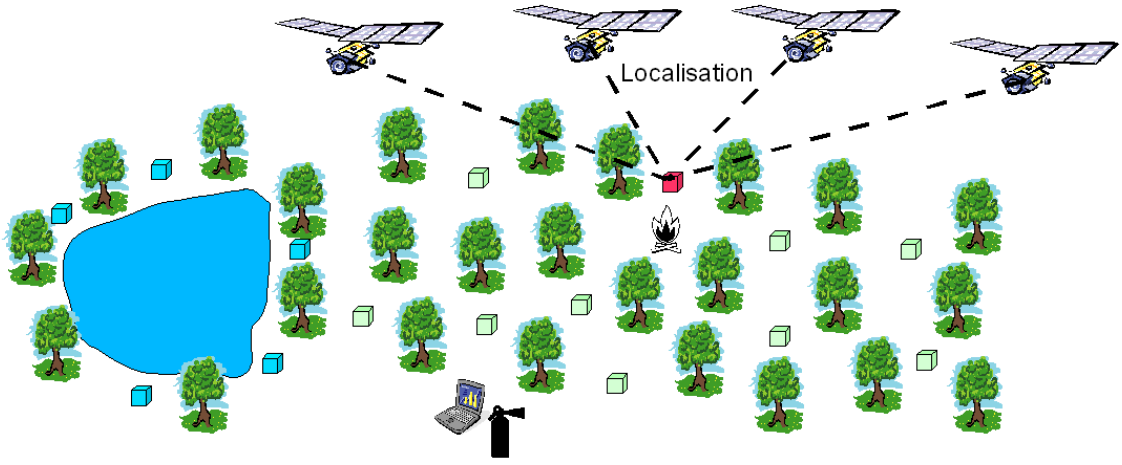

Figure 1.6 – Surveillance d'incendie et d'inondation

1.3.3 Les applications de la vie quotidienne

Les applications liées à la santé représentent une part importante des travaux de recherche sur les réseaux de capteurs, à tel point qu'un terme anglo-saxon existe pour les caractériser : on parle de « smart care, smart home ». Leur but est d'offrir un ensemble de services de surveillance à domicile.

Le projet STAR (Système Télé-Assistant Réparti) [de Vaulx 2003] [Zhou 2004a], par exemple, consistait à concevoir une plateforme dédiée à la surveillance de personnes souffrant d'arythmies cardiaques. Ce système avait aussi un rôle préventif. En effet, les arythmies cardiaques sont des phénomènes difficiles à diagnostiquer de part leur nature intermittente et très aléatoire. Généralement, une personne qui souffre potentiellement de cette maladie se voit équiper d'un dispositif (Holter) effectuant un relevé de ses signaux électrocardiogrammes (ECG) pendant une durée variant de 24 à 48 heures. Si aucun trouble n'est observé mais que des symptômes persistent, elle est hospitalisée durant une courte période de manière à être soumise à un bilan complet. Même à la suite de tous ces examens, il est possible d'avoir des doutes sur le diagnostic.

Dans le cadre du projet STAR, la personne est équipée d'un capteur sans fil intelligent capable d'acquérir et d'analyser les signaux ECG en temps réel. Mais dans ce cas, contrairement aux Holters classiques, la durée d'observation est plus longue (une voire deux semaines) et surtout la personne peut être surveillée de chez elle. En effet, le capteur utilise un module de transmission sans fil Bluetooth qui lui permet de communiquer avec une passerelle présente au domicile du patient et connectée à Internet. Cette dernière relaie ensuite les données collectées par le capteur vers un centre de traitement et d'intervention situé au sein d'un hôpital (voir Figure 1.7). Le système peut ainsi être utilisé soit pour établir un diagnostic, soit pour prévenir les risques de mort subite liés aux arythmies cardiaques.

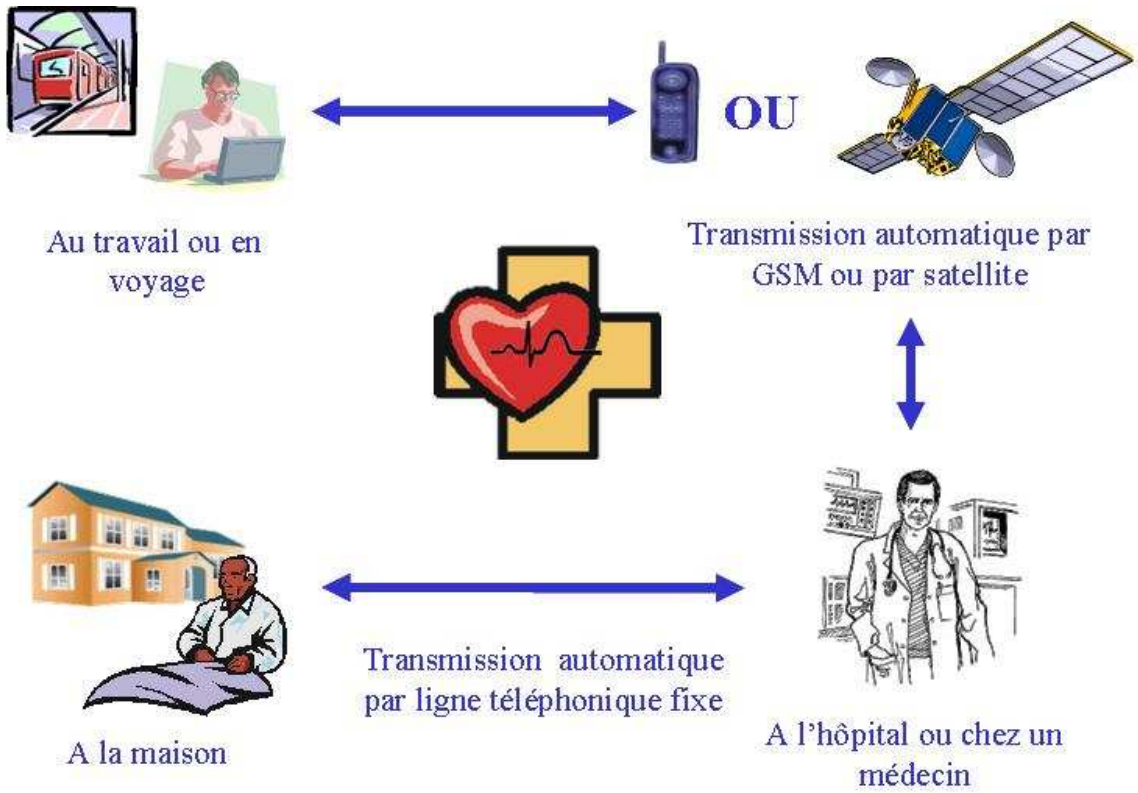

Figure 1.7 – Principe de fonctionnement du projet STAR [de Vaulx 2003]

La première phase du projet STAR se limitait à une surveillance à domicile des patients. La seconde devait permettre à la personne de vivre une vie presque normale en lui autorisant les déplacements hors de chez elle. Pour ce faire, le capteur aurait été équipé d'un module GSM pour la transmission de données et d'un GPS pour obtenir la position de la personne en cas de problème cardiaque sévère.

Dans un autre type d'application totalement différent, les capteurs munis de GPS pourraient être utilisés pour retrouver plus rapidement des animaux domestiques égarés. Le principal frein au développement de ce type d'applications est la taille des capteurs qui reste encore trop importante.

La prévention des risques de chute chez les personnes âgées est un autre exemple d'application. Ces chutes sont dans la plupart du temps causes de fractures (fémur, bassin,…) qui s'accompagnent de longues semaines d'inactivité ou dans le pire des cas d'invalidités partielles permanentes. Cette perte d'autonomie oblige la personne à faire appel à une aide soignante à domicile ou à intégrer un centre spécialisé.

Dans une première étape, Les capteurs sans fil déterminent le « degré d'équilibre » de référence de la personne. Il servira d'étalon pour les mesures effectuées durant les jours suivants afin d'évaluer les risques de chute et d'en avertir qui de droit [Hewson 2007].

Pour la surveillance d'un lieu, l'utilisation d'un RCSF a quelques avantages par rapport à un système de sécurité classique. Le premier est une vitesse de déploiement plus rapide, les travaux en infrastructure n'étant pas nécessaires (câblage d'alimentation ou réseaux,…). Le second est une modularité ou une extensibilité sans limite. Des capteurs peuvent être ajoutés, supprimés, déplacés, selon les besoins et en un temps assez bref. Pour un endroit nécessitant une sécurité ponctuelle de bas niveau, l'emploi d'un RCSF s'avérerait une solution idéale. Toutefois, avant d'envisager une exploitation massive, des problèmes de sécurité internes aux réseaux de capteurs doivent être résolus.

1.3.4 Les applications de véhicules intelligents

Les applications présentées dans cette section pourraient très bien être réparties entre celles qui ont trait à l'environnement ou à la vie quotidienne. Le terme « véhicule » est général et englobe les voitures de tourisme, les engins agricoles, les bateaux.

Depuis quelques années, la sécurité routière est devenue un enjeu important au niveau politique et économique. Les mesures politiques récentes avec l'installation de radars automatiques ont pour objectif de diminuer le nombre de blessés et de morts sur la route. A cela viennent s'ajouter des données plus économiques se rapportant aux coûts engendrés par les nombreux accidents même sans victime. De nombreux travaux de recherche visent à diminuer le nombre d'accidents soit en proposant des dispositifs d'aide à la conduite soit en concevant des systèmes d'alerte et d'anticipation [Chong 2003]. Il s'agit d'un thème clairement identifié au sein de la fédération TIMS (Technologies de l'Information, de la Mobilité et de la Sûreté) du pôle de recherche clermontois.

En équipant en masse les voitures de capteurs sans fil, les conducteurs des voitures à proximité d'un accident seraient avertis et seraient automatiquement redirigés vers un itinéraire de déviation. Cela aurait pour conséquence positive de diminuer le nombre de voitures et donc de personnes impliquées. De plus, les véhicules accidentés pourraient émettre un appel vers le poste de secouristes le plus proche en indiquant l'état de santé de ses passagers. Cette action résulterait en une intervention rapide et bien ciblée des secours (voir Figure 1.8).

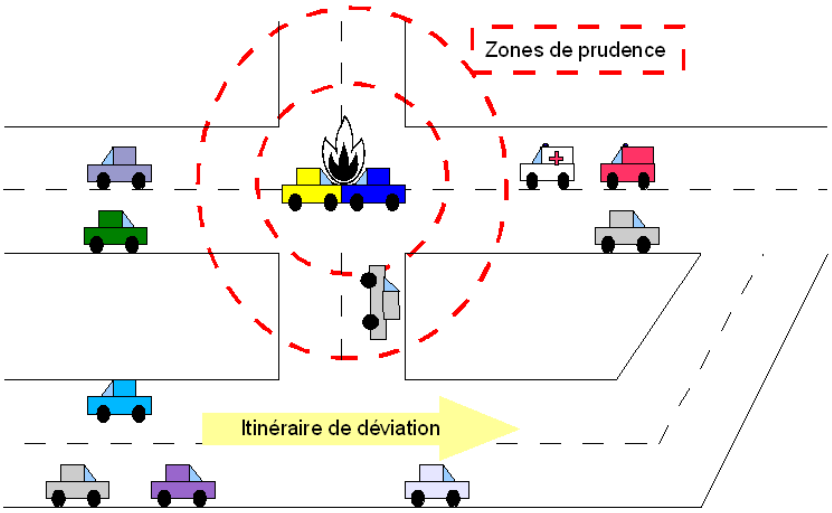

Figure 1.8 – Réseau de capteurs sans fil dédiés à la sécurité routière

A titre préventif, les conducteurs pourraient être alertés d'un ralentissement impromptu du trafic. Actuellement, les systèmes de navigation par GPS peuvent servir à réguler le trafic en proposant de nouveaux itinéraires au regard de la circulation. Ce dispositif pourrait être intégré dans des capteurs évolués offrant les autres fonctionnalités qui viennent d'être citées. Toutes les informations recueillies par tous les capteurs des voitures pourraient être traitées anonymement dans un centre en charge de la gestion des feux tricolores. La durée de ceux-ci serait modifiée en temps réel afin de résorber dès leur naissance un embouteillage ou un ralentissement.

Les capteurs peuvent être reliés à des actionneurs leur permettant d'être en lien direct avec le véhicule sans passer par le conducteur. Associé à une batterie de détecteurs et/ou de caméras, le capteur est capable d'éviter un obstacle, soit en appliquant un freinage d'urgence, soit en conduisant le véhicule. Ce type d'applications présente cependant des problèmes d'ordre techniques et juridiques. D'une part, certains algorithmes dédiés au traitement et à l'analyse d'images provenant des caméras nécessitent une puissance de calcul non présente dans les capteurs actuels. D'autre part, le fait qu'un véhicule puisse être conduit par un ordinateur pose le problème juridique de connaître le responsable en cas d'accident. Enfin, avant un déploiement à grande échelle, la fiabilité du système devra être prouvée pour dépasser les appréhensions que pourraient avoir, à juste titre, le grand public.

Des capteurs qui recevraient des données provenant d'un panneau de limitations de vitesse pourraient, par l'intermédiaire du régulateur de vitesse, adapter l'allure du véhicule. Par exemple, à l'entrée des villes, la voiture ralentirait automatiquement pour descendre à une vitesse de 50Km/h.

La plateforme PAVIN (Plate-forme d'Auvergne pour les Véhicules INtelligents) de la fédération TIMS a pour but de prolonger le réseau de transport urbain de la ville de Clermont-Ferrand. Des véhicules au gabarit réduit de type voiture de golf appelés Cycab sont mis à disposition des citoyens pour les transporter. Une personne se présente à une borne d'appel de Cycab, indique sa destination, monte dans le véhicule et est conduit automatiquement à l'endroit voulu. Ces véhicules sont même capables de circuler en convoi sans être physiquement relié. Les capteurs sans fil interviennent pour la communication entre les différents éléments de la plate-forme (véhicules, bornes,…)

Les notions de convoi et de guidage automatique se retrouvent dans les recherches sur les engins agricoles. Contrairement aux véhicules urbains, les engins agricoles doivent faire face à une circulation sur des voies glissantes et irrégulières. Equiper ces engins de capteurs permet de leur envoyer des ordres mais aussi d'obtenir une multitude de données. Ces informations concernent la trajectoire du véhicule, le travail en cours d'exécution et même des indications sur l'exploitation (qualité et humidité du sol, température,…).

Le projet SIGEMO [Soulignac 2006] du Cemagref a abouti à la réalisation d'un système d'information pour la gestion de l'épandage. L'objectif est de connaître la nature et la quantité de produit épandu dans une exploitation agricole. Les réseaux de capteurs pourraient permettre l'automatisation de la collecte d'informations pour alimenter le système d'information. Les données seraient transmises, en temps réel, par un capteur situé sur la machine agricole en charge de l'épandage.

D'autres applications plus axées sur les loisirs et les informations pratiques sont envisageables. En utilisant les capteurs comme passerelles réseau, les passagers autres que le conducteur pourraient avoir accès à Internet et poursuivre leur activité professionnelle ou se divertir grâce aux nombreux services offerts. Les touristes pourraient également recevoir des messages provenant de l'office du tourisme de la ville traversée. Des renseignements provenant des restaurants, des commerces et autres enseignes rencontrés sur le trajet pourraient être également reçus par le même procédé.

1.3.5 Les applications industrielles

Les RCSF peuvent donner des indicateurs sur l'état d'une machine ou sur le fonctionnement de l'ensemble d'une chaîne de production [Akyildiz 2002].
Grâce au mode de transmission sans fil, un capteur peut fournir des données à partir d'emplacements inaccessibles par d'autres moyens. La tendance actuelle de miniaturisation des capteurs ouvre de nombreuses perspectives comme celle d'associer un capteur à une pièce pour obtenir son niveau d'usure au cours du temps.

En plaçant des capteurs tout au long d'une chaîne de production, on peut imaginer les informations qui pourraient être consultables en temps réel. Ces données pourraient concerner la quantité et la qualité de la production à différents points de la chaîne, indiquer la cause des retards (pannes, reconfiguration de machine, etc.).

Dans les applications industrielles, on trouve également le marquage RFID (Radio Frequency Identification). Le tag RFID n'est pas en lui-même un capteur mais en combinant ces deux technologies, il est possible d'obtenir des applications intéressantes. On peut imaginer un système dans lequel des produits munis de tags RFID transiteraient dans un environnement surveillé par des capteurs. A la lecture des tags, les capteurs effectueraient des analyses, transmettraient des informations et exécuteraient des actions.

Le projet MobiPlus [Zhou 2006a] utilise ce principe de fonctionnement. Il concerne plus les applications de la vie quotidienne que celles industrielles, mais illustre bien le couplage technologies sans fil et RFID. Le but de ce projet est l'amélioration de l'accès au réseau de transport urbain des personnes handicapées. Chaque personne est munie d'un tag RFID anonyme indiquant le type d'handicap dont elle souffre. Les stations de bus classiques sont équipées d'une borne abritant un capteur sans fil connecté à un lecteur RFID. Dès qu'une personne passe à proximité d'une borne, son tag RFID est détecté et les informations contenues sont envoyées en direction du bus en approche. A l'intérieur du bus, un système avertit de manière sonore le conducteur de la présence d'une personne handicapée. Au niveau de l'arrêt, le conducteur amorce une manœuvre facilitant l'accès du bus à la personne handicapée. Ainsi, un usager en fauteuil roulant (UFR) accède par la porte centrale du bus où

est installée une palette d'accès au déploiement automatisé. Une description en détails de ce projet est donnée dans le chapitre 5.

1.3.6 Une autre classification des applications

La classification par domaines d'application démontre le rôle important que peuvent ou pourraient jouer les RCSF. Si l'on considère les applications présentées, on peut définir deux catégories :

- Les applications d'acquisition de données ;
- Les applications en remplacement d'une infrastructure de réseau filaire.

La première catégorie contient l'ensemble des applications dont le but est la collecte d'informations. Ces informations proviennent de l'environnement, d'un champ de bataille, d'une machine,… Ces applications peuvent se découper en 3 phases :

- La phase d'acquisition ;
- La phase de stockage ;
- La phase de transmission.

La phase d'acquisition est réalisée par les dispositifs qui mesurent les grandeurs physiques observées. Les phases de stockage et de transmission sont de la responsabilité du capteur sans fil. Le stockage peut être requis par l'application ou peut être nécessaire en cas d'indisponibilité du lien de communication sans fil.

Les RCSF peuvent venir en complément ou remplacement des réseaux filaires. Suite à une catastrophe naturelle comme un tremblement de terre où l'ensemble des infrastructures filaires ont été gravement endommagées, les secouristes pourraient utiliser les RCSF pour communiquer entre eux et mieux s'organiser. Dans un musée ou un monument historique où l'on souhaiterait disposer de commodités tel qu'Internet, l'utilisation des capteurs sans fil aurait l'avantage de ne pas nécessiter de lourds travaux d'infrastructure demandant des autorisations parfois difficiles à obtenir.

1.4 Les problématiques liées aux réseaux de capteurs

Derrière les RCSF se cachent une multitude de problématiques qui sont exposées dans cette section. Les travaux et pistes de recherche dans ce domaine viennent en réponse aux nombreux problèmes relatifs aux réseaux de capteurs. Un nombre important d'entre eux sont issus des réseaux sans fil Ad Hoc soit directement tels qu'ils sont, soit de manière amplifiée. A cela, des problèmes propres uniquement aux RCSF se rajoutent.

1.4.1 Les problématiques issues des réseaux sans fil Ad Hoc

Les RCSF sont, la plupart du temps, des réseaux sans fil Ad Hoc. Chaque capteur joue le rôle à la fois de routeur et de client dans le réseau. Les grands thèmes de recherche dans les réseaux sans fil Ad Hoc sont principalement le routage, la localisation, la synchronisation et la qualité de service (QdS).

1.4.1.1 Le routage

Dans un réseau Ad Hoc, l'isolement d'un nœud ou la partition (ou séparation) du réseau sont des évènements courants. La topologie d'un réseau Ad Hoc ne cesse d'évoluer et de se modifier au rythme du déplacement des nœuds qui le composent. Dans les RCSF, ce phénomène est accentué par la fragilité des capteurs cause de nombreuses ruptures de liens de communication. Par conséquent, les protocoles de routage Ad Hoc doivent être modifiés et adaptés à ce type de réseau s'appuyant sur des dispositifs à faibles ressources. Une mauvaise gestion ou politique de routage à un instant donné peut provoquer la panne d'un ensemble de capteurs par épuisement d'énergie. Un compromis doit être trouvé entre minimiser le nombre de capteurs hors service et ne pas faire peser la totalité des traitements sur les nœuds les plus valides. L'équilibrage de charge doit être mis en place dès le départ. Les protocoles de routage dédiés aux RCSF seront présentés plus en détail dans le chapitre 4. Relativement peu d'entre eux font l'objet d'évaluation en situation réelle, le contre-exemple étant OLSR [Plesse 2005].

Le routage est toutefois simplifié si l'application ne nécessite que des capteurs sans fil statiques avec une position bien établie au préalable. Le protocole de routage peut ainsi se concentrer sur les problèmes de changement de routes ou de répartition de la charge de transmission de chaque capteur de manière à répondre aux problèmes de consommation d'énergie. Par conséquent, le problème d'exploration du réseau est atténué.

En effet, dans un réseau filaire ou sans fil statique, ce problème n'existe qu'au démarrage, quand il faut déterminer les nœuds voisins c'est-à-dire les nœuds à un saut de communication, les passerelles ou un chemin vers la station de traitement distante. Connaissant la position des capteurs à l'avance, une pré-configuration est possible afin de résoudre totalement ce problème. Si la mobilité des nœuds est requise, la phase d'exploration est complexe avec l'établissement de la liste des nœuds voisins et la construction de chemins vers certains nœuds en temps réel et pour une durée variable.

Cette phase d'exploration peut devenir encore plus complexe si on lui adjoint une nécessité de former des groupes ou des clusters nécessaires au routage et à l'acquisition des données. Ce mode de fonctionnement en groupe est de plus en plus associé à la notion d'auto-configuration qui représente un thème de recherche très répandu [Heinzelman 2002] [Diao 2007a]. Les capteurs s'organisent par eux-mêmes à l'aide de règles fournies au préalable. Ces dernières peuvent être basées sur la distance en nombre de sauts ou en mètre, le nombre de voisins.

1.4.1.2 La localisation

La localisation est un élément important que l'on retrouve dans l'auto-configuration mais également dans le routage proprement dit. Pour améliorer les performances des protocoles de routage, la localisation est employée pour orienter le flux des messages et éviter une inondation systématique du réseau.

Les protocoles de routage dits « géographiques » sont bien adaptés aux réseaux de faible voire moyenne mobilité. Leurs performances reposent sur la fiabilité des informations géographiques dont disposent les nœuds. Dans un réseau où les nœuds sont très mobiles, le routage géographique risque de ne pas aboutir dans la plupart des cas. La cause est le déplacement totalement aléatoire des nœuds pouvant partir à l'opposé de leur position initialement connue ou au comportement global du réseau où de multiples partitions (ou séparation) se font et se défont. Quand le routage géographique est un échec, le protocole de routage de type « inondation » (« flooding) est utilisé. Dans ce protocole, chaque nœud qui reçoit un message, le transmet à l'ensemble de ces nœuds voisins et ainsi de suite. Tous les nœuds, sauf ceux isolés, reçoivent le message.

Si l'on se focalise sur les RCSF, la nécessité de localisation est mise en évidence par quelques applications présentées dans la section précédente. Quand un événement se produit comme un incendie, il faut déterminer rapidement où celui-ci a eu lieu. Pour obtenir cette localisation, le GPS (Global Positioning System) est naturellement le premier dispositif auquel on pense. Toutefois, ses coûts d'achat et en terme d'énergie consommée est un frein à un usage massif. De plus, dans un environnement urbain, quand son utilisation est possible, l'imprécision observée peut s'avérer importante.

D'autres pistes de recherche sont explorées pour répondre à cette problématique. L'une d'entre elle est basée sur le repérage grâce à la puissance du signal émis par le module de communication sans fil [Le Borgne 2007]. Un nœud est ainsi repéré par triangulation du signal émis par ses voisins. Dans une architecture de localisation hybride, les deux méthodes que sont le GPS et la puissance du signal peuvent être combinées. Les nœuds se repèrent, à l'aide de la puissance du signal, par rapport à des éléments géo-référencés du réseau. Ces derniers sont soit des points fixes dont les positions sont connues précisément, soit des capteurs équipés de GPS (voir Figure 1.9).

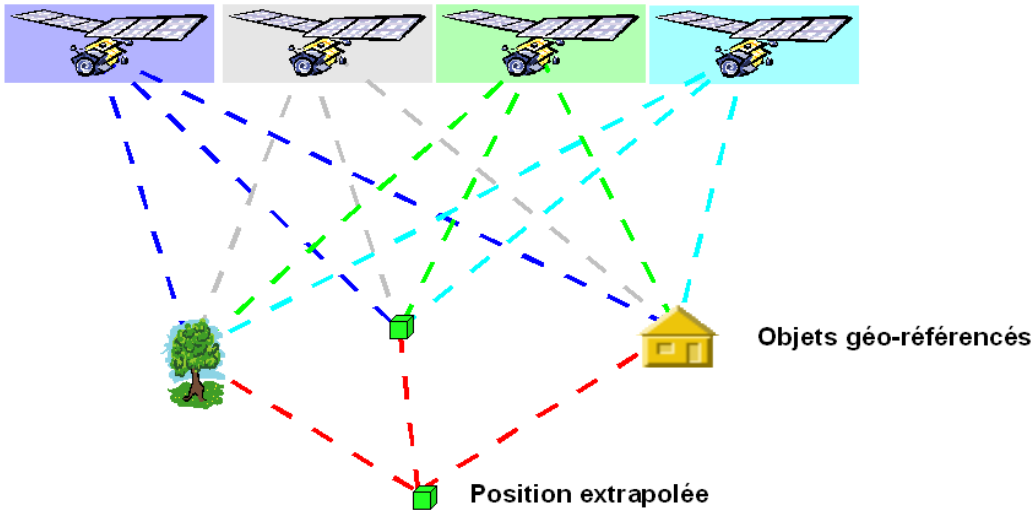

Figure 1.9 – Architecture de localisation hybride

1.4.1.3 La Qualité de Service (QdS)

Dans les RCSF, la notion de qualité de service s'applique à la communication sans fil comme dans les réseaux sans fil Ad Hoc mais également aux contraintes spécifiées par l'application. Dans les réseaux sans fil Ad Hoc, la QdS intègre tous les aspects liés à la transmission de données. Elle s'appuie sur les critères d'évaluation que sont la bande passante disponible, la latence ou le délai de livraison des messages. Dans ce cas, la QdS correspond à une gestion efficace des ressources du réseau pour répondre aux besoins de l'application. L'élément central est le réseau sur lequel l'application s'appuie et lui fait part de ces besoins au niveau de la transmission de données.

Dans un contexte plus général, la QdS est un indicateur permettant de définir un cadre d'utilisation d'un réseau. Pour ce faire, on peut se restreindre à la simple communication ou l'étendre à toute l'application à laquelle le réseau est dédié. Dans les RCSF, en plus des contraintes d'acheminement et de vitesse de propagation des données, il faut considérer la qualité et la pertinence de celles-ci. En d'autres termes, les données reçues au niveau de la station centrale de collecte doivent être acheminées dans un temps limité et doivent être identiques à celles prélevées au niveau du capteur. Au-delà des critères de performances déjà cités, la QdS se mesure suivant le déploiement et la couverture du réseau, le nombre de nœuds actifs à un instant donné.

Pour différencier ces deux définitions l'une de l'autre, le terme de « niveau d'exigence » pourrait être utilisé à la place de qualité de service quand on parle d'une application. Enfin, il paraît raisonnable de penser que la qualité de service du réseau soit intégrée dans celle de l'application.

1.4.1.4 La synchronisation temporelle

En général, les temps d'aller-retour ou RTT (Round-Trip Time) sont souvent utilisés dans les méthodes d'estimation de la bande passante disponible [Jain 2003] [Amamra 2008] et donc pour évaluer la qualité de service. Une autre technique d'estimation consiste à considérer les temps d'aller simple OTT (One-way Transit Time) plutôt que les RTT. Celle-ci est moins intrusive en termes de trafic réseau généré mais nécessite que les nœuds soient synchronisés temporellement.

La synchronisation temporelle est un élément important pour une grande partie des applications de RCSF. La plupart des applications d'acquisition de données requièrent un marquage temporel pour indiquer la date de collecte. Des décalages au niveau de l'heure de certains capteurs impliquent un travail fastidieux de synchronisation post-détection pour avoir une chronologie des évènements correcte. Dans le pire des cas, des messages peuvent être considérés comme obsolètes et égarés dans le réseau alors qu'ils transportent des données récentes. La conséquence est leur suppression par un capteur présent sur le chemin qui les mène vers la station centrale de collecte et d'archivage des données.

Grâce à un temps d'horloge identique, les capteurs peuvent établir entre eux des « rendez-vous » pour communiquer. Pour économiser de l'énergie, les capteurs alternent entre phase active et phase de veille. Le premier élément mis en veille est le module de communication sans fil très consommateur d'énergie. Sans une parfaite synchronisation temporelle, ce mode de fonctionnement ne serait pas possible car certains capteurs tenteraient de communiquer avec d'autres endormis.

Cette synchronisation peut être obtenue à l'aide du GPS. Toutefois, comme pour la localisation, le coût d'un tel système aussi bien d'ordre financier qu'énergétique n'est pas

anodin. Par conséquent, différents protocoles ont été développés pour répondre à cette problématique. Certains sont des adaptations de protocoles existants dans les réseaux filaires. Cette transition n'est pas toujours possible comme l'illustre la complexité voire l'impossibilité d'utiliser la protocole (S)NTP ((Simple) Network Time Protocol) car nécessitant une puissance de calcul importante [Stojmenovic 2005].

1.4.2 Les problématiques propres aux réseaux de capteurs sans fil

Dans la section précédente, les problématiques communes aux réseaux de capteurs sans fil et Ad Hoc ont été présentées en apportant des précisions sur les différences existantes. Maintenant, nous allons nous intéresser aux problématiques propres aux RCSF. Certaines d'entre elles ont déjà été introduites implicitement dans le paragraphe 1.3.

1.4.2.1 La gestion des ressources

Le point crucial à considérer dans les RCSF est la limitation des ressources que sont l'énergie disponible, la mémoire et la puissance de calcul. Un capteur sans fil intègre des technologies qui sont, a priori, maîtrisées dans les domaines allant de l'équipement informatique grand public aux systèmes embarqués. Pour les RCSF, la complexité se situe au niveau des restrictions de ressources à considérer dans, par exemple, l'implémentation d'un algorithme de cryptographie ou de contrôle des erreurs dans un tel capteur.

Au niveau de l'énergie, le module de communication sans fil est un des éléments les plus gourmands en énergie. Les premières transmissions par ondes radio datent de la fin du XIX^{e} siècle mais l'adaptation et la diffusion à grande échelle de ce mode de communication aux réseaux d'ordinateurs sont assez récentes. Des modules de communication dédiés aux RCSF sont en cours de développement comme en attestent les travaux actuels sur les technologies UWB [Nekoogar 2004]. L'objectif est donc d'avoir un dispositif peu consommateur en énergie et de faible coût.

Dans la conception d'un RCSF, l'une des premières questions à se poser est celle du type d'alimentation à utiliser. En reprenant les idées déjà présentées dans le paragraphe 1.1, pour l'alimentation, on va très rapidement s'orienter vers des sources d'énergie quasi illimitées fournies par la nature comme l'énergie solaire ou éolienne. Mais la plupart du temps, ce genre d'énergie n'est pas disponible ou mal adapté. L'exemple des panneaux solaires a déjà été cité. Leur utilisation en environnement chaud et désertique paraît logique et intéressante. Malheureusement, ceux-ci ne peuvent pas être utilisés en plein milieu d'un désert saharien car ils sont soit enterrés par les tempêtes de sable et n'ont donc plus accès au soleil, soit détériorés par les grains de sable qui rayent leur surface.

Le problème du dimensionnement de l'alimentation c'est-à-dire de l'autonomie du capteur vient s'additionner à tout ceci. Le capteur est conçu le plus petit possible mais pour le satisfaire énergétiquement on lui ajoute un panneau solaire ou une éolienne, dispositif assez volumineux et encombrant. La solution la plus répandue actuellement est une alimentation par piles standards (en général pile de type bouton ou AA etc.).

Enfin, les pannes énergétiques qui peuvent potentiellement affecter certains capteurs influent sur la topologie et l'organisation du réseau et rendent le routage encore plus complexe.

L'énergie, la mémoire et la puissance de calcul sont des ressources liées entre elles. Pour limiter la transmission de données qui consomme énormément d'énergie, deux solutions sont possibles. La première est le stockage des données au niveau du capteur associé à une transmission de celles-ci à la demande plutôt que de manière systématique. En effet, la consommation d'énergie durant une communication sans fil est proportionnelle au nombre

d'octets transmis. La seconde solution est l'agrégation de données et repose également sur ce principe. Cette méthode consiste à effectuer des traitements basés sur des fonctions mathématiques (somme/différence, moyenne, maximum/minimum, filtrage …) ou des algorithmes au niveau de certains nœuds pour diminuer la quantité de données transmises à chaque fois. Des explications plus détaillées et un exemple d'application seront fournis dans le paragraphe suivant. Que ce soit l'une ou l'autre de ces deux solutions, la consommation d'énergie ne devient pas nulle, elle est moindre par rapport à un mode basé uniquement sur la transmission. Simplement, l'énergie consommée diminue au détriment soit de la mémoire, soit de la puissance utilisée (voir Figure 1.10).

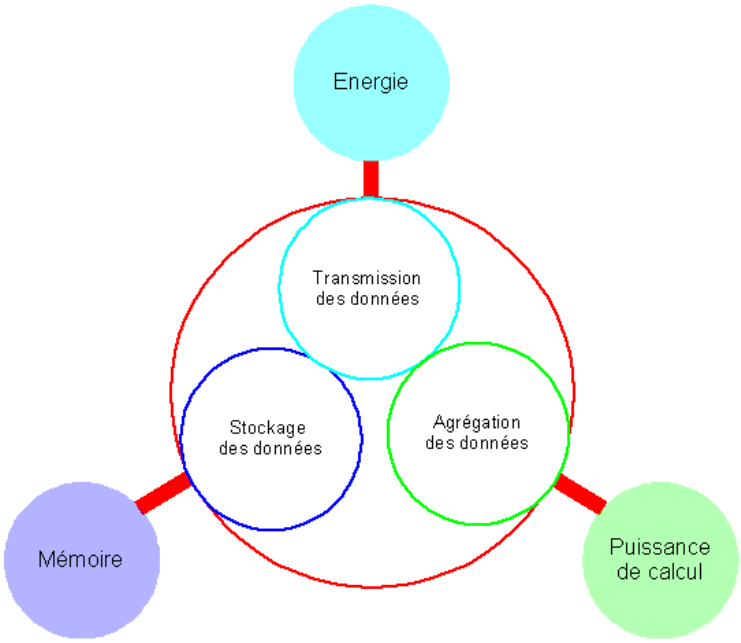

Figure 1.10 – Répartition de la consommation des ressources d'un capteur sans fil

La gestion des ressources intègre également le passage du capteur en mode veille. L'ensemble des éléments du capteur ou uniquement le module de communication peuvent être concernés. A titre indicatif, un dispositif de communication de type ZigBee a une consommation inférieure à 1µA voire 2µA en mode veille profond [XBee 2007].

1.4.2.2 La gestion des données collectées

La gestion des données collectées rassemble l'ensemble des traitements subis par les données durant leur cycle de vie au sein du RCSF (voir Figure 1.11). Les phases d'acquisition et de présentation des données à l'utilisateur viennent s'ajouter aux actions présentées dans la Figure 1.10.

L'acquisition à proprement parler est déportée sur le dispositif de mesure de la grandeur physique. Le capteur sans fil est en charge de la bonne récupération des données à la fréquence d'acquisition souhaitée. La réalisation de cette tâche constitue le fonctionnement de base du capteur sans fil auxquelles viennent s'ajouter d'autres, plus ou moins complexes, selon l'application supportée.

La présentation des données se résume à l'envoi des données vers une station centrale de collecte qui affiche les données de manière brute ou sous un format spécifié par l'utilisateur. Dans une version plus élaborée, l'utilisateur peut envoyer des requêtes d'interrogation au RCSF. Il récupère ainsi seulement les données dont il a besoin.

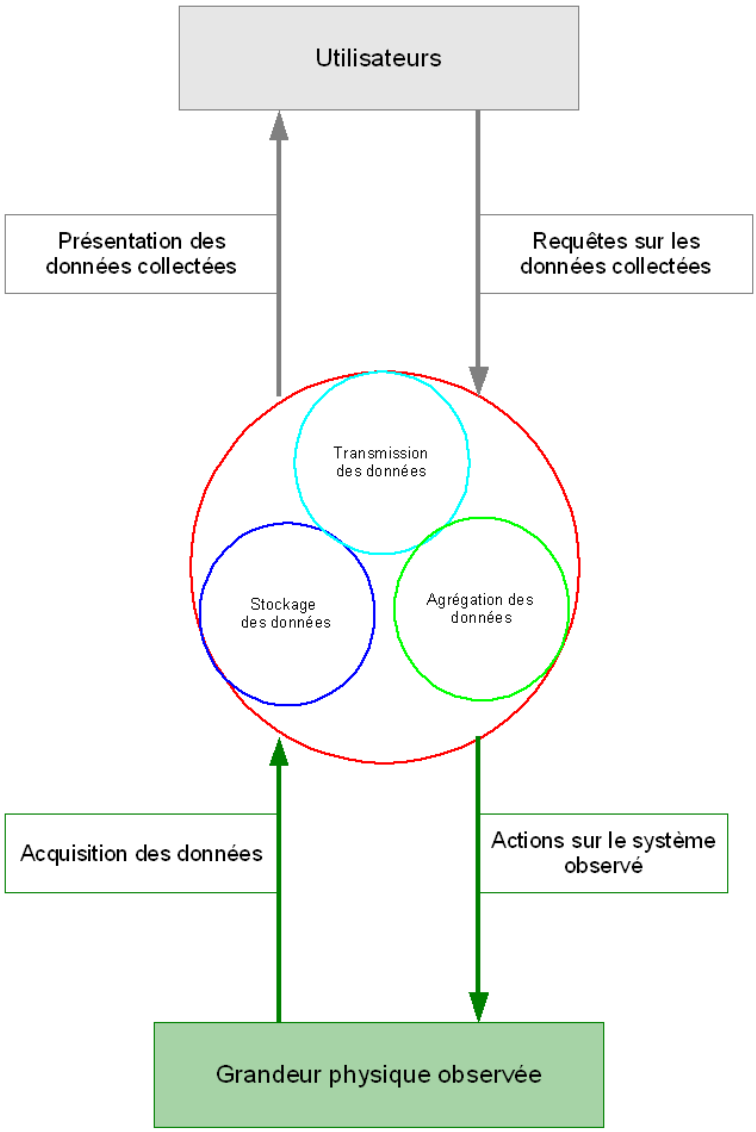

Figure 1.11 – Cycle d'acquisition des données

Le stockage de données plutôt que la transmission ne répond pas seulement à une politique d'économie d'énergie. Si les ressources disponibles sont importantes comparativement aux traitements à effectuer, la solution la plus simple et la plus rapide à mettre en place est une acquisition suivie d'une transmission immédiate des données. Malheureusement, les médias de communication sans fil sont moins fiables que leurs homologues filaires et pâtissent des interférences, des collisions, des multi-trajets causés par le relief montagneux et les bâtiments qui pourraient être situés aux alentours. Ce phénomène est d'autant plus accentué que le déplacement des capteurs sans fil est rapide. En outre, la qualité de service dans les réseaux sans fil est plus complexe à établir et subit d'importantes fluctuations. Par conséquent, durant l'indisponibilité de la communication sans fil, les données devront être stockées. L'emploi d'une mémoire non volatile pourrait être préconisé pour le stockage de données importantes car elles seraient toujours disponibles même en cas de panne d'énergie du capteur.

Une application de détermination de la température moyenne d'une zone donnée illustre parfaitement l'intérêt de l'agrégation de données. Dans un mode de fonctionnement

simple, tous les capteurs transmettent à la station centrale de collecte la valeur de température obtenue à l'endroit où ils étaient placés. En intégrant l'agrégation de données, un arbre à différents niveaux est constitué. Les nœuds feuilles de cet arbre sont regroupés en sous-ensemble de capteurs formant des sous-zones ou clusters de premier niveau. Un nœud maître de niveau supérieur est désigné pour chacune de ces sous-zones. Son rôle est de réaliser la première agrégation et de transmettre ensuite ce résultat à son propre nœud maître de niveau supérieur. Ce fonctionnement est répété jusqu'à ce que la station centrale, racine de l'arbre soit atteinte (voir Figure 1.12). Le nombre de messages échangés au sein du réseau s'en trouve ainsi diminué. De plus, les risques de saturation de la station centrale sont limités, le nombre de données à traiter étant très réduit. Le point critique de ce type d'application est l'élection des nœuds maîtres de zone.

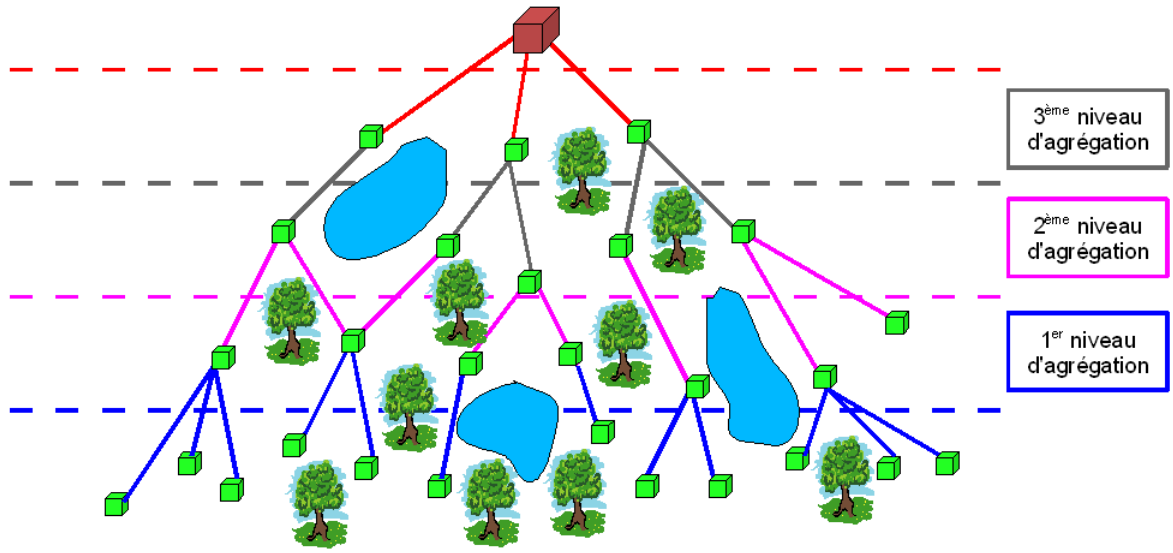

Figure 1.12 – Arbre d'agrégation de données

1.4.2.3 La mise à l'échelle

Au niveau des réseaux sans fil Ad Hoc, le problème de mise à l'échelle ou d'extensibilité existe bien mais celui-ci est plus important dans les RCSF et le sera encore plus dans les années à venir. Les capteurs sans fil du futur sont destinés à avoir des dimensions microscopiques permettant, par exemple, de les disperser sur la zone à étudier à partir d'un hélicoptère. Des milliers voire des dizaines de milliers de capteurs pourront ainsi être disséminés dans un périmètre restreint.

La capacité de mise à l'échelle d'un réseau est son aptitude à être suffisamment flexible pour répondre aux variations du nombre de nœuds qui le composent. Logiquement, l'organisation structurelle et le protocole de routage d'un réseau sont évalués en augmentant progressivement le nombre de nœuds. A partir des résultats obtenus, on établit le profil du réseau dans lequel est présente sa capacité de mise à l'échelle. Toutefois, actuellement, que ce soit en simulation ou encore plus en réalité, les tests sur un grand nombre de nœuds sont difficiles voire impossibles à réaliser [NS 2008].

La dispersion à la volée d'un nombre important de capteurs sans fil dans un espace restreint crée des zones d'interférences ou de collisions de communication. Dans tout réseau sans fil, les problèmes de station cachée et exposée existent et sont amplifiés selon la quantité de nœuds au mètre carré ou densité du réseau [Tanenbaum 2003] [Pujolle 2007].

Le problème de la station cachée est illustré par la transmission au même instant d'un message de A et C vers le nœud B (voir Figure 1.13). Les deux messages vont interférer et être illisibles au niveau de B. La cause de ce problème est le rayon de portée limité de chaque nœud du réseau. Cette limitation empêche le nœud A de connaître les actions de communication menées par certains nœuds voisins de B comme C. Ce constat s'applique également pour C par rapport au nœud A. La résolution de ce problème est à la charge de la couche MAC (Medium Access Control) utilisée par l'intermédiaire, par exemple, du protocole IEEE 802.11. Ce dernier s'appuie sur le mécanisme CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance) qui consiste à avertir les nœuds voisins avant une communication à l'aide de message de type RTS (Request To Send) et CTS (Clear To Send). Les messages RTS sont issus du nœud source A et constituent des demandes d'envoi. Plus précisément, ils servent à avertir le nœud destination B de l'imminence d'une tentative de communication. Des messages CTS envoyés par B viennent ensuite en réponse aux RTS. Ils constituent d'une part une invitation à émettre au nœud A et d'autre part une indication aux autres nœuds voisins de B, dans ce cas C, à ne pas entrer en communication avec lui.

Dans le cas où le nœud A souhaite communiquer avec B et C avec D, le problème de la station exposée apparaît. Si le nœud C écoute le médium sans fil, il va détecter la communication entre A et B. Par précaution, C va attendre la fin de la communication entre A et B. Or, les interférences ou collisions subies par C provenant de B peuvent être suffisamment faibles pour ne pas interdire la communication entre C et D. Le mécanisme CSMA/CA présenté ci-dessus résout en partie ce problème. Quand C reçoit un message CTS de B, cela lui indique que B est déjà en communication avec un autre nœud en l'occurrence A. C peut ainsi considérer qu'une communication avec D est possible. De plus, l'utilisation d'antennes directionnelles permet de minimiser les interférences potentielles entre B et C.

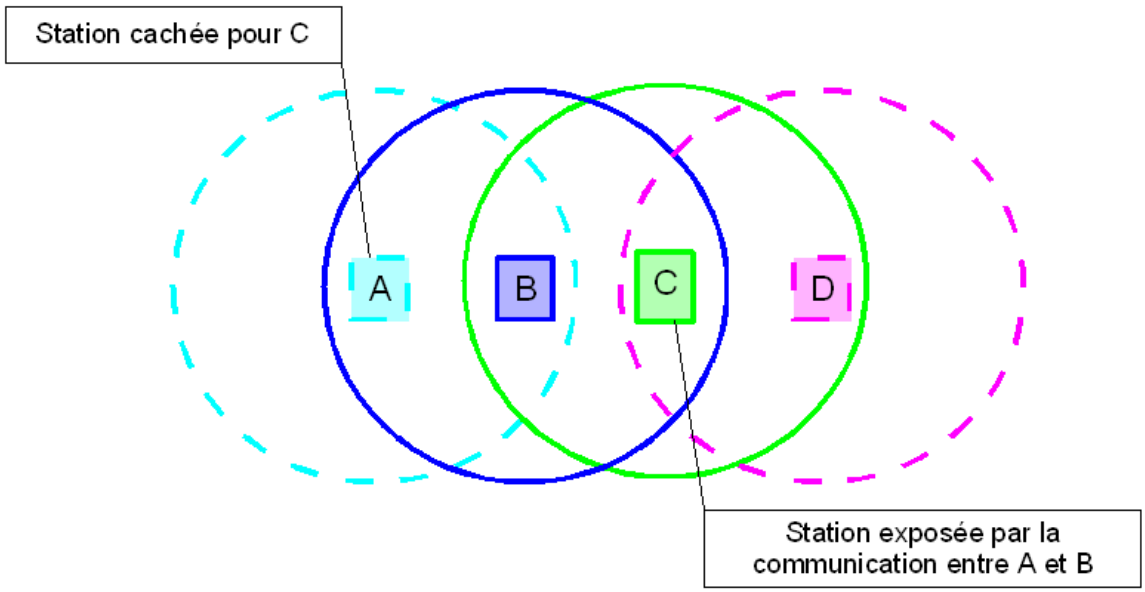

Figure 1.13 – Illustration des problèmes de station cachée et exposée

Si un RCSF fait appel à des fonctions d'agrégation de données, la mise à l'échelle implique la prise en compte du surplus de traitements à effectuer au sein de chaque capteur. Dans l'architecture de type arbre présentée dans le paragraphe 1.4.2.2, pour répondre à une mise à l'échelle, on peut soit augmenter le nombre de nœuds maîtres, soit accroître la charge de traitements de ces derniers.

1.4.2.4 L'adressage

Le problème de l'adressage est lié à celui de la mise à l'échelle. Le nombre important de capteurs sans fil déployé oblige à se poser des questions sur l'adressage à utiliser. Deux tendances ressortent au sujet de cette problématique :

- Identification unique des capteurs ;
- Identification par localisation.

La première tendance consiste à offrir à chaque capteur un identifiant unique. Quand le nombre de capteurs est limité à une cinquantaine, des solutions simples à mettre en place sont possibles. Cela peut se résumer à affecter un identifiant au capteur au moment de sa réalisation. Le numéro de série du capteur jouerait le rôle d'adresse. Si les capteurs sont équipés de module de communication IEEE 802.11, l'adresse MAC peut être utilisée pour l'adressage.

L'adressage peut être de type IPv4 avec la mise en place d'un serveur NAT (Network Address Translation) au niveau de la station centrale pour relier, si besoin, le RCSF à Internet. Dans une vision globale, chaque capteur pourrait être muni d'une adresse IPv6 « définitive ». Ce type d'adresse est codé sur 16 octets ce qui représente environ $3.4\ 10^{38}$ adresses possibles. Par conséquent, une plage de ces adresses pourrait être réservée aux capteurs sans fil.

Pour les applications de RCSF, les données sont l'aspect le plus important et pas l'identifiant du nœud qui les a collectées. La deuxième tendance est de ne pas fournir d'identification aux nœuds mais de se centrer sur les données en leur associant un repère spatial et temporel. De cette manière, on sait d'où provient la donnée ainsi que l'heure à laquelle elle a été collectée. Un capteur est ainsi identifié par rapport à sa position. Par exemple, son adresse GPS ou sa zone d'appartenance peut servir d'identifiant et donc, en toute logique, en cas de déplacement, cet identifiant changerait.

L'un des avantages de cette méthode est de simplifier la phase de déploiement du réseau. Les nœuds équipés de système de localisation (précis ou non) n'ont qu'à être dispersés sur le lieu d'action. Toutefois, le routage, lui, est complexifié avec l'impossibilité de nommer correctement les nœuds voisins ou faisant partie d'un chemin vers la station centrale de collecte de données.

Des travaux actuels combinent les deux méthodes présentées ci-dessus [Zhao 2004]. Les capteurs sont identifiés à partir d'une adresse GPS à laquelle est associée une adresse IP. L'intérêt est de pouvoir d'une part communiquer individuellement avec un capteur et d'autre part, d'exécuter des récoltes d'information sur une zone géographique donnée.

1.4.2.5 La tolérance aux pannes

La robustesse des capteurs sans fil est un élément très important. Le premier argument dans ce sens est lié aux endroits inaccessibles pour une opération de maintenance où les capteurs peuvent être déployés. Lors d'une utilisation à l'extérieur, il est très rare de disposer d'un atelier électronique mobile adapté à une réparation des capteurs in-situ.

De plus, au vue du nombre de capteurs qui peuvent être déployés pour une application, il est important de conserver un taux de panne demandant un remplacement relativement faible. Si le taux de panne est de 10%, cela reste raisonnable pour une dizaine de capteurs mais pas pour un millier.

Dans les RCSF, les causes de pannes sont nombreuses et sont, par exemple, des dommages liés à l'environnement (chute de pierre, feu, eau, animaux,…), un manque d'énergie (batteries endommagées,…), des interférences liées à l'environnement

(interférences électromagnétiques ou liées aux bâtiments,…). Ce dernier point correspond à un état où le capteur fonctionne correctement mais n'arrive pas à communiquer avec les autres capteurs du réseau. Parallèlement à cela, on peut ajouter les pannes logicielles liées au système d'exploitation ou aux programmes embarqués dans le capteur.

La tolérance aux pannes peut également être définie pour le RCSF dans sa globalité. Le niveau de tolérance d'un réseau correspondrait à la capacité qu'aurait celui-ci à faire face aux différentes pannes des capteurs qui le composent. Cette capacité se mesurerait en nombre de capteurs pouvant tomber en panne sans que cela n'empêche le réseau d'exécuter la tâche qui lui a été attribuée.

Dans une application d'acquisition de température sur plusieurs zones, le niveau de tolérance aux pannes pourrait correspondre au nombre de capteurs affecté à chacune des zones. En outre, des capteurs de secours en veille profonde pourraient venir suppléer à un moment donné des capteurs en difficulté (voir Figure 1.14).

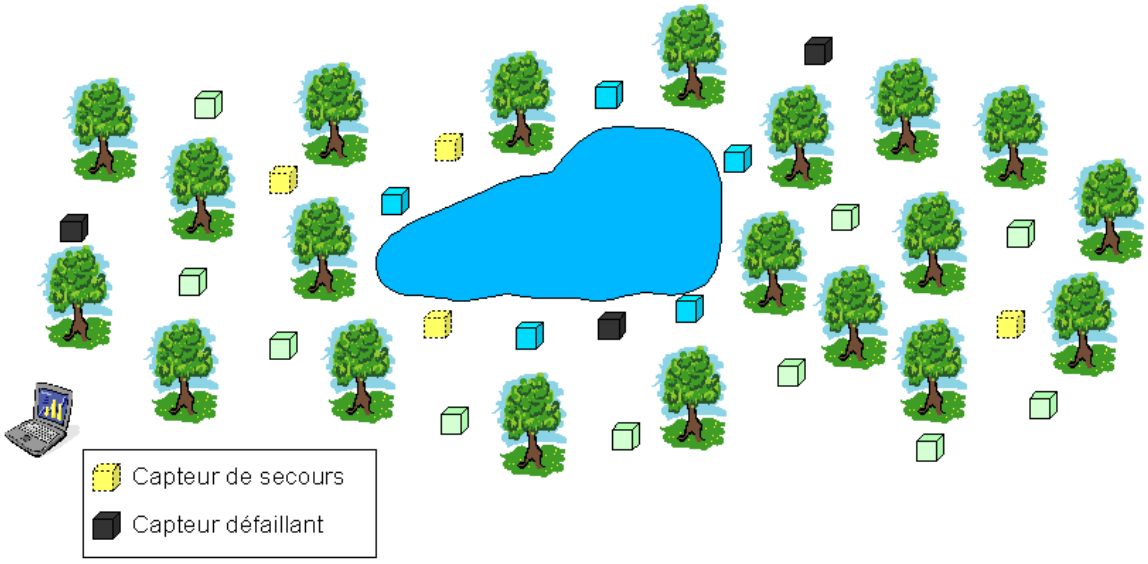

Figure 1.14 – Tolérance aux pannes dans les réseaux de capteurs sans fil

1.4.2.6 L'organisation et le fonctionnement du réseau

Une simplification trop grande du mécanisme de tolérance aux pannes serait de penser que plus le nombre de capteurs dédiés à une zone est important plus le réseau est robuste. Or, les interférences de communication augmentent avec le nombre de capteurs au mètre carré. En outre, dans un réseau très dense, le routage est fortement lié à la gestion de l'énergie. Ainsi, les routes peuvent être rallongées en terme de nombre de sauts c'est-à-dire de nœuds rencontrés pour préserver certains nœuds affaiblis. A l'inverse, à la mise en place du réseau, les routes les plus courtes doivent être préférées surtout celle qui relie le nœud à la station centrale de collecte de données. Ces problématiques s'inscrivent dans une démarche de recherche à mi-chemin entre réseau et recherche opérationnelle.

Les aspects qui viennent d'être cités font partie des éléments à considérer au niveau de l'organisation du réseau. Face à une application donnée, l'une des premières choses à établir est le nombre de capteurs à déployer c'est-à-dire le dimensionnement du réseau. La complexité de celui-ci dépend du degré de connaissance sur le phénomène observé. Dans une application de surveillance d'incendie ou de suivi à la trace, le nombre et l'emplacement des capteurs sont parfois choisis aléatoirement.

Par ailleurs, de multiples architectures de réseau existent et s'appuie sur des topologies à plat, hiérarchiques à un ou plusieurs niveaux, avec ou sans groupes ou clusters dont les contours sont parfaitement dessinés. L'une ou l'autre organisation est encore choisie selon des critères de gestion d'énergie, de protocole de routage et de contraintes d'application.

Des topologies en clusters vont souvent de paires avec des réseaux hétérogènes au niveau des capteurs utilisés. Les nœuds chefs de cluster peuvent être en charge de la centralisation, de l'agrégation et de la transmission des données collectées vers la station centrale. Par conséquent, afin d'assurer leur tâche, ils peuvent être munis de plus de ressources. Ces topologies apportent des solutions au processus de mise à l'échelle des RCSF mais complexifie sa gestion. Les difficultés associées sont la construction des clusters [Diao 2007a] et le déploiement dans le sens du positionnement de ces nœuds chefs. Pour ce dernier point, il en va de même si des nœuds à position fixe connue ont à être déployés. Un réseau est donc hétérogène par les différences matérielles de ses capteurs et/ou par les tâches évoluées allouées à ces mêmes capteurs (agrégation, routage,…).

L'habilité de reconfiguration d'un réseau est un point à prendre en compte dans son fonctionnement. En cas de déploiement dans un lieu peu ou pas accessible, l'attribution à distance de nouvelles tâches ou missions aux nœuds est une fonctionnalité intéressante voire primordiale. SMP (Sensor Management Protocol) [Arkyildiz 2002] est un protocole conçu dans cette optique et intègre des fonctionnalités allant de la reconfiguration à la gestion des données. Si chaque capteur possède une identification unique, une configuration individuelle est possible. Cependant, on préfèrera probablement une reconfiguration des capteurs présents dans une zone donnée.

1.4.2.7 La sécurité

Dans les premiers travaux sur les RCSF, les aspects liés à la sécurité avaient été peu ou pas abordés. La faute n'était pas à la motivation des chercheurs mais aux ressources disponibles au sein des capteurs. Pour mettre en place une application de réseau de capteurs, il faut résoudre essentiellement les problématiques de routage, de gestion de données. La sécurité vient ensuite et s'appuie sur les ressources restantes.

Pourtant, les besoins en sécurité sont réels et indispensables pour une démocratisation des RCSF. En effet, la confidentialité des données tout au long de la chaîne d'acquisition est une nécessité pour un bon nombre d'applications. Les données doivent donc être sécurisées durant la transmission par communication sans fil et, même parfois, durant le stockage. La sécurisation d'un réseau sans fil seul est déjà complexe car elle s'applique à un support de communication non guidé. L'ajout de mécanisme de cryptographie au niveau du stockage des données rend la tâche encore plus difficile au regard de l'énergie et de la puissance de calcul disponibles dans un capteur sans fil.

Des travaux de recherche se consacrent exclusivement à la sécurité dans les RCSF. Le projet Sizzle [Gupta 2005] définit une version du protocole SSL (Secure Socket Layer) dédiée aux dispositifs à faibles ressources. L'architecture proposée s'appuie sur des capteurs de type « mica motes » de l'Université de Berkeley. Cet exemple tend à montrer qu'en toute logique, l'amélioration de la sécurité dans les RCSF se fera soit par l'adaptation d'algorithmes actuels de cryptographie, soit par le développement de nouveaux algorithmes plus économes, soit par l'augmentation des ressources disponibles. Les techniques de sécurité utilisées dans les cartes à puce forment une base de travail mais qui est moins évidente à exploiter qu'elle n'y paraît. Une carte à puce n'a pas d'alimentation propre mais a besoin pour fonctionner de l'énergie fournie par son lecteur associé.

D'autre part, l'identification par zone géographique, donc non unique pour les capteurs, ou changeante selon leur localisation est un facteur aggravant. Une politique d'adressage mal établie représente une faille exploitable par les attaques par usurpation d'identité. Un intrus malintentionné peut, après avoir trouvé sa place dans le réseau, envoyer des paquets par inondations successives ou diffusions étendues (« broadcast ») afin d'épuiser le réseau.

Les RCSF s'organisent autour du principe de coopération qui intervient dans le routage, les traitements sur les données (agrégation de données par exemple),… Les opérations effectuées sont basées sur la confiance existante au sein du réseau. La technique de sécurité employée doit constituer un compromis entre efficacité de protection et préservation d'une vitesse satisfaisante d'échange entre capteurs.

Au final, les RCSF sont, dans un futur proche, amenés à être utilisés dans des applications dédiées à la santé, aux secours d'urgence et à la surveillance. Par conséquent, les qualités en termes de sécurité dont devront se munir les réseaux de capteurs sont :

- les mécanismes d'authentification
- la confidentialité, l'intégrité et la disponibilité des données

L'intégrité des données se rapporte au stockage. Les données ne doivent pas être altérées d'un capteur à l'autre d'où, par exemple, la nécessité de somme de contrôle (« checksum ») pour détecter les erreurs potentielles.

1.4.3 Les différentes problématiques abordées

La résolution de la plupart des problématiques qui viennent d'être énoncées passe par la mise à disposition au niveau de chaque capteur ou du RCSF d'un ensemble de données ou d'informations. Par exemple, si l'on considère le routage, des informations sur la position des nœuds voisins à chaque capteur ou sur les chemins disponibles doivent pouvoir être accessibles. Ces informations peuvent ensuite être couplées à celles sur la QdS pour déterminer la qualité des chemins disponibles.

Au sein d'un RCSF, chaque capteur manipule des données appartenant à l'une ou à l'autre des catégories suivantes :

- les données collectées ;
- les données de fonctionnement.

Les données collectées regroupent l'ensemble des données issues des différents capteurs physiques présents dans le RCSF. Les problématiques qui sont associées à leur gestion ont été introduites dans la section précédente.

Les données appartenant à la deuxième catégorie sont essentiellement celles générées par le système d'exploitation et les différents composants logiciels qui lui sont associés comme ceux dédiés, par exemple, à la communication sans fil ou au routage.

Pour gérer ces différents types de données, chaque capteur doit principalement disposer des deux fonctionnalités suivantes :

- le stockage des données ;
- l'accès rapide aux données recherchées.

De part la relative fragilité des capteurs sans fil et/ou l'hostilité des lieux de déploiement, le stockage non volatile des données est, la plupart du temps, requis. Plus

précisément, la mise hors tension accidentelle (suite à une panne) ou souhaitée (par économie d'énergie) de certains composants du capteur oblige à prévoir ce type de stockage. Celui-ci est généralement obtenu par l'utilisation de mémoires de type Flash.

Les données, stockées dans une première étape, pourront être ensuite soit utilisées par un composant logiciel dédié au fonctionnement du capteur, soit traitées et transmises à l'utilisateur final. Dans ce dernier cas, la récupération des données peut être optimisée grâce à l'utilisation d'un moteur d'interrogation.

En règle générale, la problématique du stockage en mémoire Flash [Mathur 2006a] et celle de la gestion et de l'interrogation des données collectées sont abordées séparément [Madden 2005]. Or, quel que soit le type d'application de RCSF considéré, les données qu'elles aient été collectées ou transitées temporairement au niveau d'un capteur doivent pouvoir être sauvegardées de manière non volatile si elles n'ont pu être transmises au capteur ou à tout autre élément (station centrale de collecte) auquel elles sont adressées. En outre, le stockage doit être effectué de manière à pouvoir récupérer les informations aussi rapidement que possible mais surtout en préservant au maximum les ressources du capteur voire du RCSF. Cela tend à prouver la relation étroite qui existe entre les deux problématiques qui viennent d'être citées.

De ce fait, dans ce mémoire, nous proposons de traiter ces deux problématiques en même temps en les intégrant dans un cadre plus large qui est celui de la gestion des données dans les RCSF.

1.5 Une synthèse à cette introduction aux réseaux de capteurs

Le capteur sans fil idéal devrait avoir la taille d'un cube de quelques millimètres de côté, un prix inférieur à un euro et tout ceci en incluant le microcontrôleur, le module de communication sans fil et les dispositifs d'acquisition des données [Karl 2003]. A cela, il faudrait associer des applications embarquées, tolérantes aux fautes, efficaces en termes de consommation d'énergie et avec une faible empreinte mémoire.

La réalité actuelle est toute autre. Deux catégories de capteur peuvent être établies avec d'un côté les micro-capteurs et de l'autre les capteurs « évolués ». Pour les micro-capteurs, le prix doit être le plus bas possible (inférieur à 10 €) pour permettre une dispersion par centaine voire millier. Les traitements à effectuer sont simples et répondent à des contraintes de consommation de ressources très strictes.

Pour un capteur évolué, l'approche est différente. Dans un premier cas, le développement du capteur répond à un besoin nouvellement créé et s'intègre dans une démarche d'innovation et de recherche. Ainsi, une importance toute particulière est donnée à la phase de pré-industrialisation qui permettra de réduire le coût du capteur. Dans un second cas, une solution non basée sur les capteurs sans fil existe déjà. Par conséquent, il faut en proposer une autre à un prix moindre ou plus élevé mais avec plus de fonctionnalités. L'objectif sur ce dernier point sera de bien justifier la différence de prix et prouver l'intérêt de remplacer l'ancienne infrastructure par une autre utilisant les RCSF.

Au regard du nombre d'applications possibles, les perspectives au niveau des RCSF sont immenses qu'il s'agisse d'utilisations grand public ou plus spécifiques. Deux groupes de RCSF peuvent être définis si l'on se réfère aux applications. Le premier contient les applications d'observation ou d'acquisition de données à partir de l'environnement de déploiement. Le second correspond à un emploi des RCSF en substitution d'une infrastructure filaire. Bien entendu, des applications appartenant en même temps à ces deux catégories peuvent exister.

Dans les RCSF, le comportement global et local sont bien distincts et sont à prendre en considération. La gestion des ressources est une priorité au sein du capteur alors qu'au niveau global ce qui compte est la réponse du réseau à l'application supportée. Ces deux notions sont parfois difficiles à concilier. Par exemple, l'objectif du capteur sans fil est d'économiser son énergie en réduisant au maximum le nombre de communications. Et à l'inverse, le niveau de qualité de service global va demander à ce même capteur la transmission de messages de tests de lien pour s'assurer du bon fonctionnement du réseau. Le fonctionnement local n'intègre peu ou pas certains mécanismes de gestion de données tels que l'agrégation et l'interrogation.

De vastes problématiques sont présentes au sein des RCSF. Elles prennent leur source au niveau du capteur sans fil. Les ressources limitées dont il dispose impliquent une réflexion dans l'utilisation de technologies qui sont normalement maîtrisées. Appréhender l'ensemble de ces problématiques n'est pas envisageable. Le présent manuscrit s'attache à apporter des solutions dans les domaines des systèmes d'exploitation dédiés aux RCSF, thème central du chapitre 2, et de la gestion de données avec les chapitres 3 et 4.

Chapitre 2
Les systèmes d'exploitation pour les réseaux de capteurs sans fil

Les systèmes d'exploitation représentent un socle sur lequel s'appuient les programmes d'application. Ils servent de lien ou d'interface entre l'architecture matérielle et la partie logicielle. Dans les réseaux de capteurs sans fil (RCSF), les systèmes d'exploitation conservent ce rôle. Mais, au regard des contraintes énoncées dans le chapitre 1, ils possèdent différentes fonctionnalités et en nombre plus restreint en comparaison d'un système d'exploitation « classique » c'est-à-dire présent dans un ordinateur personnel. La gestion des utilisateurs fait partie des fonctionnalités mises de côté.

Le thème central de ce chapitre est les systèmes d'exploitation dédiés aux RCSF. Dans un premier temps, quelques éléments sur les systèmes embarqués seront introduits. Puis, dans un second temps, les systèmes d'exploitation dédiés aux RCSF dont fait partie le système LIMOS (LIghtweight Multithreading Operating System) seront présentés. Enfin, une synthèse avec les notions essentielles à retenir conclura ce chapitre.

2.1 Une introduction aux systèmes d'exploitation

Les difficultés à gérer les composants matériels d'un ordinateur n'ont cessé d'augmenter. La diversité et le nombre grandissant de ceux-ci en sont les principales causes. Ce constat a amené à développer, à partir des années 1960, une couche logicielle destinée à jouer le rôle d'interface entre le matériel et les programmes d'application. Par ailleurs, l'exécution de plusieurs programmes alternativement a été mise en évidence par les pertes de temps observées durant les phases d'impression. Ces dernières monopolisaient l'unité centrale d'un ordinateur dont chaque seconde de traitement était précieuse au regard du coût du dispositif [Bloch 2003]. Dans [Tanenbaum 2006], le système d'exploitation est défini comme une machine virtuelle permettant à l'utilisateur une meilleure compréhension et une utilisation plus facile du matériel.

2.1.1 L'architecture matérielle d'un ordinateur

La représentation simplifiée de l'architecture d'un ordinateur classique donne des indications sur le rôle de coordinateur associé aux systèmes d'exploitation (voir Figure 2.1). Un ordinateur est, de manière générale, constitué des composants suivants :

- un processeur (une unité de contrôle ou de commande, une unité de traitement ou Unité Arithmétique et Logique (UAL)) ;
- des mémoires ;
- des périphériques ;
- des bus.

L'unité de contrôle a une place centrale et son rôle est la lecture, le décodage et le séquençage des instructions avant un envoi vers l'unité de traitement. Cette dernière est en charge d'effectuer les calculs demandés.

Dans un ordinateur personnel, les mémoires sont de différents types en relation avec leur fonction et la vitesse d'accessibilité aux données :

- mémoires caches de type L1 et L2 (ou mémoire statique SRAM (Static Random Access Memory)) : intégrées au processeur ;
- mémoire centrale de type DRAM (Dynamic Random Access Memory) : dédiée au stockage des informations, des résultats et des programmes à traiter ;
- mémoires auxiliaires comprenant disques durs, lecteurs de disquette, etc.

Les bus permettent la communication entre les différents éléments cités ci-dessus. Les bus sont de trois types :

- bus de données : pour l'acheminement des données entre les différents éléments ;
- bus d'adresses : pour la sélection d'un emplacement mémoire ou d'un registre ;
- bus de commandes : pour la synchronisation des différents éléments du système.

Les unités de contrôle et de traitement, les mémoires caches ainsi que les registres utilisés pour les instructions constituent le processeur.

Les périphériques ou unités d'Entrées/Sorties sont nombreux et comprennent la souris, le clavier, l'écran, les imprimantes, les scanners, etc.

Un processeur ou micro-processeur est, de manière simplifiée, constitué d'une unité de contrôle, d'une unité de traitement, de mémoire cache et de registres (utilisés dans le cadre des instructions).

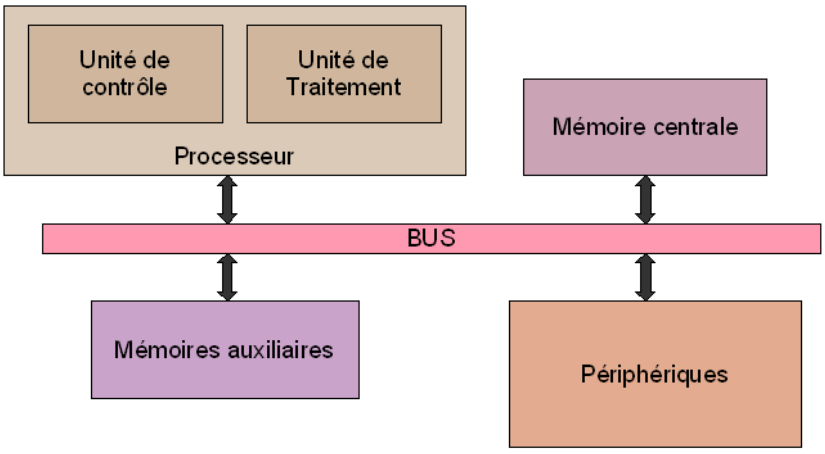

Figure 2.1 - Architecture matérielle simplifiée d'un ordinateur

2.1.2 Les fonctionnalités d'un système d'exploitation

Au regard des éléments de la section précédente, les fonctionnalités d'un système d'exploitation sont réparties de la manière suivante :

- La gestion de la mémoire ;
- La gestion des processus ;
- La gestion des périphériques ;
- La gestion des fichiers.

2.1.2.1 La gestion de la mémoire

La gestion de la mémoire ne se résume pas aux fonctionnalités d'écriture et de lecture des données. Elle comprend des mécanismes complexes tels que :

- l'allocation dynamique de mémoire avec la gestion de zones spécifiques telles que les tas (« heap ») et les piles (« stack ») ;
- l'implantation de la mémoire virtuelle ;
- la protection des zones de mémoires ;
- le ramasse-miettes ou récupérateur de mémoire.

Ces éléments, comme par exemple la mémoire virtuelle, ne sont pas présents dans tous les systèmes d'exploitation [de Vaulx 2003]. Quoi qu'il en soit, la gestion de la mémoire est d'autant plus importante que la mémoire physique est disponible en faible quantité comme c'est le cas dans les RCSF.

2.1.2.2 La gestion des processus

Une application comprend différents programmes qui sont eux-mêmes composés d'un ensemble d'instructions. L'exécution d'un programme par un processeur crée un *processus* ou une *tâche* dont la gestion est à la charge du système d'exploitation [Bonnet 1999].

Au processus est associé son contexte qui correspond à l'ensemble des programmes et des ressources nécessaires à son exécution. Plus précisément, ce contexte contient l'état des différents registres du processeur, le pointeur courant de la pile de stockage des données temporaires et l'adresse de la prochaine instruction à exécuter plus connue sous le terme de *compteur ordinal* (voir Figure 2.2).

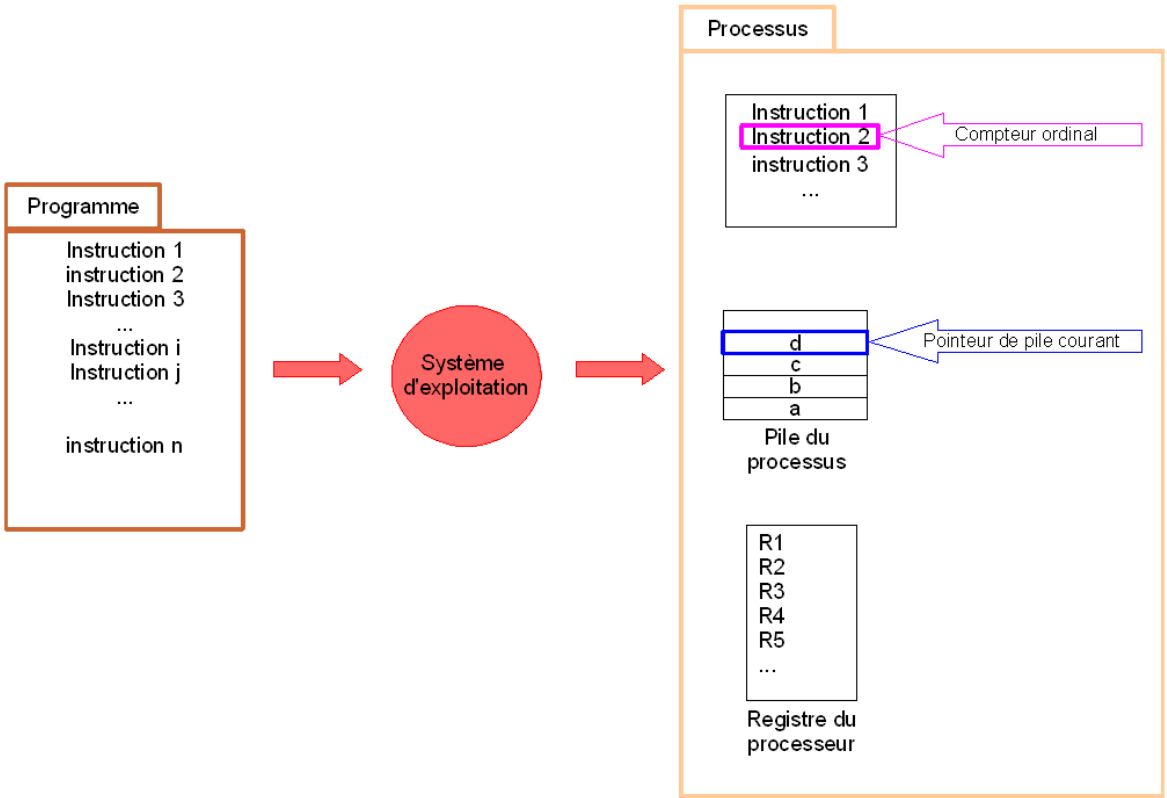

Figure 2.2 – Passage d'un programme à un processus

Généralement, on distingue les notions de processus lourd (« process ») et de processus légers (« thread »). Un processus lourd regroupe l'ensemble des ressources nécessaires pour l'exécution du programme comme l'espace d'adressage comprenant, entre autres, le code et les données qui s'y rapportent. Mais l'exécution proprement dite est à la charge d'un ou, suivant les cas, de plusieurs processus légers [Bonnet 1999] [Tanenbaum 2006]. Un processus lourd dispose donc d'au moins un processus léger. Et, un processus léger est obligatoirement associé à un processus lourd.

Les processus légers partagent un certain nombre de ressources du processus lourd auquel ils appartiennent. Ainsi, de la même manière que les processus lourds sont en concurrence entre eux pour accéder au processeur, les processus légers d'un même processus lourd le sont également en ce qui concerne les ressources qu'ils partagent. Ainsi, les processus légers comme les processus lourds peuvent prendre différents états.

Selon les systèmes, différents types de processus sont disponibles. Le micronoyau DREAM (Distributed REAl-time Micro-kernel) [de Vaulx 2003] dispose, par exemple, de :

- processus *périodiques* ;
- processus *apériodiques*.

Les processus périodiques sont exécutés à intervalles réguliers prédéfinis par l'utilisateur. Ce fonctionnement est adapté aux applications d'acquisition de données à une fréquence d'échantillonnage fixée. La collecte de la température d'un endroit toutes les heures illustre la nécessité de tels processus.

Les processus apériodiques sont plus classiques et s'appliquent à l'observation de phénomènes aléatoires et donc aux applications de détection d'intrusion ou de suivi à la trace. Le système d'exploitation n'est jamais inactif même en dehors des phases d'acquisition et de traitements de données. Un processus apériodique particulier, le *processus* ou *tâche de fond* est en charge de maintenir le système en alerte au cas où un événement surviendrait.

La gestion des processus comprend à la fois l'ordonnancement et la communication et synchronisation des processus. Une répartition par ordre d'importance des traitements à effectuer est nécessaire pour accéder aux ressources mises à disposition par l'ordinateur. Cette opération est appelée *ordonnancement* et est à la charge d'une partie du système d'exploitation qui est appelée *ordonnanceur* (« scheduler »). Différentes politiques d'ordonnancement existent et seront présentées tout au long de ce chapitre.

La communication et la synchronisation des processus est une autre fonction majeure du système d'exploitation. Dans les premiers systèmes, les processus étaient bien séparés les uns des autres avec, la plupart du temps, une exécution séquentielle sans transmission de résultat. L'opérateur lançait l'exécution d'un premier traitement, recueillait le résultat et le donnait en paramètre au traitement suivant. Or, dans un souci d'efficacité et, en réponse à certaines applications, la communication voire la synchronisation entre processus se sont avérées indispensables. Là aussi, différentes techniques existent comme l'utilisation de sémaphores.

2.1.2.3 La gestion des périphériques

Le système d'exploitation gère les périphériques à l'aide de l'ensemble des pilotes associés à ceux-ci. Ce principe oblige à modifier certains éléments du système d'exploitation à chaque changement de périphériques. Dans l'optique de faciliter le passage d'un périphérique à un autre de même catégorie (une imprimante par exemple), la disponibilité et l'utilisation de pilotes (« drivers ») standards seraient une avancée importante.

2.1.2.4 La gestion des fichiers

Les quantités de données produites par les processus d'une application sont de plus en plus importantes et ne permettent pas toujours un stockage uniquement dans les mémoires du processeur et centrale. De plus, les données stockées dans ce type de mémoire seraient perdues soit à la fin de l'exécution du processus, soit au redémarrage de l'ordinateur. Ainsi, l'archivage de ces données doit avoir lieu dans des mémoires auxiliaires non volatiles et pas de manière brute mais sous forme d'un fichier. Un fichier est constitué d'un ensemble de blocs de caractères encodant l'information. Leurs principales particularités résident dans leurs métadonnées associées (type et taille de fichiers,…) et la possibilité de les organiser en arborescence. Toutes les opérations de gestion des fichiers font partie des attributions du système d'exploitation.

Tous les fichiers ont un propriétaire choisi parmi les utilisateurs du système d'exploitation. Dans un ordinateur personnel, la présence de plusieurs utilisateurs n'est pas rare. Dans une famille, chaque membre peut demander un accès privilégié à l'ordinateur c'est-à-dire disposer d'un compte où sont présentes ses données et les applications dont il a besoin. En fait, les privilèges d'un utilisateur sont mesurables par rapport aux fichiers qu'il a le droit de manipuler et aux processus qu'il peut exécuter. La notion de « super-utilisateur » ou d'« administrateur » existant dans beaucoup de systèmes d'exploitation est un cas particulier d'utilisateur pouvant accéder à tous les fichiers et exécuter tous types de processus et instructions au niveau du processeur. Par conséquent, la gestion des utilisateurs peut être issue de celle des fichiers et des processus (voir Figure 2.3).

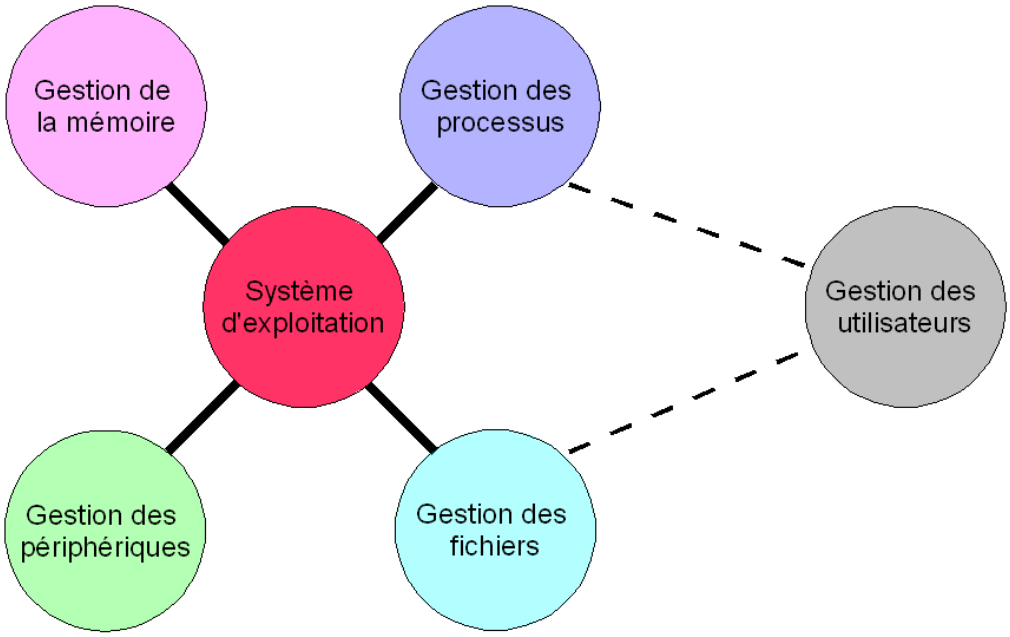

Figure 2.3 – Fonctionnalités d'un système d'exploitation

2.1.3 Les systèmes d'exploitation temps réel

Dans le chapitre 1, les applications de détection d'intrusion ou de début d'incendie ont mis en évidence la nécessité de disposer de RCSF réactifs, disposant d'une certaine autonomie de contrôle et d'exécution. La plupart des applications de RCSF définissent des intervalles de temps dans lesquelles l'acquisition et les traitements des données doivent être effectués.

Pour une application de type acquisition de données, la durée de cet intervalle est étroitement liée à la fréquence d'acquisition ou la période d'échantillonnage. Pour les applications de substitution de l'infrastructure fixe, les données doivent circuler à la vitesse offerte par le module de communication sans fil ce qui implique de minimiser le ralentissement occasionné par le protocole de routage utilisé.

Par conséquent, dans bien des cas, les capteurs sans fil doivent être munis de systèmes d'exploitation temps réel. Ils peuvent être temps réel dur ou lâche (« soft ») suivant que le retard dans un traitement déclenche ou non une exception [Bonnet 1999]. Le manquement à une échéance de traitement a des conséquences plus ou moins graves selon l'application. Pour certaines, la perte d'une donnée n'aura que très peu d'impact A l'inverse, dans la surveillance de personnes atteintes de problèmes cardiaques [Zhou 2004b], toutes les données sont importantes pour établir un diagnostic correct.

Un système d'exploitation temps réel n'a pas forcément vocation à être plus rapide qu'un système classique. Que le traitement à réaliser le soit toutes les secondes ou tous les ans, son seul objectif est de l'accomplir dans l'intervalle de temps qui lui est indiqué. Face à cette problématique, les systèmes d'exploitation dédiés aux RCSF développent différentes techniques, qui seront présentées dans la section suivante, afin de garantir l'exécution des processus dans un intervalle temporel borné.

2.2 La conception d'un système d'exploitation dédié aux réseaux de capteurs sans fil

Pour répondre aux contraintes de ressources des RCSF, certaines parties voire l'ensemble du système d'exploitation peuvent être organisées différemment d'une structure classique en couches superposées (voir Figure 2.4).

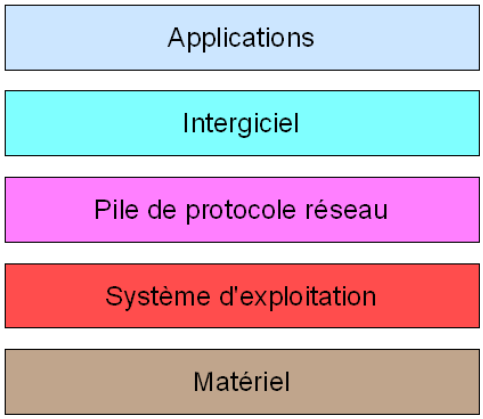

Figure 2.4 – Architecture par couches superposées

En effet, dans les RCSF, l'approche par « couches croisées » (« cross-layer design ») partielle ou totale est privilégiée par rapport à celle par couches superposées classique. Ce choix s'applique non seulement à l'interaction entre les composants du système d'exploitation mais également à la structure interne de ceux-ci. On peut citer l'exemple de la pile de protocole réseau dont certaines parties peuvent être fusionnées entre elles ou avec d'autres composants du système d'exploitation.

Une organisation par couches superposées apporte une structure plus stable et bien définie. Les interférences entre les différents niveaux sont limitées par une communication inter-couches basée sur des interfaces bien établies (voir Figure 2.5).

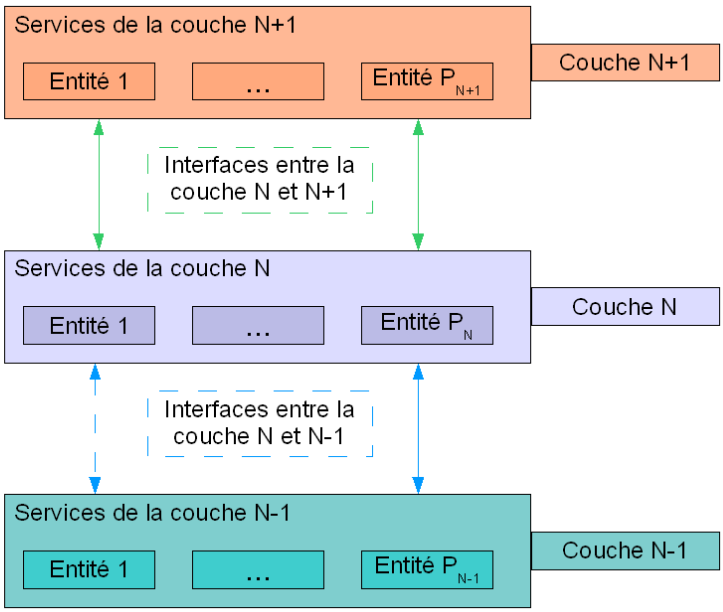

Figure 2.5 – Communication par interfaces dans un modèle en couches superposées

Toutefois, comme observé dans l'utilisation du modèle de conception OSI (Open Systems Interconnection) (voir Figure 2.6), la mise en place est parfois complexe et mal adaptée à une application réelle qu'elle concerne les RCSF [Akyildiz 2002] ou non [Tanenbaum 2003]. L'exemple des couches « Présentation » et « Session » du modèle OSI peu utilisées en pratique en sont une illustration.

Inversement, la conception par « couches croisées » est plus rapide et réduit la quantité d'énergie consommée car chaque couche transversale est conçue dans cette optique. Le principal inconvénient de cette approche est une difficulté accrue dans la maintenance et la réutilisabilité. Chaque module est imbriqué de manière spécifique dans le système d'exploitation. L'intégration de l'un de ces modules dans un autre système en est complexifiée. Apporter des solutions à ce type de problème était et reste l'objectif du modèle OSI pour l'interconnexion entre systèmes hétérogènes.

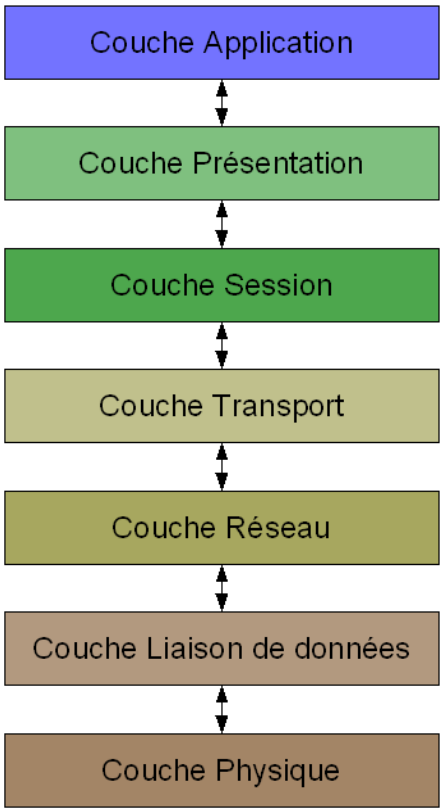

Figure 2.6 – Le modèle OSI en couches superposées

Un système d'exploitation peut, au premier abord, avoir une organisation en couches superposées avec un empilement de composants. Toutefois, si l'on s'intéresse aux fonctionnalités telles que la communication, la gestion des données, une structure plus complexe peut être extraite. D'autres systèmes d'exploitation sont construits autour de composants reliés les uns aux autres sans hiérarchie clairement établie.

2.3 Les systèmes multitâches

Le nombre croissant et la diversité des équipements d'un capteur sans fil requièrent une gestion des ressources par un système d'exploitation. En outre, des périphériques divers et variés comme le module de communication ou le dispositif d'acquisition sont connectés au capteur sans fil par l'intermédiaire d'une interface RS232, USB, SPI ou I^2C.

Dans le domaine des systèmes d'exploitation temps réel, une distinction existe entre les systèmes pilotés par le temps et ceux pilotés par les événements [Bonnet 1999]. Les premiers effectuent leurs traitements à des instants déterminés à partir d'un référentiel temporel interne. Les seconds systèmes n'accomplissent leur tâche qu'à la suite de stimuli provenant de leur environnement. Cette classification se retrouve partiellement dans les systèmes dédiés aux RCSF [Stojmenovic 2005].

L'étude, par des exemples, de systèmes d'exploitation dédiés aux RCSF est le point central des sections suivantes.

Pour répondre à ce besoin, certains travaux de recherche ont adapté, au domaine des RCSF, l'architecture des systèmes d'exploitation multitâches classique. Le système d'exploitation UNIX est un exemple de système multitâche mais ne peut être utilisé dans les RCSF pour deux raisons majeures. La première est qu'il ne s'agit pas d'un système temps réel qui, comme souligné dans le paragraphe 2.1.3, est une propriété importante pour les RCSF. L'autre raison est son empreinte mémoire qui est trop importante pour un capteur sans fil.

2.3.1 Les systèmes multitâches embarqués temps réel

Des études et des comparaisons entre les principaux systèmes temps réel existants ont été réalisés [Hill 2000] [de Vaulx 2003] [Zhou 2004b]. Les premières conclusions ont eu tendance à démontrer les manquements de ces systèmes dans une utilisation dans des dispositifs à ressources limitées très proches de capteurs sans fil. En effet, d'une part, le nombre de systèmes dont l'empreinte mémoire n'excède pas les 10Ko est restreint (voir Table 2.1). D'autre part, ces systèmes ne disposent pas de certaines fonctionnalités importantes et en proposent d'autres non indispensables. Un capteur sans fil implique une gestion particulière liée aux ressources restreintes disponibles. Ainsi, toute fonctionnalité du système d'exploitation mal pensée ou superflue pénalise l'ensemble du dispositif.

Systèmes D'exploitation	Windows CE .NET	RT Linux	QNX	µC/OS-II	DREAM
Complexité spatiale (Ko)	$\geq$ 200	$\geq$ 250	> 64	> 6	> 6

Table 2.1 – Taille de quelques systèmes d'exploitation embarqués temps réel

2.3.2 Le système d'exploitation MANTIS

Le système d'exploitation MANTIS (MultimodAl system for NeTworks of In-situ wireless Sensors), conçu au sein de l'Université du Colorado à Boulder, est un exemple de système multitâche dédié aux RCSF [Abrach 2003] [Bhatti 2005]. En effet, sa configuration de base adaptée aux applications de réseaux de capteurs comprenant le noyau, l'ordonnanceur et la pile de protocole réseau a une empreinte mémoire inférieure à 1Ko.

Plus précisément, le système MANTIS est un système multitâche préemptif basé sur le partage de temps. Les ressources logicielles et matérielles du système sont ainsi affectées alternativement et pour un temps fixé entre les différents processus légers en cours

d'exécution. Ce fonctionnement fait en sorte qu'un programme ne puisse pas être mis en attente ou bloquer le système indéfiniment. Ainsi, les tâches de longue durée ne peuvent pas accaparer les ressources du système au-delà d'un certain temps au détriment d'une autre dont l'exécution est critique temporellement.

A une taille réduite, le système MANTIS dispose de fonctions absentes des systèmes d'exploitation mentionnés dans la Table 2.1. Par exemple, comparativement avec µCos dont l'empreinte mémoire est relativement faible, le système MANTIS utilise un ordonnanceur prenant en compte la consommation énergétique et comprend un mécanisme de programmation à distance des capteurs sans fil. Il a également la possibilité de passer en phase de veille quand aucun traitement n'est à réaliser.

L'architecture complète (voir Figure 2.7) du système d'exploitation MANTIS intègre différents éléments regroupés de la manière suivante :

- le noyau et l'ordonnanceur ;
- le module de communication (COMM) et la pile de protocole réseau ;
- le gestionnaire de périphériques (DEV) ;
- l'API (Application Programming Interface) du système ;
- l'interpréteur de commande (shell).

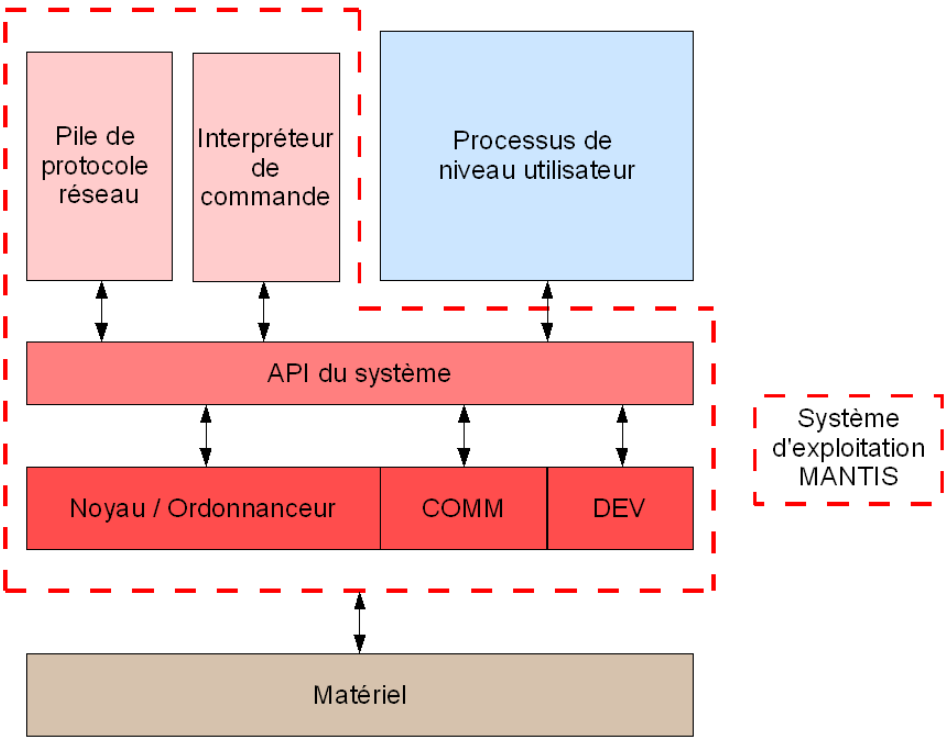

Figure 2.7 – Architecture du système d'exploitation MANTIS [Bhatti 2005]

Cette architecture illustre bien la conception en « couches transversales » avec une pile de protocole réseau située sur la même couche que les processus de niveau utilisateur.

Le rôle d'interpréteur de commande (shell) est de fournir un accès à distance à l'utilisateur. Ce dernier peut interagir avec le système par l'intermédiaire de fonctions prédéfinies. Les actions possibles se répartissent entre le changement de configuration, l'inspection et la modification de la mémoire et des opérations sur les processus (démarrage, suppression,…).

L'API du système permet simplement, à l'aide de la primitive *thread_new()*, de créer de nouveaux processus. La demande de création de ces processus est effectuée par une application donnée de niveau utilisateur ou par la pile de protocole réseau ou le serveur de commande. Les autres modules du système vont être présentés plus en détails dans ce qui suit.

2.3.2.1 Le noyau MANTIS

Le système MANTIS comprend les principales fonctionnalités des systèmes d'exploitation classiques présentées dans le paragraphe 2.1.2. Tout d'abord, tout en incorporant les contraintes de ressources limitées propres aux RCSF, le système MANTIS est basé sur des ordonnanceurs de type UNIX. La politique d'ordonnancement s'appuie sur la notion de priorité. Les processus de priorité la plus élevée s'exécutent avant ceux de plus faible priorité. Pour des processus de même priorité, le principe du *tourniquet avec quantum de temps* est utilisé. Dans celui-ci, les processus de même priorité s'exécutent alternativement durant un temps fixe et ceci jusqu'à leur terminaison. Pour la gestion des ressources qui pourraient être partagées, des mécanismes tels que les mutex ou les sémaphores peuvent être implémentés dans le système MANTIS.

Une attention toute particulière est portée sur la gestion de la mémoire. Dans MANTIS, la mémoire RAM disponible est répartie entre un espace alloué au moment de la compilation pour les variables locales et le tas réservé aux processus. Chaque processus réserve de la mémoire du tas pour leur pile qui est rendu à la fin de son exécution (voir Figure 2.8). A la base, il n'est pas possible d'augmenter dynamiquement la taille du tas sous MANTIS mais cela peut être ajouté en s'appuyant sur l'API. Toutefois, cette fonctionnalité est techniquement complexe à mettre en place et est plutôt déconseillée. Cette restriction est un choix qui se retrouve dans la plupart des systèmes d'exploitation développés pour les RCSF.

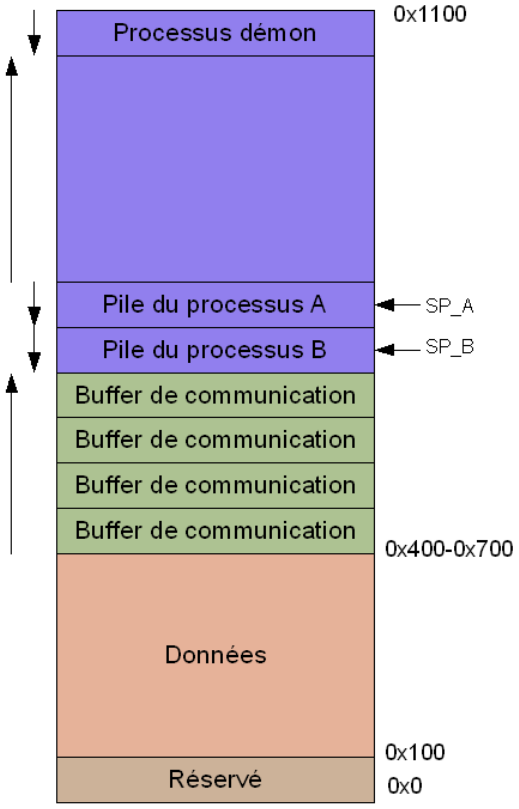

Figure 2.8 – Organisation de la mémoire dans le système MANTIS [Stojmenovic 2005]

Par défaut, la taille du tas spécifié permet un nombre maximum de douze processus. Les processus sont représentés à l'aide d'une table dont chaque entrée a une taille de dix octets et contient les informations suivantes :

- la dimension de la pile avec sa taille et son pointeur de début ;
- le pointeur courant de pile ;
- le pointeur sur la fonction de démarrage du processus ;
- le niveau de priorité du processus ;
- le pointeur vers le processus suivant.

Ces éléments permettent, à partir de la pile d'un processus suspendu, de sauvegarder et de restaurer son contexte courant (contenus des registres du microprocesseur, etc.). A ce propos, l'ordonnancement du système MANTIS utilisant la politique du tourniquet avec quantum de temps, ce dernier est fixé, par défaut, à 10ms. Les processus de même priorité sont regroupés dans une même liste dont le noyau dispose des pointeurs de début et de fin et sur le processus courant.

Dans les premières versions du système MANTIS, seules les interruptions matérielles sont supportées. Elles sont toutes transmises aux pilotes du dispositif associé sauf celles de type « compteur de temps » (ou « timer ») qui sont gérées par le noyau lui-même. Plus précisément, suite à une interruption, un sémaphore est activé dans le but de réveiller un processus en attente. Sous MANTIS, les sémaphores avec compteur et d'exclusion mutuelle (ou *mutex*) sont utilisables.

Le dernier élément à considérer est l'existence d'un processus démon de priorité la plus basse. Il s'exécute quand tous les autres processus sont soit endormis ou non activés, soit suspendus en attente d'une ressource ou du déclenchement d'une interruption. Les actions visant à économiser l'énergie consommée peuvent être du ressort d'un tel processus.

2.3.2.2 La communication

Ce système est dédié aux RCSF et comprend des éléments pour la communication. L'architecture du système MANTIS inclut une pile de protocole réseau qui suit, dans ce cas, un découpage en couches :

- la couche Application ;
- la couche Transport ;
- la couche Réseau ;
- la couche Communication.

Les trois premières couches reprennent, au niveau fonctionnalité, les recommandations fournies par le modèle OSI. La couche Communication est plus singulière car, contrairement aux autres couches, elle est située en dehors du niveau utilisateur et au même niveau logique que le noyau. L'un de ses rôles est le support de la sous-couche MAC associée au protocole de communication utilisé. De base, la sous-couche MAC permet une communication radio par modulation de fréquences sur 30 canaux. D'autres techniques d'accès au médium sans fil telles que le TDMA (Time Division Multiple Access) et CSMA (Carrier Sense Multiple Access) peuvent être utilisées. Une autre fonctionnalité visée est d'essayer d'offrir une interface standard aux pilotes des dispositifs de communication employés.

Cette pile de protocole, à la manière de celle d'autres systèmes, est développée de manière à minimiser le nombre de buffers, et par conséquent la mémoire utilisée, pour l'émission et la réception de messages [Zhou 2004b]. Pour ce faire, la partie « données » du paquet est stockée dans le même emplacement du début à la fin de son parcours dans la pile.

Quatre primitives sont utilisées en pratique pour la communication :

- la fonction *com_send()* : envoi de paquets ;
- la fonction *com_recv()* : réception de paquets ;
- la fonction *com_mode()* : activation ou désactivation du module de communication ;
- la fonction *com_iotcl()* : action spécifique à chaque module de communication.

La fonction *com_send()* crée un processus d'envoi niveau utilisateur qui à l'aide d'un pointeur va référencer un emplacement mémoire ou « buffer de communication ». L'entité COMM du système va ensuite prendre le relai de ce processus qui sera mis en attente, et transmettre le message au dispositif de communication. Après l'envoi physique du paquet, le déclenchement d'une interruption viendra débloquer le processus d'envoi précédemment suspendu.

La réception des messages est de l'autorité de l'entité COMM. Cette dernière possède un certain nombre de buffers de communication qu'elle met à disposition du module de communication. Lorsqu'un processus fait appel à la fonction *com_recv()*, il reste bloqué jusqu'à l'arrivée d'un paquet dans un buffer de communication annoncée par une interruption. Une rotation est opérée entre les buffers de communication pour éviter la copie de données dans leur cheminement vers les couches hautes du système.

Par ce mode de fonctionnement, l'envoi de message est synchrone au niveau du système mais la réception peut être quasiment gérée en tâche de fond par l'entité COMM. En outre, cette entité est en charge de la gestion des communications asynchrones au niveau du système MANTIS.

2.3.2.3 Les autres périphériques

Les communications synchrones sont du ressort de l'entité DEV. Elle assure donc, en général, la communication avec les dispositifs de collecte de données et des périphériques tels que ceux de stockage externe avec, par exemple, les EEPROM (Electrically Erasable Programmable Read Only Memory) (voir Figure 2.9).

De la même manière que pour l'entité COMM, chaque type de dispositif a une interface associée dans l'entité DEV. Les primitives pour utiliser ces interfaces sont :

- la fonction *dev_read()* : lecture sur le périphérique ;
- la fonction *dev_write()* : écriture sur le périphérique ;
- la fonction *dev_mode()* : activation ou désactivation du périphérique ;
- la fonction *dev_ioctl()* : action spécifique à chaque périphérique (e.g. configuration).

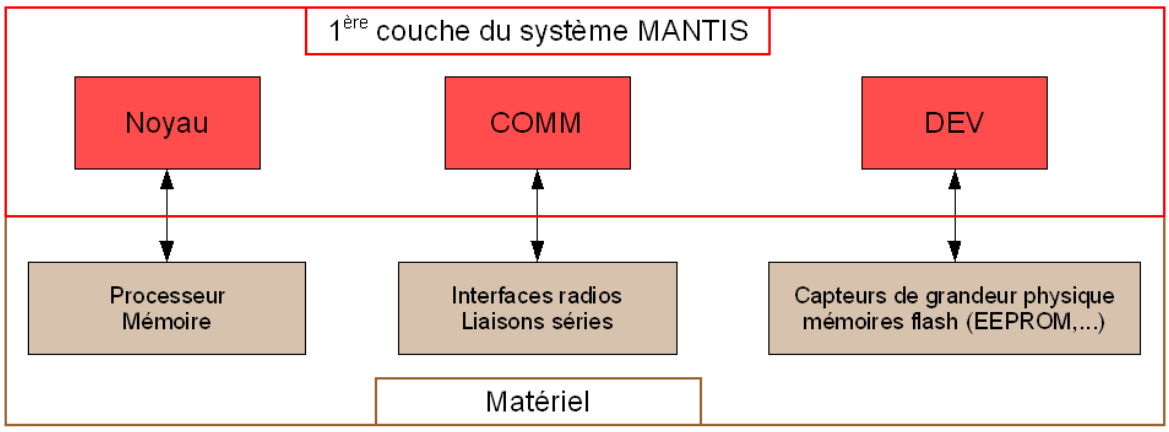

Figure 2.9 – Support de la couche matérielle dans le système MANTIS

Comme le montrent les primitives utilisées, le mode de fonctionnement de cette entité est quasiment identique à celui de l'entité COMM. Quelques différences existent cependant : chaque élément géré étant répertorié dans une table de taille fixe qui comprend la localisation des fonctions associées à celui-ci et un sémaphore d'exclusion mutuelle.

2.3.2.4 Les avantages et inconvénients du système MANTIS

En se basant sur son architecture et l'organisation des différents composants logiciels qui le constituent, il est possible d'établir quelques uns des avantages et inconvénients propres au noyau MANTIS. Tout d'abord, nous sommes en présence d'un système d'exploitation embarqué multitâche adapté aux applications où plusieurs traitements, chacun associé à un ou plusieurs processus, sont en concurrence pour accéder aux ressources du capteur sans fil. Ce fonctionnement facilite également le portage de programmes provenant d'un système multitâche classique UNIX. Ce type d'opération doit toutefois être réalisé après une étude sur la possibilité, au regard des ressources disponibles, d'effectuer ou non ce portage.

Dans le cadre des RCSF, cette architecte comporte également certains inconvénients. Dans les systèmes d'exploitation multitâches, en raison du passage d'un processus à un autre, des changements de contexte se produisent. Ceux-ci sont sources à la fois d'une certaine lenteur et d'une consommation de mémoire proportionnelle au nombre de processus, chacun d'entre eux nécessitant un espace de mémoire propre. Or, suivant l'application de RCSF considérée, ces deux critères d'évaluation sont déterminants dans le choix du système d'exploitation utilisé.

2.4 Les systèmes basés sur les événements

De part leur mode de fonctionnement, les systèmes d'exploitation pilotés par les événements sont adaptés aux RCSF [Stojmenovic 2005]. L'architecture de ce type de système est organisée autour des traitements à effectuer et de la gestion des ressources disponibles. Le système d'exploitation TinyOS, développé à l'Université de Californie de Berkeley, fait partie de cette famille de systèmes d'exploitation et permet d'en illustrer certaines caractéristiques communes [Hill 2000]. En outre, TinyOS représente un cadre pour le développement de systèmes d'exploitation dédiés aux RCSF.

2.4.1 L'architecture du système TinyOS

Les événements pris en compte et leurs réponses associées sont centraux dans TinyOS. Sous ce système, une application est construite à partir de l'association et de l'empilement de différents composants. Ces derniers sont constitués des éléments suivants :

- un gestionnaire de commandes ;
- un gestionnaire d'événements ;
- une liste de processus associés ;
- un contexte d'exécution avec un espace mémoire de taille fixe.

Les commandes et les événements permettent à l'aide d'interfaces la communication entre les composants (voir Figure 2.10).

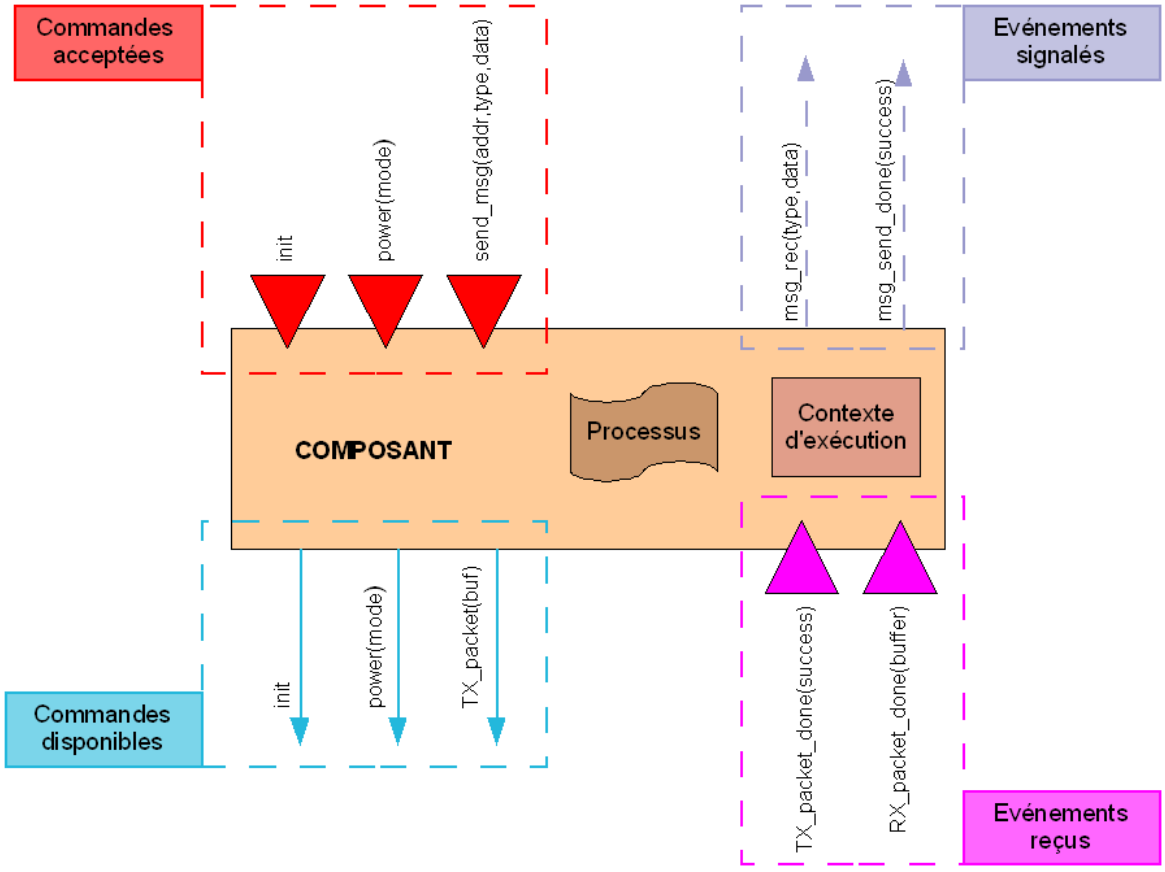

Figure 2.10 –Exemple de composant du système TinyOS avec ses interfaces [Hill 2000]

Les commandes, comme leur nom l'indique, sont des ordres ou des demandes adressés aux composants de niveau inférieur. A l'inverse, les événements se propagent à partir du matériel et remontent jusqu'à atteindre le composant de plus haut niveau de l'application (voir Figure 2.11).

```
/*Déclaration du Composant */

// Commandes acceptées
char TOS_COMMAND(AM_send_msg)(int addr;int type,char* data);
void TOS_COMMAND(AM_power)(char mode);
char TOS_COMMAD(AM_init)();

// Commandes disponibles
char TOS_COMMAND(AM_SUB_TX_packet)(char* data);
void TOS_COMMAND(AM_SUB_power)(char mode);
char TOS_COMMAND(AM_SUB_init)();

// Evénements signalés
char AM_msg_rec(in type,char* data);
char AM_msg_send_done(char success);

// Evénements reçus
char AM_TX_packet_done(char success);
char AM_RX_packet_done(char* packet);
```

Figure 2.11 – Eléments associés au composant précédent [Hill 2000]

Ces événements sont essentiellement matériels, initiés par des interruptions déclenchées par une horloge, un dispositif d'observation de grandeur physique ou un module de communication. Cependant, selon que l'on remonte l'arbre des composants formant une application, les événements peuvent rendre compte de la fin d'opérations effectuées sur les données (agrégation, formatage, etc.) et donc être considérés comme étant de type logiciel (voir Figure 2.12). On retrouve cet aspect dans la répartition des composants en trois catégories :

- les composants d'abstraction de niveau matériel ;
- les composants « synthétiques » de niveau matériel ;
- les composants de haut niveau logiciel.

Le rôle des composants d'abstraction de niveau matériel est l'intégration des dispositifs matériels dans le modèle basé sur les composants. Ceux-ci offrent aux composants de niveau supérieur des interfaces (commandes) pour la gestion des entrées/sorties tels que les ports série. Les composants « synthétiques » implémentent des opérations de base sur les données. Dans une application de communication [Levis 2004], un tel composant regrouperait en blocs d'octets les données reçues bit par bit provenant du composant d'abstraction. Tous les traitements ou fonctions élaborés, requis par l'application, sont contenus dans des composants de haut niveau logiciel. Ce type de composants contient les processus dédiés au routage, au cryptage et à l'agrégation de données.

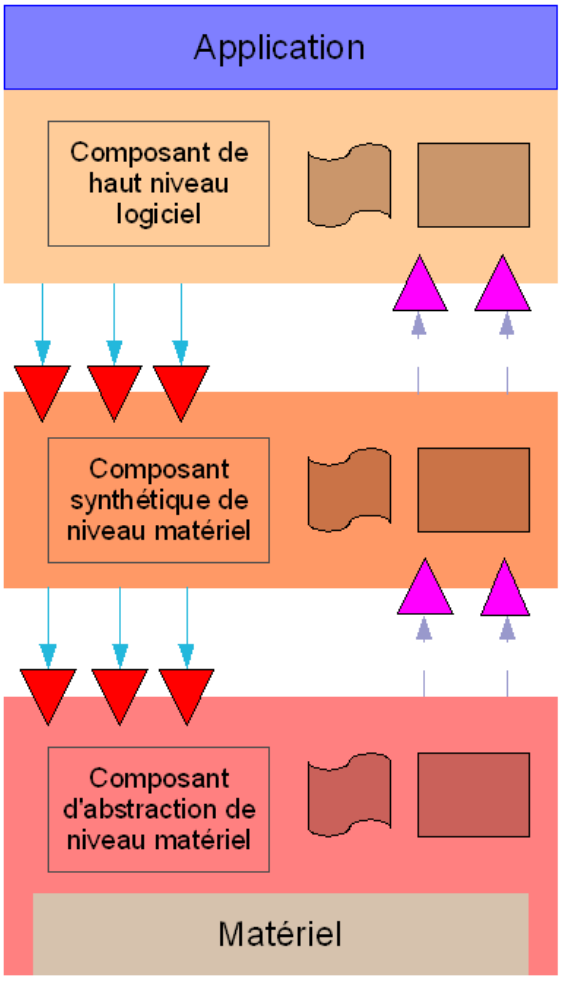

Figure 2.12 – Les différents types de composants du système TinyOS

2.4.2 L'ordonnancement dans le système TinyOS

L'autre particularité du système TinyOS concerne sa politique d'ordonnancement des processus. Un événement matériel comme la réception d'un message va déclencher une réaction de la part du composant associé à la communication. Cette réaction va se matérialiser par l'exécution d'un ensemble de processus. Mais contrairement à un système multitâche, les processus vont s'exécuter successivement sans pouvoir être préemptés [Levis 2004]. Ce procédé permet d'une part d'éviter les changements de contexte potentiels qui ont lieu durant le passage d'un processus actif à un autre et implique donc une accélération de la vitesse de traitement. D'autre part, la quantité de mémoire nécessaire est réduite car, en supprimant les changements de contexte, les processus n'ont plus besoin d'un espace mémoire propre. Ils vont accéder chacun leur tour à l'espace mis à disposition au niveau du composant. A défaut de pouvoir être préemptés, les processus sous TinyOS sont interruptibles pour permettre l'interception des interruptions matérielles.

Le fonctionnement général du système TinyOS le rend très adapté aux applications d'acquisition de données et même de surveillance. Dès sa prise en charge, un événement va donner lieu à une suite de traitements qui ne s'arrêtera que lorsque la réponse souhaitée sera atteinte. Par conséquent, la réactivité du système TinyOS dépend essentiellement du nombre d'événements en attente, à un instant donné, et à l'absence d'erreurs dans l'implémentation des composants.

2.4.3 Les avantages et inconvénients du système TinyOS

Le noyau TinyOS propose une architecture modulaire innovante dédiée au RCSF auquel vient s'ajouter l'utilisation de la notion d'événement en lieu et place de celle plus répandue de processus. Ces éléments font du noyau TinyOS une alternative viable vis-à-vis des systèmes multitâches ce qui représente un avantage. De plus, les différents composants constituant ce système d'exploitation peuvent être ajoutés ou supprimés suivant les contraintes de l'application supportée. Ces composants sont d'autant plus nombreux que le noyau TinyOS bénéficie actuellement du soutien d'une communauté de recherche dynamique et assez importante. Parmi les composants logiciels de haut niveau disponibles, on peut citer le système d'interrogation de données TinyDB [Madden 2005].

A l'inverse, beaucoup de programmes sont développés pour être exécutés sous un système multitâche. Le portage de ces programmes, s'il est possible, n'en demeure pas moins complexe. De la même manière, des programmes développés pour fonctionner sous le noyau TinyOS pourront être difficilement utilisables sous un autre système d'exploitation. Enfin, comme nous le verrons plus en détails dans la partie 2.7, l'utilisation des événements peut poser quelques problèmes en ce qui concerne la réactivité du système d'exploitation.

2.5 Le système d'exploitation AmbientRT

Le système d'exploitation AmbientRT est un autre exemple de système conçu spécifiquement pour les RCSF [Hofmeijer 2005]. Sa conception s'appuie sur deux grands principes :

- un ordonnancement basé sur la politique EDFI (Earliest Deadline First Inheritance)
- une architecture centrée sur les données (« data centric »).

2.5.1 L'ordonnancement dans le système AmbientRT

Le système AmbientRT utilise la politique d'ordonnancement EDFI [Jansen 2003] élaborée à partir des algorithmes EDF (Earliest Deadline First ou « échéance la plus proche d'abord ») [Liu 1973] et DI (Deadline Inhéritance ou « héritage d'échéance ») [Sha 1990]. La priorité d'un processus ou d'une tâche sous le système AmbientRT est calculée ou recalculée à partir de sa date d'échéance. L'intérêt principal de ce fonctionnement est de pouvoir favoriser, à un instant donné, un processus de faible priorité pour qu'il puisse s'exécuter avant la date limite de son échéance.

L'utilisation de ces deux algorithmes d'ordonnancement s'explique par la nécessité d'avoir une politique d'ordonnancement dynamique sans les problématiques d'inversion de priorité (voir Figure 2.13). En effet, l'ordonnancement des processus est établi de manière à ce qu'il n'y ait pas, à un instant donné, de concurrence entre les ressources partagées. L'exclusion mutuelle de ces ressources est effectuée en amont par rapport à d'autres systèmes d'exploitation. Au moment de l'exécution, les risques d'inter-blocages (« dead lock ») sont donc écartés à la source.

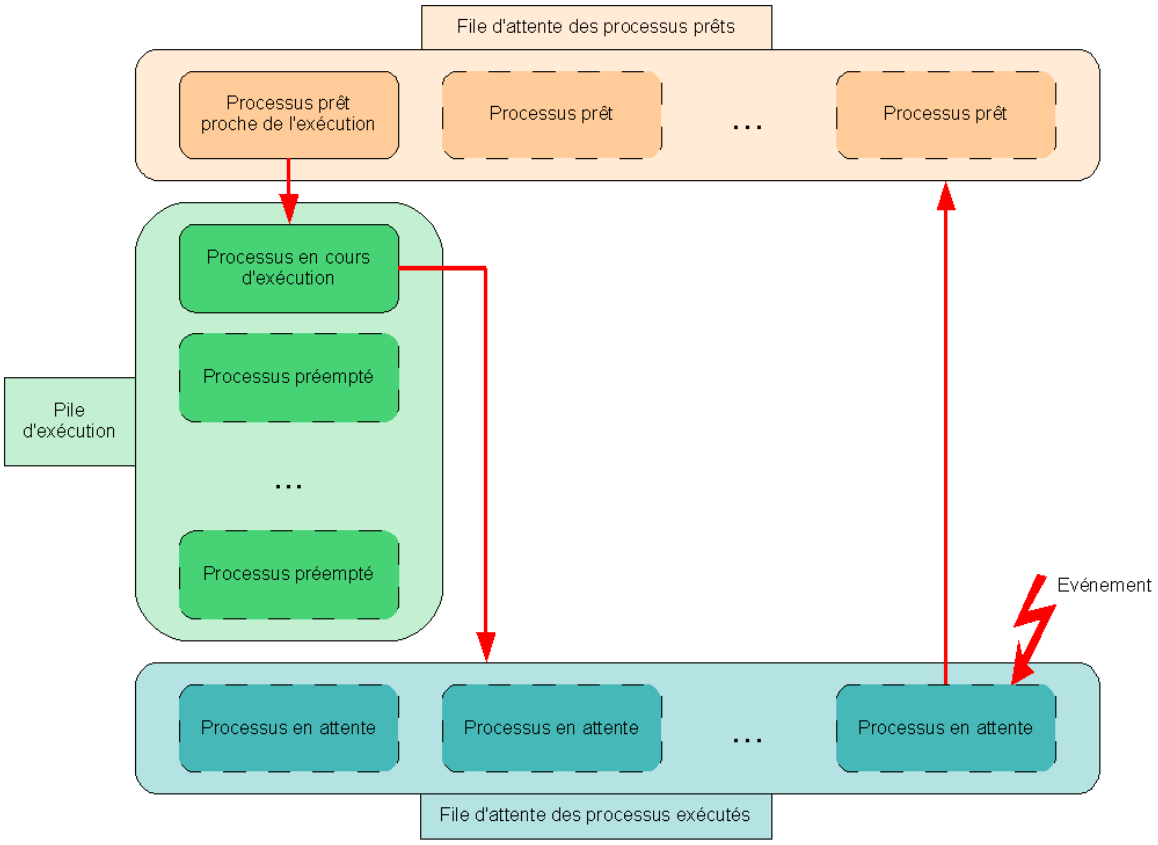

Figure 2.13 – Ordonnancement des processus dans le système AmbientRT [Jansen 2003]

Dans l'algorithme EDFI, l'ordonnancement des tâches s'organise autour de trois structures :

- la file des processus prêts ;
- la pile d'exécution ;
- la file des processus exécutés.

Dans la file des processus prêts, les processus sont triés par date d'échéance, le processus à l'échéance la plus proche étant en tête de file. La date d'échéance de ce processus est comparée avec celle du processus en cours d'exécution. Si le résultat est en faveur (c'est-à-dire date d'échéance plus proche) du processus de la file d'attente, celui-ci devient le processus en cours d'exécution et déplace l'autre processus dans la pile d'exécution en tant que processus préempté. Tous ces traitements sont effectués de manière périodique.

Les contextes des processus de la pile d'exécution sont eux-mêmes stockés sous forme de pile. Quand un processus finit son exécution, le processus suivant se trouve soit dans la file des processus prêts, soit dans la pile d'exécution. Par conséquent, pour passer au processus suivant, on dépile le contexte du processus venant de se terminer et, soit on empile le nouveau contexte, soit on descend d'un niveau dans la pile de stockage des contextes. Le passage au processus suivant en est ainsi simplifié.

A la fin de son exécution, un processus est reversé dans la file d'attente des processus exécutés et attend l'arrivée d'un stimulus ou événement pour devenir un processus prêt. Le diagramme d'états et de transitions des processus du système AmbientRT est donné dans la Figure 2.14.

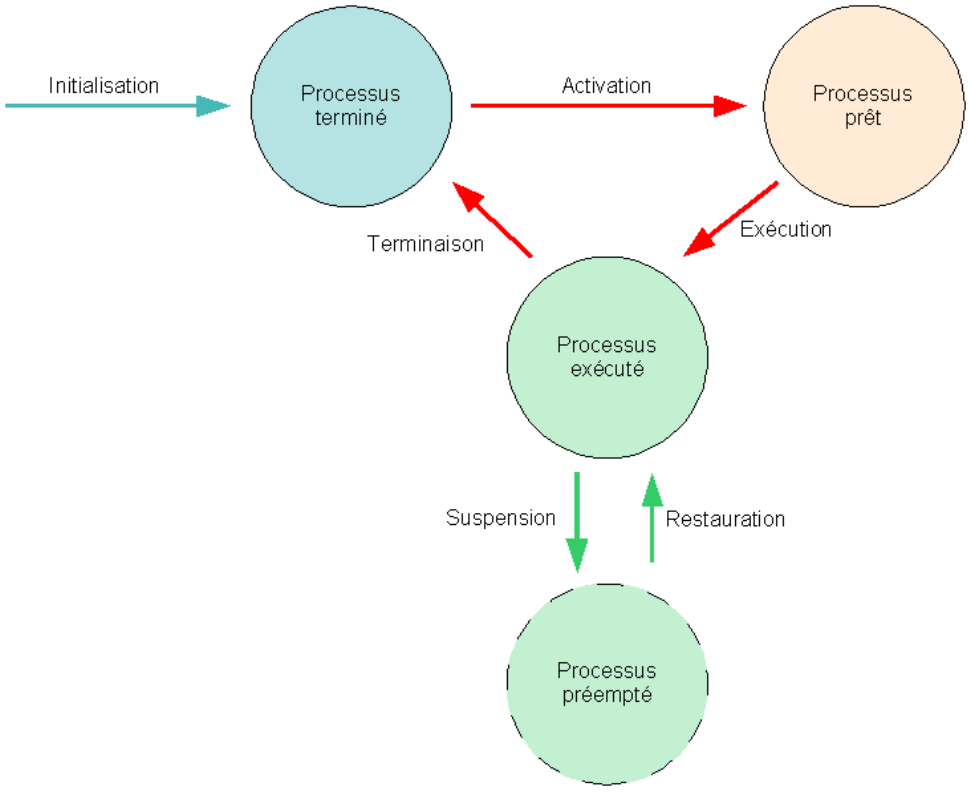

Figure 2.14 – Diagramme d'états et de transitions des processus du système AmbientRT [Hofmeijer 2005]

La description précédente présente la partie relative à l'emploi de l'algorithme EDF. L'algorithme DI intervient, quant à lui, au moment de la comparaison entre le processus prêt à être exécuté et celui en cours d'exécution. Les ressources nécessaires au processus proche de

l'exécution sont analysées et comparées avec l'ensemble des ressources utilisées par les processus présents dans la pile d'exécution. Si une ressource en commun existe, le processus ne passe pas dans la pile d'exécution et ceci même si sa date d'échéance est plus proche.

En termes de structure, les processus du système AmbientRT possèdent un ensemble d'attributs :

- la date d'échéance ;
- la période ;
- le coût processeur ou CPU ;
- la liste des ressources nécessaires à l'exécution.

Si la période est inconnue comme c'est le cas pour les processus apériodiques, le temps minimal possible entre deux exécutions consécutives est choisi. Comme nous venons de le voir, la liste des ressources revendiquées est importante pour établir l'ordonnancement avec exclusion mutuelle.

On peut rajouter à ceux-ci, deux attributs utilisés pour contenir les informations durant l'exécution du processus :

- la date de sélection pour l'ordonnanceur ;
- la date d'échéance effective.

2.5.2 La gestion des composants dans le système AmbientRT

Dans les RCSF, l'organisation en couches croisées des éléments logiciels est généralement favorisée par rapport à celle en couches superposées classiques. Ainsi, des fonctionnalités portant sur l'ensemble du système comme la gestion de l'énergie peuvent être remplies au sein d'un même et unique composant. Une autre illustration de ce point est l'élaboration d'un module de contrôle d'erreurs commun à l'ensemble des couches de la pile de protocole de communication plutôt qu'à une seule.

Cette approche se retrouve dans le système AmbientRT par l'utilisation de composants ou DCE (Data Centric Entities) pour la construction des applications supportées [Dulman 2004]. La notion de « centré sur les données » (ou « data centric ») qui caractérise le système AmbientRT est présente dans la méthode d'assemblage des composants. Un composant peut être représenté par une boîte noire dont les traitements effectués sont déclenchés par l'arrivée de données. Ces traitements vont eux-mêmes produire des données et demander l'accès à d'autres données globales du système. Le processus de production de données est la *fonctionnalité* du composant. Plusieurs composants peuvent avoir la même fonctionnalité et pour pouvoir les différencier, la notion de *capacité* est introduite. Elle correspond au coût associé au composant pour réaliser sa fonctionnalité et intègre le niveau de performance et de qualité observée (voir Figure 2.15).

L'accès aux données est obtenu à l'aide d'un mécanisme de type publication/souscription. On distingue les tâches d'un composant qui vont modifier la donnée et la publier, et celles qui répondent ou souscrivent à ces modifications. Puisque les composants ne vivent qu'à travers les données manipulées, leur politique d'assemblage est, par extension, de type publication/souscription.

Dans ce qui vient d'être mentionné, le terme « donnée » ne se résume pas aux objets en mémoire manipulés par le système. Il inclut également les événements et une « notice » pour permettre leur utilisation par les composants. Les événements considérés sont issus soit de l'environnement observé soit de commandes exécutées par les utilisateurs. Plus généralement, une donnée est composée d'un ensemble d'objets distincts appelé DT (Data

Type). Ces derniers sont de nature variée allant des éléments en mémoire aux interruptions matérielles. Chaque donnée est caractérisée par son nom et son contenu. Des informations comme sa durée de validité peuvent lui être associées. Lors du déclenchement d'une interruption, une donnée DT spécifique est publiée pour être ensuite prise en compte par le composant souscripteur qui en a la charge.

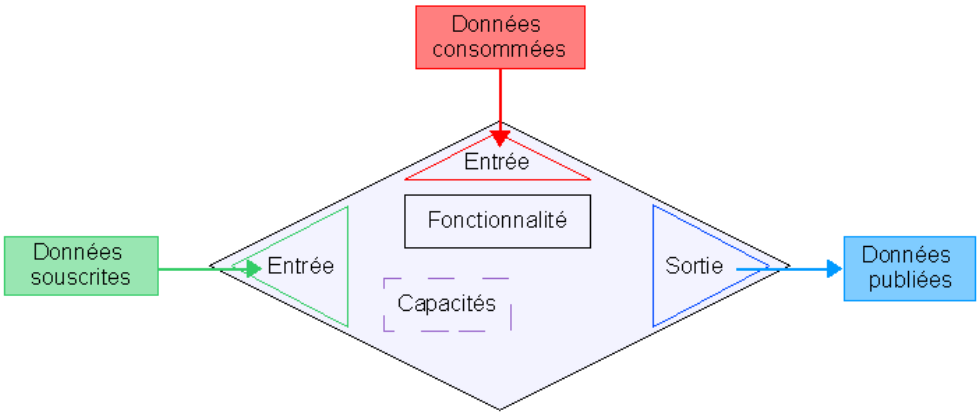

Figure 2.15 – Composant DCE du système AmbientRT [Dulman 2004]

Sous le système AmbientRT, le DCS (Data Centric Scheduler) gère les tâches ou processus qui s'exécutent et les composants (voir Figure 2.16). Il joue donc à la fois le rôle d'un ordonnanceur de tâches et celui d'un gestionnaire des données. Au niveau des composants, il permet d'activer ou de désactiver en direct un composant [Dulman 2004]. Des modules dits DLM (Dynamic Loadable Module) peuvent également être ajoutés dynamiquement après un téléchargement transitant par les liaisons radio ou série.

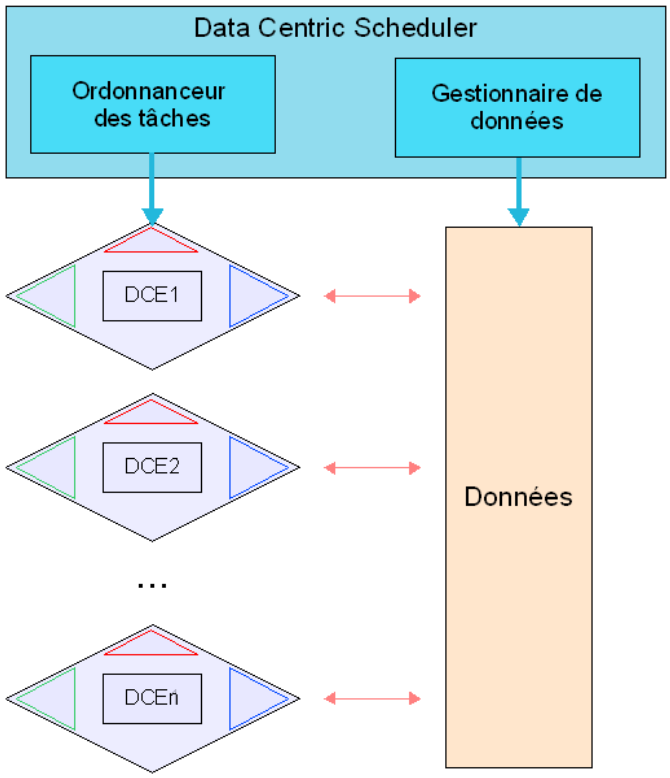

Figure 2.16 – L'architecture à base de composants du système AmbientRT

Au regard de la Figure 2.16, l'organisation du système AmbientRT s'articule autour de la politique d'ordonnancement, présentée dans la section 2.5.1, appliquée aux tâches issues de différents composants qui sont liés entre eux par l'intermédiaire de données de différentes natures (objets en mémoire, événements, interruptions, etc.).

2.5.3 Les avantages et inconvénients du système AmbientRT

Le noyau AmbientRT se situe à l'intersection des systèmes TinyOS et MANTIS. Comme ce dernier, il s'agit d'un système multitâche dont la politique d'ordonnancement est basée sur la date d'échéance des processus. Il possède donc les inconvénients relatifs à ce type de système d'exploitation c'est-à-dire le temps et la mémoire consommés par les changements de contexte.

Comme le système TinyOS, le noyau AmbientRT dispose d'une architecture à base de composants. Mais si les composants du système TinyOS regroupés par catégorie sont pratiquement organisés en couches, ce n'est pas le cas pour le noyau AmbientRT. Celui-ci propose une approche différente centrée sur les données plutôt que les traitements. Les composants sont ainsi reliés entre eux plus par les données qu'ils manipulent que par leurs rôles ou fonctions. Les données produites par un composant seront utilisées en entrée par un autre. Cette architecture est bien adaptée à la mise-à-jour, l'ajout ou la suppression, de manière dynamique, de composants du système. En revanche, cette architecture complexifie le portage de programmes, quel que soit le sens, à partir ou vers d'autres systèmes d'exploitation.

2.6 Les systèmes d'exploitation hybrides

Les avantages et inconvénients des systèmes basés sur les événements et multitâches, qui seront présentés dans le premier paragraphe, ont conduit à la conception de systèmes d'exploitation hybrides. L'un des premiers systèmes hybrides dédié aux RCSF est le système Contiki qui sera étudié dans le second paragraphe.

2.6.1 Intérêt des systèmes d'exploitation hybrides

Dans les sections précédentes, différents types de systèmes d'exploitation dédiés aux réseaux de capteurs ont été présentés. On distingue deux grandes familles de système :

- Les systèmes multitâches ;
- Les systèmes basés sur les événements.

Le système AmbientRT, de par sa politique d'ordonnancement, peut être considéré comme un système multitâche centré sur les données. Chaque type de système dispose d'un certain nombre d'avantages et d'inconvénients (voir Table 2.2)

Type de système	Avantages	Inconvénients
Système multitâche	• Logique de programmation classique (UNIX,…) • Robustesse : tolérant aux fautes d'une ou de plusieurs tâches	• Durée du changement de contexte • Consommation de mémoire par tâche
Système basé sur les événements	• Utilisation de la mémoire • Pas de compétition pour accéder aux ressources critiques • Modularité : construction d'une application par combinaison d'événements	• Pas ou mal adapté aux applications temps-réel et nativement multitâches • Intégration complexe des traitements à longue durée d'exécution

Table 2.2 – Avantages et inconvénients des systèmes multitâches et basés sur les événements

Le principal inconvénient des systèmes multitâches reste l'espace mémoire alloué à chaque processus ou tâche [Levis 2004] et la durée des changements de contexte [Dunkels 2006]. Ainsi, la taille du système dépend énormément des traitements à réaliser et donc de l'application supportée. En ce qui concerne les systèmes basés sur les événements, la programmation d'une application demande au préalable la définition d'une machine d'états [Hill 2000]. Cette dernière peut s'avérer plus ou moins complexe selon la durée des traitements considérés [Dunkels 2006].

Ce constat a amené à la conception de systèmes hybrides combinant les fonctionnalités des systèmes multitâches et des systèmes basés sur les événements afin de bénéficier des avantages de chacun tout en minimisant leurs inconvénients.

2.6.2 Le système d'exploitation Contiki

Avoir à sa disposition un système d'exploitation hybride élargit potentiellement la palette d'applications de RCSF envisageables. Cet aspect se retrouve dans le système Contiki qui s'appuie sur un modèle de fonctionnement hybride [Dunkels 2004].

2.6.2.1 L'architecture hybride du système Contiki

Pour économiser de la mémoire, l'approche basée sur les événements a été, dans un premier temps, privilégiée. Les incertitudes sur la taille de la pile d'exécution et le nombre de processus à prévoir sont ainsi évitées. Cependant, les opérations longues telles que la cryptographie de données s'accordent mal avec l'utilisation des événements par la

monopolisation du système pour une durée conséquente. Pour cette raison, dans un second temps, le système Contiki s'est vu ajouter un composant qui lui permet de fonctionner comme un système multitâche. Ce composant est une bibliothèque de fonctions optionnelle appelée explicitement par le programme qui en a besoin. Cette bibliothèque permet la gestion des processus et de leur pile d'exécution respective.

Le cœur du système Contiki est composé de différents éléments :

- le noyau ;
- le chargeur de programme ;
- les bibliothèques utilisées ;
- la pile de communication avec les pilotes pour le matériel ;
- le module de gestion des bibliothèques systèmes.

L'ajout de programmes configure le noyau pour une application donnée (voir Figure 2-17). Le système ne contient pas de niveau d'abstraction pour l'économie d'énergie mais offre des informations comme la taille de la file d'attente des événements pour que l'application puisse réaliser cette opération.

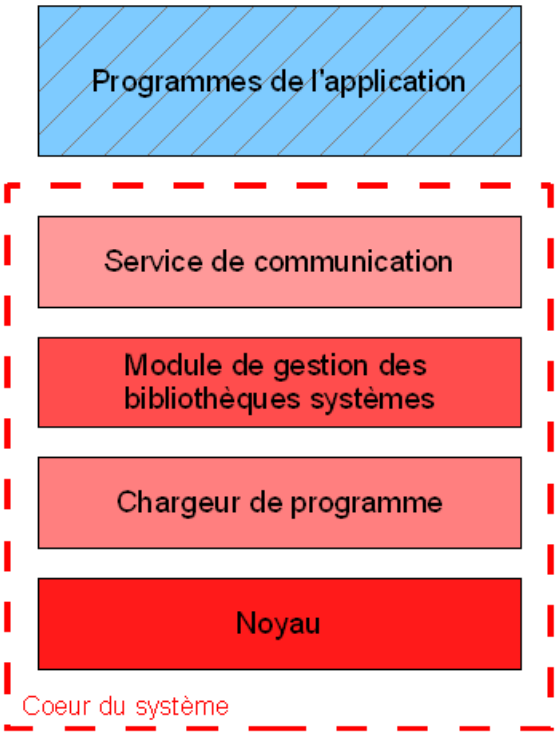

Figure 2.17 – Architecture du système Contiki [Dunkels 2004]

Le noyau gère des événements asynchrones et synchrones. Les premiers sont placés dans une file d'attente après leur appel. Les traitements associés à ce type d'événement sont donc déclenchés après un certain délai. A l'inverse, la réponse à un événement synchrone est quasi immédiate (voir Figure 2.18).

La gestion des interruptions matérielles est différente par rapport à d'autres systèmes. Une interruption matérielle va modifier un drapeau plutôt que d'activer directement un événement. La prise en considération du drapeau par le noyau déclenchera le traitement

souhaité. Le but de cette technique est la gestion de la concurrence entre les interruptions dans leur activation d'événements.

Dans le système Contiki, des fonctions utilisées par les traitements associés à différents événements sont regroupés dans un même *service*. Cette notion est très proche de celle d'une bibliothèque partagée. Les bibliothèques comme les services sont modifiables et remplaçables dynamiquement en cours d'exécution. Ils offrent donc des possibilités de reconfiguration très intéressantes et utiles si l'on considère le cadre d'utilisation des RCSF.

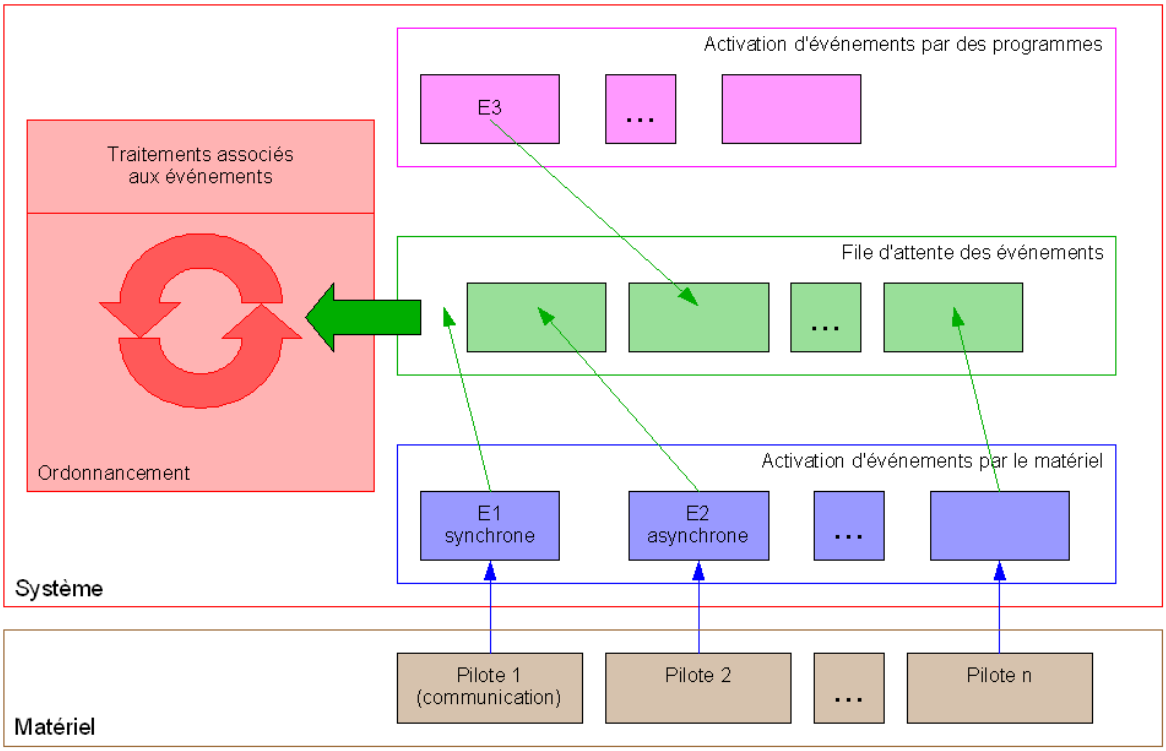

Figure 2.18 – Gestion des événements par le système Contiki

En ce qui concerne les bibliothèques, selon leur emplacement, leur reconfiguration est plus ou moins envisageable. Celles qui font partie du cœur du système, le plus souvent intégrées au module de gestion des bibliothèques, sont considérées comme statiques. Les bibliothèques associées aux programmes de l'application et les services ont vocation à être remplacés dynamiquement. Pour les services, ce remplacement s'effectue de manière transparente pour les programmes ou traitements utilisateurs. Pour éviter les problèmes d'incompatibilité, des contrôles sont réalisés au début du processus de remplacement. La reconfiguration du cœur du système est également possible mais est très complexe à réaliser.

L'un des principaux exemples de service est la communication. Les différents modules ou la pile de communication dans son ensemble sont ainsi reconfigurables ou remplaçables. Une des conséquences est la possibilité d'utiliser plusieurs protocoles de communication pour pouvoir les évaluer au même moment.

Le service de communication établit le lien entre l'application et le matériel à l'aide des pilotes associés. Le noyau du système intervient dans la gestion des événements synchrones activés durant les phases de réception et d'émission d'un paquet ou d'un message. De manière générale, la communication entre services est obtenue par publication d'événements.

Jusqu'à présent, seul le fonctionnement basé sur les événements a été abordé. Logiquement, par rapport au système TinyOS, certains principes sont en commun, d'autres diffèrent. Comme nous l'avons indiqué précédemment, le mode multitâche est assuré par une bibliothèque spécifique. Cette dernière est divisée en deux parties :

- l'ensemble des fonctions permettant l'interfaçage avec le noyau qui gère les événements ;
- l'ensemble des fonctions pour réaliser la préemption des processus avec le mécanisme de changement de contexte associé (gestion de la pile d'exécution de chaque processus).

Ce découpage permet d'équiper le système de sa propre politique d'ordonnancement des processus en modifiant très peu la partie d'interfaçage avec le noyau (voir Figure 2.19).

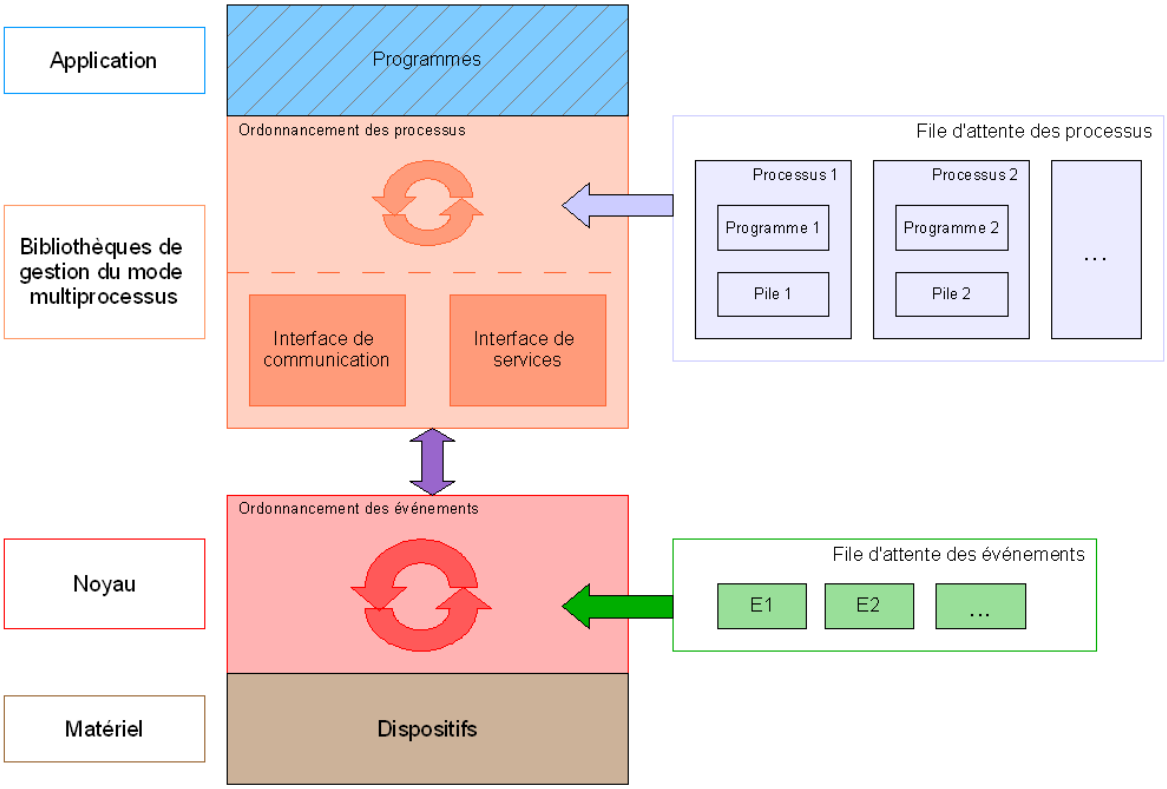

Figure 2.19 – Schématisation du fonctionnement du système Contiki en mode multitâche

Sous le système Contiki, pour simplifier la transformation d'une application en un ensemble d'événements, un nouveau principe dénommé « protoprocessus » (« proto-threads ») a été développé et est également disponible grâce à une bibliothèque [Dunkels 2006]. Les « protoprocessus » sont semblables dans leur fonctionnement aux événements à la différence près, qu'à la manière des processus, ils peuvent être bloqués. Cependant, ce blocage est localisé à des emplacements précis symbolisés par la présence de la primitive PT_WAIT_UNTIL(cond1). Le déblocage se produit quand la condition « cond1 » est vérifiée.

Comme pour les événements, les « protoprocessus » partagent la même pile d'exécution. Le mécanisme de blocage est obtenu au dépend d'un surplus de quelques octets de mémoire par « protoprocessus », équivalent à la quantité nécessaire pour stocker un pointeur. Dans certains cas, les fonctions dans lesquelles les « protoprocessus » sont bloqués, doivent disposer d'un espace mémoire pour stocker l'état de leurs variables locales. Le contexte d'exécution n'est donc plus attaché à chaque « protoprocessus » mais aux fonctions utilisées.

Comme nous avons pu le constater tout au long de cette présentation dédiée au système Contiki, celui-ci offre une architecture hybride novatrice mais qui s'appuie surtout sur un système basé sur les événements. On passe d'un mode de fonctionnement à l'autre sans pouvoir réellement les combiner sauf si l'on utilise les « protoprocessus ». L'objet de la partie suivante est l'étude du système d'exploitation LIMOS (LIghtweight Multithreading Operating System) qui lui est nativement hybride.

2.6.2.2 Les avantages et inconvénients du système Contiki

L'architecture hybride du noyau Contiki autorise deux modes de fonctionnement soit multitâche, soit basé sur les événements. A ce titre, elle permet à ce système d'offrir plusieurs solutions pour répondre au, plus près, aux contraintes de l'application supportée, un mode pouvant être plus performant que l'autre. Cela constitue le principal avantage de ce système d'exploitation.

Cependant, le noyau Contiki reste, nativement, un système d'exploitation basé sur les événements. Pour obtenir le mode multitâche, une bibliothèque doit être installée. Les fonctions associées à cette bibliothèque n'accèdent pas directement à l'ensemble des ressources du capteur sans fil. Elles doivent, dans certains cas, faire appel à la partie du noyau dédié à la gestion des événements. Cette structure à deux niveaux a pour conséquence une dégradation des performances du système quand le mode multitâche est activé.

2.7 Le système d'exploitation LIMOS

Chaque catégorie de systèmes possède des qualités au regard de l'application visée [von Behren 2003] [Dabek 2002]. De plus, tout programme peut être décrit et implémenté sous forme de processus ou d'événements [Lauer 1978]. Le développement du système d'exploitation LIMOS, à l'architecture nativement hybride, découle du souhait de disposer d'un système au large spectre d'utilisation dans le cadre des RCSF. Cette approche a déjà été abordée sous différents angles [Adya 2002] mais pas de la façon que nous allons présentons.

La conception du système LIMOS s'appuie sur le système basé sur les événements TinyOS [Hill 2000] et sur le système multitâche SDREAM (Super-small Distributed REal-time Micro-kernel) [Zhou 2004b]. Le système TinyOS fut l'objet de la section 2.4 pour présenter les systèmes basés sur les événements. Le système SDREAM de Monsieur Hai-Ying Zhou est un système d'exploitation dédié aux applications embarquées mais pas aux RCSF à cause d'une empreinte mémoire trop grande. Ce système ainsi que M-LINDA de Monsieur Eric Yao [YAO 1996], H-LINDA de Monsieur Young-Hwan Park [PARK 1997] et DREAM de Monsieur Christophe de Vaulx [de Vaulx 2003] ont été développés sous la direction du Professeur Kun-Mean Hou et font partie de la famille des systèmes dérivant du système LINDA [Rowstron 1996].

Cette partie présente le cœur du système LIMOS [Zhou 2006b]. Dans le chapitre suivant, des indications seront fournies sur les travaux réalisés sur ce système dans les domaines, entre autres, du positionnement par GPS, de la qualité de service (QdS) dans la communication et le routage [Zhou 2006c] [Chanet 2007].

2.7.1 L'architecture du système

Le système d'exploitation LIMOS a une architecture hybride à deux niveaux : l'un pour les événements, l'autre pour les processus. Cette organisation permet au système LIMOS de fonctionner sur les modes basés sur les événements, multitâche ou hybride et d'offrir la configuration la plus performante possible en minimisant les ressources requises. Cette architecture peut être représentée de manière formelle (voir Figure 2.20).

$LIMOS = \{E1 \rightarrow E2 \rightarrow E3 \rightarrow \ldots \rightarrow En\text{-}1 \rightarrow En\}$	→ opérateur de succession d'événements
où $Ei = \{Pi1 \,//\, Pi2 \,//\, \ldots \,//\, Pim_k\}$ pour $i = 1,\ldots,n$	// opérateur de concurrence de processus

Figure 2.20 – Représentation formelle de l'architecture du système LIMOS

Le système LIMOS a un nombre « n » d'événements E qui se succèdent les uns aux autres suivant un ordre défini par les priorités ou les éléments déclencheurs. Aucun chevauchement n'est possible de par la nature même des événements. Chaque événement regroupe en son sein un nombre variable « m_k » de processus P qui eux sont en concurrence vis-à-vis des ressources systèmes.

Une configuration où chaque événement n'abrite qu'un unique processus correspond à un mode de fonctionnement basé sur les événements pur à la manière de TinyOS. A l'inverse, pour obtenir un système multitâche tel que SDREAM, un seul événement avec plusieurs processus doit être défini (voir Figure 2.21).

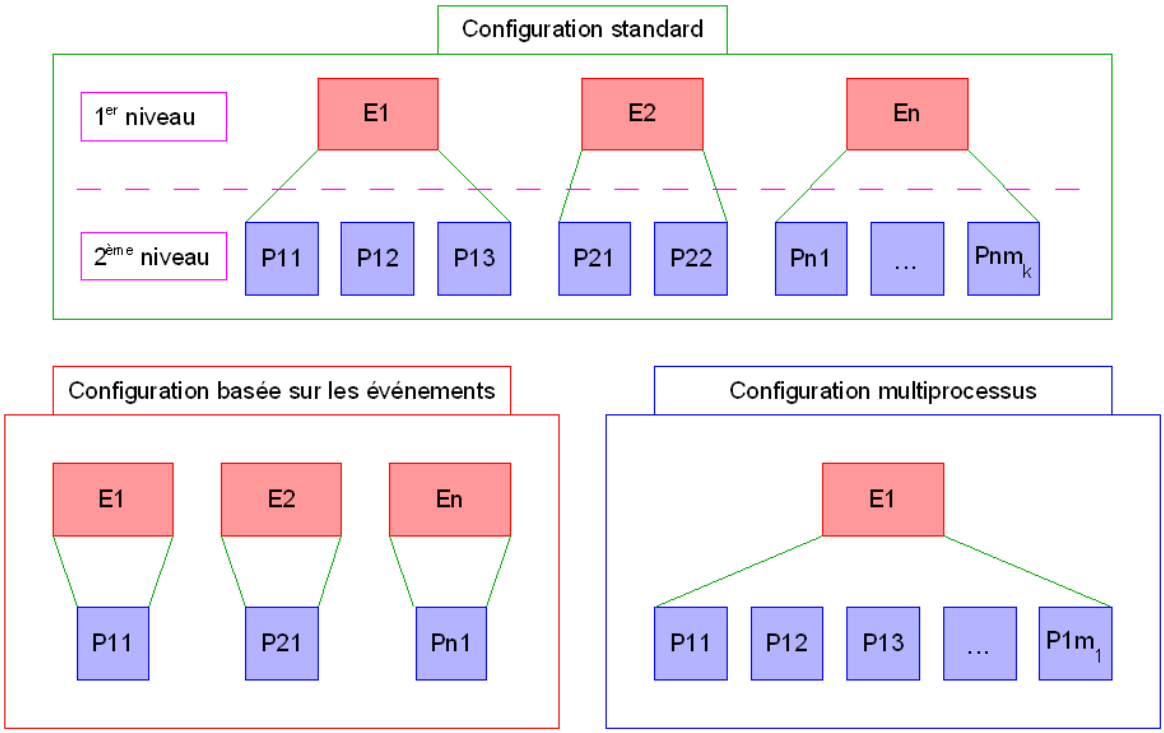

Figure 2.21 – Différentes configurations du système LIMOS

2.7.1.1 Les événements

Au regard de l'architecture du système, l'événement reste a priori le point central. Pour les applications de RCSF d'acquisition de données, les systèmes basés sur les événements comme TinyOS sont, la plupart du temps, les mieux adaptés. Cependant, pour la gestion, entre autres, des traitements de longue durée, chaque événement du système LIMOS peut héberger en son sein un ensemble de processus en concurrence pour l'accès aux ressources. Le terme d'*événement* ne signifie pas uniquement l'élément déclenchant mais comprend l'ensemble des traitements qui en découle. Chaque processus prend sa source dans un programme composé d'une ou plusieurs fonctions dont le rôle est clairement identifié. Les processus sont regroupés pour fournir les traitements venant en réponse à un événement donné.

La configuration du système LIMOS pour une application donnée s'effectue en deux étapes. La première consiste à identifier correctement les événements que l'on peut associer à l'application comme la réception ou la transmission d'un message lors d'une communication sans fil. La complexité de cette étape dépend de l'application [Dunkels 2006]. La seconde est l'association à chaque événement du ou des processus nécessaires à la réalisation des traitements souhaités (voir Figure 2.22).

En règle générale, les événements sont répartis selon deux catégories :

- les événements « matériels » ;
- les événements « logiciels ».

Les événements « matériels » sont en correspondance directe avec les dispositifs matériels gérés par le système. Ces événements viennent en réponse à des interruptions matérielles provenant de temporisateurs programmables (« timer»), des capteurs physiques, des modules de communication, etc. La vitesse de réaction est importante pour éviter, par

exemple, le débordement du buffer de réception de données associé à un module de communication.

Les événements « logiciels » viennent en réponse à des signaux « internes » au système initiés par d'autres événements. Sous le système LIMOS, une application peut être vue comme une succession d'événements dont l'ordre change suivant les situations où la fin d'un événement provoque le début d'un autre.

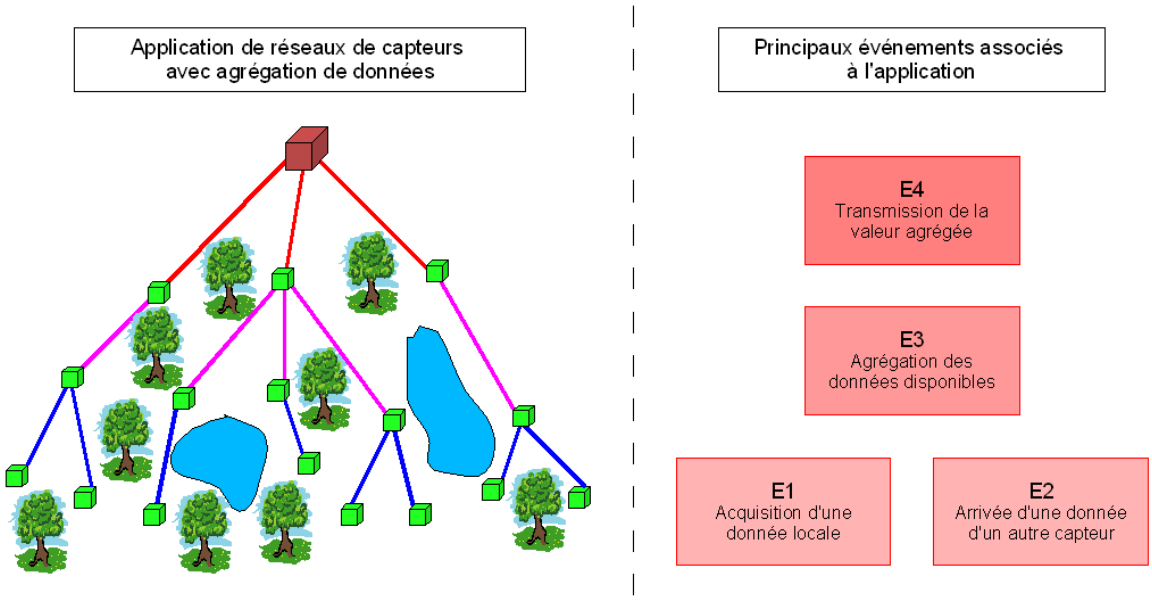

Figure 2.22 – Transformation d'une application en ensemble d'événements

Un autre classement selon la fréquence d'appel est également possible :

- les événements périodiques ;
- les événements apériodiques.

Les événements périodiques sont des événements qui se répètent à intervalles de temps réguliers. Ils correspondent parfaitement aux applications d'acquisition de données dont la fréquence d'échantillonnage est connue. La difficulté réside dans l'estimation précise ou dans un laps de temps borné, de la durée d'exécution de chaque événement afin de prévoir le comportement temporel du système [Bonnet 1999]. Cet aspect est primordial selon que les contraintes de l'application requièrent un système temps réel dur ou lâche.

Les opérations transitoires dont le profil temporel n'est pas clairement établi sont représentées dans le système par l'intermédiaire des événements apériodiques. Selon les cas, des priorités plus importantes sont affectées à l'un ou l'autre type d'événements.

2.7.1.2 Les processus

Les processus du système LIMOS sont très similaires à ceux présents dans les systèmes multitâches classiques et présentés dans le paragraphe 2.1.2.2. Plus précisément, ils dérivent de ceux des systèmes DREAM et SDREAM avec quelques différences. La principale est la suppression des processus périodiques qui ne pouvaient pas être préemptés. Sous le système LIMOS, ils ont été remplacés par les événements périodiques hébergeant un ou plusieurs processus. Par conséquent, on retrouve le même fonctionnement mais intégré dans une architecture plus hiérarchisée. Une description détaillée de leur structure est donnée, entre autre, dans la section suivante.

2.7.2 Les politiques d'ordonnancement

La politique d'ordonnancement des événements et des processus s'effectue en accord avec leurs caractéristiques propres. L'architecture à deux niveaux du système se retrouve dans l'ordonnancement de ces différents éléments. Ainsi, le système est à la fois muni d'un gestionnaire d'événements et d'un ordonnanceur de processus, représentant un des autres aspects novateurs introduits par le système LIMOS.

2.7.2.1 Le gestionnaire d'événements

De la même manière que sous les systèmes TinyOS et Contiki, un événement ne peut pas être préempté par un autre événement. Du moment où les traitements associés à un événement ont débuté, ils se poursuivent jusqu'à leur terminaison. L'unique arrêt autorisé concerne la prise en compte d'une interruption. A l'arrivée d'une interruption, le noyau du système traite celle-ci puis redonne la main de suite à l'événement suspendu. En résumé, un événement est donc interruptible mais ne peut être préempté (voir Figure 2.23). Par conséquent, les différents états d'un événement sont :

- l'état « Inactif » ;
- l'état « Prêt » ;
- l'état « Elu ».

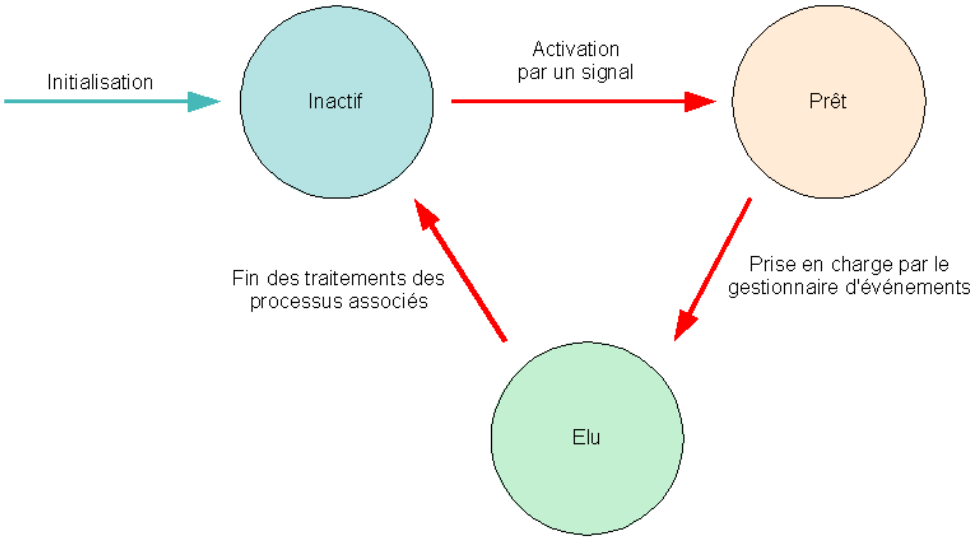

Figure 2.23 – Diagramme d'états et de transitions des événements

Au démarrage du système, l'événement débute dans l'état « Inactif ». Il passe ensuite dans l'état « Prêt » dès l'arrivée d'un signal matérialisé par l'apparition d'un message dans son *tuple* associé. En outre, un événement périodique est synchronisé soit sur la fréquence d'acquisition d'une donnée issue d'un capteur physique soit sur sa fréquence propre gérée par le système. La notion de tuple est très importante dans le système LIMOS et correspond, pour un événement, à un espace mémoire réservé où les messages à son attention sont déposés. Une définition et une présentation plus détaillées sont fournies dans la section 2.7.3.2 dont l'objet est la communication et la synchronisation au sein du système LIMOS. Comme nous allons le voir, les tuples interviennent également dans le choix de l'événement courant à exécuter. Dans l'état « Prêt », l'événement entame une phase d'attente qui se termine par sa prise en charge par le système accompagné d'un passage à l'état « Elu ». A la fin de son exécution, l'événement revient à l'état « Inactif ».

Au-delà de son état, les autres attributs des événements sont :

- son identifiant;
- la liste ou table des processus qui lui sont associés ;
- l'identifiant de son tuple ;
- le nombre de messages en attente dans son tuple ;
- sa priorité.

L'ensemble de ces attributs donne la structure d'un événement telle qu'elle est présente dans le système LIMOS (voir Figure 2.24).

```
/*!
   \struct Event
   define Event structure
*/
struct Event {
        char            event_id;               /* identifiant */
        char            event_state;            /* état */
        struct Thread*  thread_tab[MAXTHREAD];  /* liste des processus associés */
        char            event_tuple;            /* identifiant du tuple */
        char            event_tuple_num ;       /* nombre de messages du tuple */
        char            event_prior;            /* priorité */
};
```

Figure 2.24 – Structure d'un événement dans le système LIMOS

Le gestionnaire des événements du système LIMOS combine un ordonnancement simple sur les priorités et un autre basé sur la politique EDF (Earliest Deadline First). L'ordonnancement obtenu est à 2 niveaux (voir Figure 2.25).

```
/* Ordonnancement des événements */

/* Ordonnancement simple sur les priorités */
Sélection du ou des événements « Prêt » de plus haute priorité

SI égalité entre événements ALORS
        /* Ordonnancement basé sur la politique EDF */
        Sélection du ou des événements avec le plus de messages en attente

        SI égalité du nombre de messages en attente ALORS
                Exécution de l'événement le plus ancien
        SINON
                Exécution de l'événement avec le plus de messages en attente
        FINSI
SINON
        Exécution de l'événement de plus haute priorité
FINSI

Retour au début de la procédure d'ordonnancement des événements
```

Figure 2.25 – Algorithme d'ordonnancement des événements

L'événement suivant dans l'ordre d'exécution est celui qui a la plus grande priorité stockée dans l'attribut « event_prior ». Si plusieurs événements ont la même priorité, celui qui a le plus grand nombre de messages en attente dans son tuple est choisi. En cas de nouvelle égalité au niveau du nombre maximum de messages en attente, l'événement le plus ancien est sélectionné. La dimension d'un tuple étant limitée, il faut éviter que des messages soient perdus du fait qu'un événement n'a pas été pris en charge à temps. La priorité d'un événement est déterminée selon l'importance de celui-ci à l'initialisation du système.

2.7.2.2 L'ordonnanceur des processus

A contrario d'un événement, un processus peut être préempté. Cette propriété se retrouve dans les attributs qui lui sont associés (voir Figure 2.26) :

- son identifiant ;
- son état ;
- l'adresse de la fonction qui lui est associée ;
- l'identifiant de son tuple ;
- l'adresse de début de sa pile ;
- l'adresse courante de sa pile ;
- sa priorité ;
- son quantum de temps d'exécution ;
- l'identifiant de l'événement auquel il appartient.

```
/*!
   \struct Thread
   define Thread structure
*/
struct Thread {
        char    thread_id;              /* identifiant */
        char    thread_state;           /* état */
        char    thread_tuple;           /* identifiant du tuple */
        char    thread_address          /* adresse des traitements associés */
        char    thread_stack_inisp      /* pointeur de début de pile */
        char    thread_stack_cursp      /* pointeur courant de pile */
        char    thread_prior            /* priorité */
        char    thread_quantum          /* quantum de temps */
        char    thread_event            /* identifiant de l'événement */
};
```

Figure 2.26 – Structure d'un processus dans le système LIMOS

L'état « Suspendu » se rajoute aux états d'un événement et correspond soit au cas où le processus est en attente d'une ressource indisponible pour l'instant soit au moment où il est préempté par un autre processus (voir Figure 2.27).

Sous certaines conditions, un processus suspendu peut entrer directement dans l'état « Inactif ». Celui-ci correspond à la fois aux processus non activés et à ceux terminés. Quand les processus d'un événement ont réalisé leur traitement et sont passés dans l'état « Inactif », l'événement a terminé son exécution et devient à son tour « Inactif ». Il est à souligner qu'un événement passe dans l'état « Prêt » quand il a au moins un processus prêt à être exécuter et réciproquement.

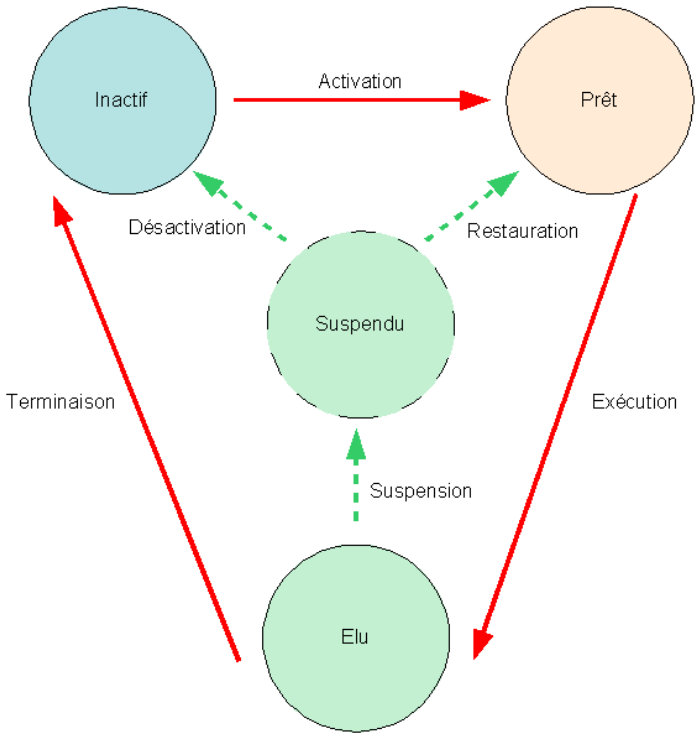

Figure 2.27 – Diagramme d'états et de transitions d'un processus

Les tuples sont de nouveau utilisés comme espace d'échange entre processus. Chaque processus dispose d'un tuple propre où sont postés les données qui le concernent. Une pile est attachée à tout processus. Elle permet de sauvegarder son contexte d'exécution s'il est préempté. Une attention toute particulière doit accompagner la définition des actions et du mode de fonctionnement des processus pour éviter les situations d'inter-blocage. En général, ce type de problème survient quand un processus de forte priorité préempte un autre processus et attend une donnée qui doit lui être fourni par le processus qu'il vient de préempter ou, plus généralement, un processus de plus faible priorité. Une des solutions est d'affecter, dès le départ, une priorité plus faible ou égale au processus consommateur par rapport au processus producteur. Quand le nombre de processus s'échangeant des informations est supérieur à 2, une étude approfondie de leurs interactions est nécessaire. En cas de blocage avéré et persistant, un mécanisme de type chien de garde peut être utilisé pour redémarrer et réinitialiser le système sans toutefois résoudre totalement le problème.

A la base, dans le système LIMOS, les processus sont ordonnancés selon la politique du tourniquet à quantum de temps déjà utilisée au sein des micronoyaux DREAM et SDREAM (voir Figure 2.28).

La sélection d'un processus se fait par comparaison de leur priorité. En cas d'égalité, les processus se partagent les ressources systèmes chacun leur tour pendant une durée de temps fixé (quantum de temps). Une rotation est donc effectuée entre les processus jusqu'à ce qu'ils aient fini leurs traitements (voir Figure 2.29). Le temps alloué à chaque processus est obtenu à partir de son attribut « thread_quantum ». Si la configuration du système spécifie un même quantum de temps pour tous les processus, l'attribut « thread_quantum » peut être omis de la structure des processus. Un processus ne pouvant pas être stoppé brutalement, la préemption se produit dès que celui-ci n'est plus dans une section critique d'exécution.

```
/* Ordonnancement des processus */

/* Ordonnancement simple sur les priorités */
Sélection des processus « Prêt » de plus haute priorité de l'événement courant

SI égalité entre processus ALORS
        /* Utilisation de la politique d'ordonnancement du tourniquet par quantum de temps */
        Exécution alternée durant leur quantum de temps associé des processus à égalité
SINON
        Exécution du processus de plus haute priorité
FINSI

SI plus aucun processus n'est dans l'état « Prêt » ALORS
                Appel au gestionnaire d'événements
SINON
                Retour au début de la procédure d'ordonnancement des processus
FINSI
```

Figure 2.28 – Algorithme d'ordonnancement des processus

Dans l'exemple présenté ci-dessous, si les processus de P1, P2 et P4 ne finissent pas leur exécution avant que le processus P5 ne retourne dans l'état « Prêt », ils seront préemptés par ce dernier. Cette remarque est valable pour le processus P3 pouvant être préempté à la fois par le processus P5 mais également par P1, P2 et P4.

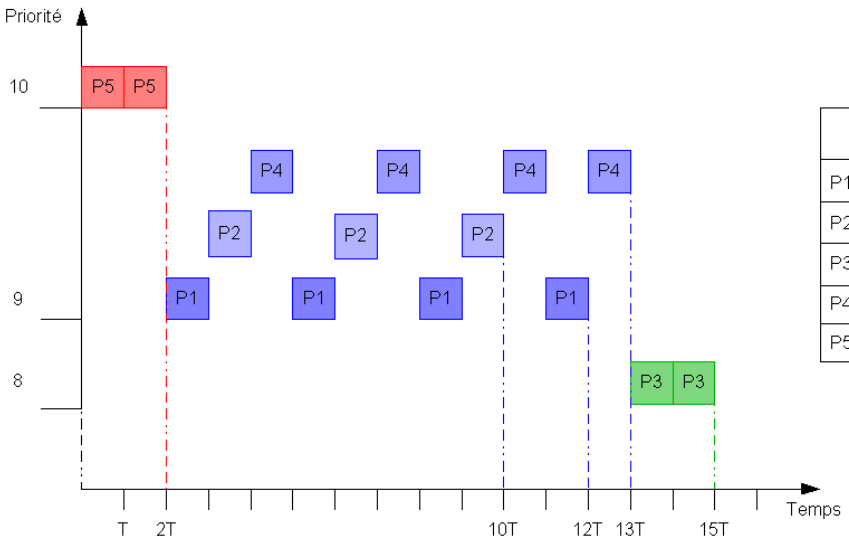

Table des processus

	Priorité	Durée	Quantum de temps
P1	9	4	1T
P2	9	3	1T
P3	8	2	1T
P4	9	4	1T
P5	10	2	1T

Figure 2.29 – Illustration de l'ordonnancement de processus sous le système LIMOS

2.7.2.3 L'ordonnancement dans le système LIMOS

La déclaration d'un certain nombre d'événements et de processus fait partie de l'étape d'initialisation du noyau LIMOS (voir Figure 2.30). Ces derniers ne peuvent pas être créés

dynamiquement pour garantir le déterminisme du système vis-à-vis des exigences liées à l'application temps réel supportée.

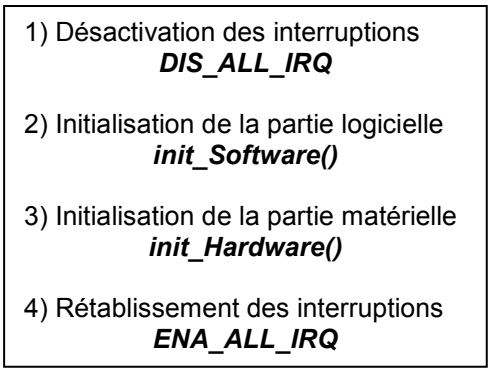

Figure 2.30 – Initialisation du système LIMOS

La création des tuples, des processus, des événements ainsi que leurs relations est réalisée au sein de la fonction *init_Software()*. Les éléments matériels à initialiser sont la gestion des Entrées/Sorties, les différents ports (série par exemple), le convertisseur analogique-numérique et le temporisateur programmable (« timer ») (Voir Figure 2.31).

Au regard de l'architecture à deux niveaux du système LIMOS, le gestionnaire d'événements est le premier en action et détermine l'événement courant. L'ordonnanceur de processus prend le relai pour mettre à disposition les ressources systèmes aux différents processus de l'événement courant.

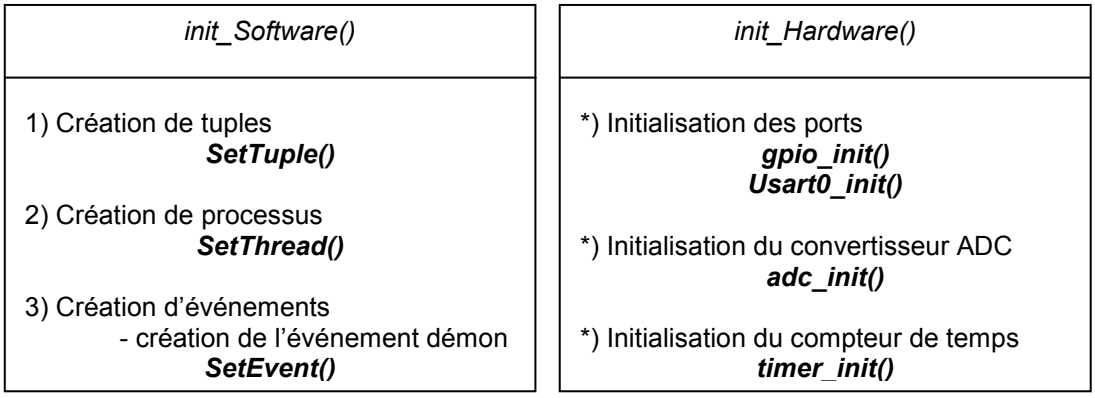

Figure 2.31 – Rôles des fonctions « init_Software() » et « init_Hardware() »

Quand tous les processus ont réalisé leurs traitements, le gestionnaire reprend la main pour sélectionner l'événement suivant (voir Figure 2.32). Les événements E1 et E2 ont la même priorité. Par conséquent, le premier à être pris en charge par le système est déterminé en se basant sur le nombre de messages en attente dans les tuples respectifs. En cas de nouvelle égalité, l'événement le plus ancien dans la file d'attente est choisi. L'événement E1 possède 5 processus dont 3 avec la même priorité qui sont donc exécutés alternativement durant leur quantum de temps associé. Le même principe est appliqué aux processus P31 et P32 de l'événement E3. Les processus de l'événement E2 ayant tous des priorités différentes, ils s'exécutent normalement les uns après les autres. L'événement démon (« daemon ») est activé lorsqu'aucun autre événement n'a besoin des ressources du système. Sa priorité est la plus basse du système c'est-à-dire 0, la plus haute étant fixée arbitrairement à 10. Si le noyau

reste inutilisé pendant un laps de temps assez long, l'événement démon sera en charge de la mise en veille de certains, voire de la totalité des composants du système, afin de diminuer l'énergie consommée. La mise en place de ce genre de mécanisme ne peut se faire qu'en s'appuyant sur une architecture matérielle adéquate.

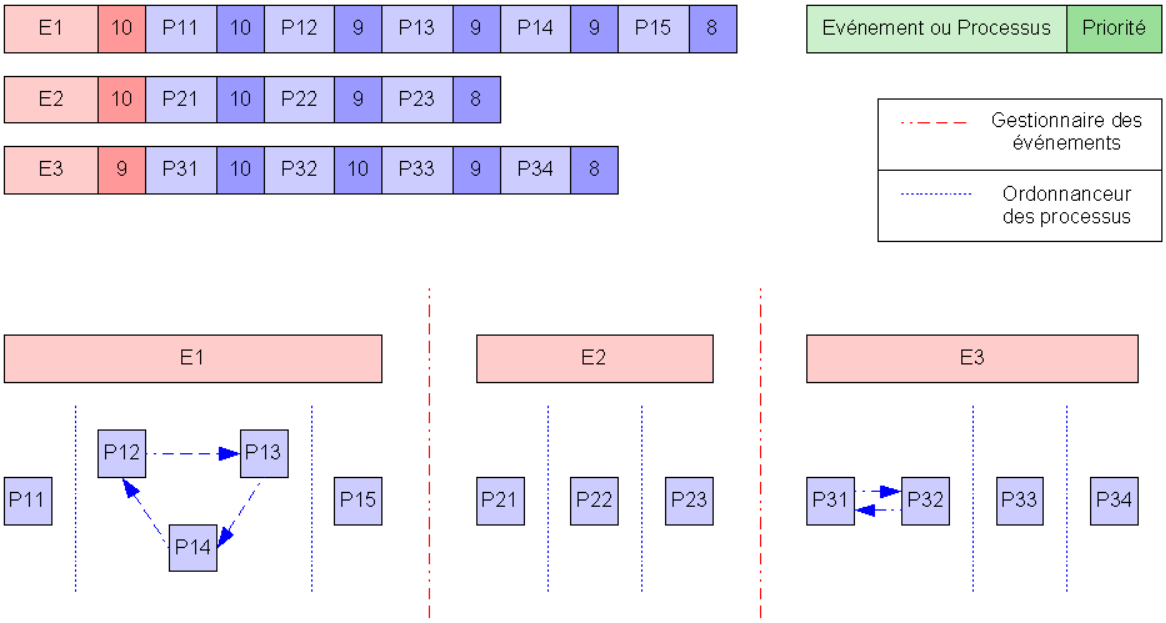

Figure 2.32 – L'ordonnancement des événements et des processus

2.7.3 La communication et la synchronisation

La communication et la synchronisation entre événements et processus du système LIMOS reposent sur l'utilisation des tuples qui est un principe tiré du langage LINDA [Gelernter 1985] [Ahuja 1986] [Rowstron 1996]. Ce langage est dédié à la programmation parallèle et introduit des concepts appropriés à la réalisation d'applications distribuées sur des architectures multiprocesseurs hébergeant une multitude de processus en concurrence.

La notion d'espace de tuples est l'élément central qui, par son mode de fonctionnement, autorise une communication entre les objets (processus ou événement) avec :

- découplage spatial ;
- découplage temporel.

Un objet dispose d'un tuple propre stocké localement ou dans un endroit distant (un autre microcontrôleur). Pour communiquer avec cet objet, les autres n'ont pas à connaître son adresse ou sa localisation, ils ont uniquement besoin d'identifier son tuple d'où la notion de découplage spatial. Parfois, ils doivent connaître le microcontrôleur auquel le tuple est rattaché [Park 1997]. L'identification d'un tuple est obtenue par comparaison du contenu ou, plus précisément, par appariement des champs du tuple.

Le découplage temporel est basé sur le fonctionnement de l'accès au tuple et permet la communication asynchrone et la synchronisation. Pour le mode asynchrone, dès qu'un message est placé dans le tuple d'un objet, les autres pourront y accéder (le lire) par la suite soit une seule fois pour l'ensemble des objets, soit indéfiniment jusqu'à ce que le message soit remplacé par un autre [Park 1997]. Le producteur du message est libre d'effectuer d'autres traitements immédiatement après son dépôt. Deux objets voulant se synchroniser vont se mettre d'accord sur un tuple « rendez-vous » vide au départ. Ce dernier est, la plupart du temps, associé à l'un des deux objets. L'un des objets, le consommateur, va effectuer une

lecture sur ce tuple et donc attendre l'arrivée d'un message. Le placement d'un message dans le tuple par le producteur va permettre la synchronisation entre les deux objets.

Les principes que nous venons d'énoncer ont déjà été repris de manière simplifiée dans les systèmes H-LINDA, M-LINDA, DREAM et SDREAM. Les simplifications apportées ont permis d'adapter ceux-ci à des systèmes s'exécutant sur des architectures matérielles aux ressources limitées. La principale est l'identification d'un tuple à l'aide d'un unique identifiant appelé la *clé*.

Dans le système LIMOS, les concepts de LINDA ont été adaptés pour prendre en considération l'architecture hybride événement/processus. La structure des tuples des événements est identique à celle des processus (voir Figure 2.33). Leur principale caractéristique est l'utilisation d'une file circulaire pour le stockage des messages. Des attributs d'adresses de début et de fin de file sont donc disponibles pour délimiter celle-ci en mémoire. Pour gérer les messages dans la file, deux pointeurs sont utilisés. L'un indique quel est le message courant c'est-à-dire le premier devant être lu. L'autre pointe sur le premier emplacement disponible pour l'insertion ou l'écriture d'un nouveau message.

Les autres attributs des tuples sont :

- son identifiant ou sa clé ;
- son état ;
- sa taille ;
- le nombre de messages stockés.

```
/*!
   \struct Tuple
   define Tuple structure
*/
struct Tuple {
        char            tuple_id;       /* identifiant */
        char            tuple_state     /* état */
        unsigned char   tuple_len;      /* taille */
        unsigned long   tuple_stadr     /* adresse de début de file */
        unsigned long   tuple_endadr    /* adresse de fin de file */
        unsigned char*  tuple_wptr      /* pointeur pour l'écriture de messages */
        unsigned char*  tuple_rptr      /* pointeur pour la lecture de messages */
        char            tuple_num       /* nombre de messages stockés */
};
```

Figure 2.33 – Structure d'un tuple du système LIMOS

Les 2 états d'un tuple sont « Libre » pour aucun message présent ou « Utilisé » dans le cas contraire. Les tuples permettent la communication entre les différents objets définis au sein du système LIMOS avec le respect de certaines règles. La première est qu'une communication entre événements ou processus entre eux est possible, mais pas d'événement à processus, ni dans l'autre sens. De plus, la communication interprocessus n'est autorisée qu'entre processus appartenant au même événement. Les différents objets peuvent écrire dans autant de tuples qu'ils souhaitent mais ne peuvent lire que leur propre tuple. Enfin, il ne s'agit pas d'une règle mais plutôt d'un rappel : tout message écrit dans un tuple par un événement n'est accessible qu'à la fin de l'exécution de celui-ci.

Les opérations de lecture et d'écriture sur les tuples sont réalisées à l'aide de 2 primitives (voir Figure 2.34) :

- la fonction In() ;
- la fonction Out().

L'extraction (lecture) d'un tuple est réalisée à l'aide de la primitive In(). Son fonctionnement diffère selon qu'il s'agit d'événement ou de processus. Quand un processus accède à un tuple sans message par la primitive In(), il est « Suspendu » et préempté jusqu'à l'arrivée d'un message et de son tour d'exécution. A l'inverse, pour éviter le blocage du système, la primitive In() interdit l'accès des événements à un tuple sans message. Pour un événement confronté à ce problème, les deux solutions envisageables dépendent de l'importance du message normalement attendu dans le tuple et se résument soit à la poursuite des traitements soit à la fin de l'exécution. Dans ce dernier cas, l'événement retourne dans l'état « Inactif » et attend de nouveau un signal correspondant à l'arrivée du message souhaité.

La fonction Out() permet le stockage de données ou l'écriture de messages dans le tuple. Sa principale particularité est d'être non bloquante. L'écriture d'un message entraîne, dans certains cas, la préemption du processus courant par un autre processus de priorité plus grande et en attente de l'information déposée. Cette écriture s'accompagne parfois de la modification des attributs du tuple et parfois d'un ré-ordonnancement des événements en attente d'être pris en charge par le noyau.

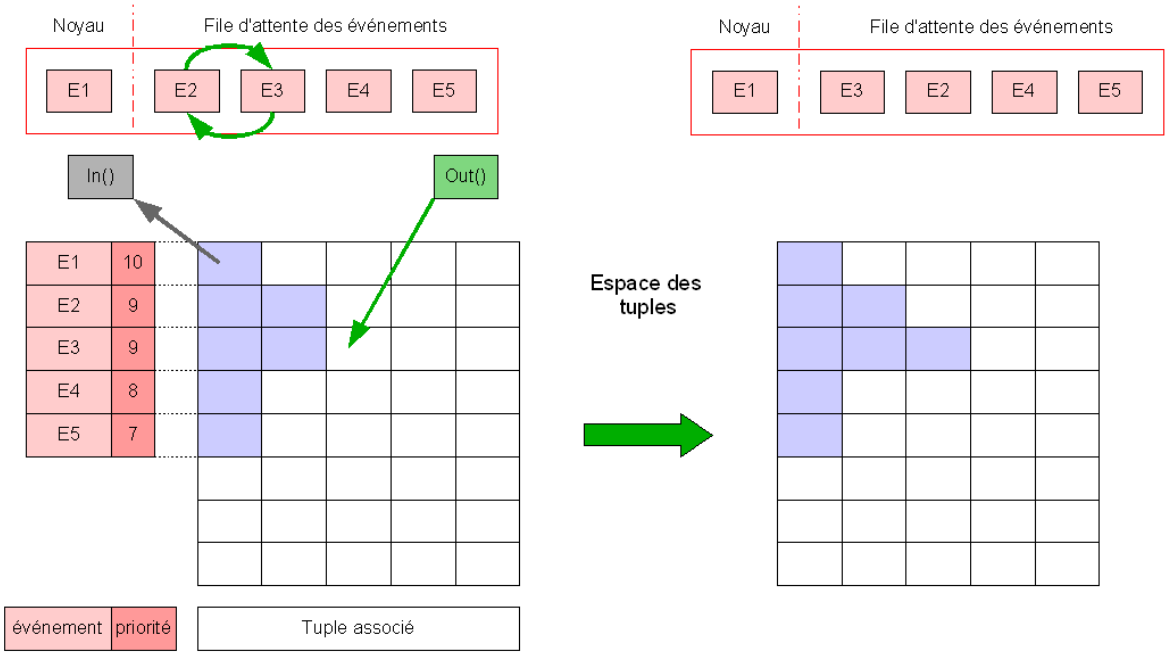

Figure 2.34 – Fonctionnement des primitives In() et Out()

2.7.4 La gestion des périphériques

Les interruptions matérielles sont primordiales dans la gestion des périphériques par le système. Les tuples sont encore utilisés pour faire le lien entre une interruption matérielle et un événement (voir Figure 2.35). L'occurrence d'une interruption est suivie de l'écriture d'un message dans le tuple de l'événement associé. On parle dans ce cas de *signal* plutôt que de message. Un événement est au maximum lié à une interruption.

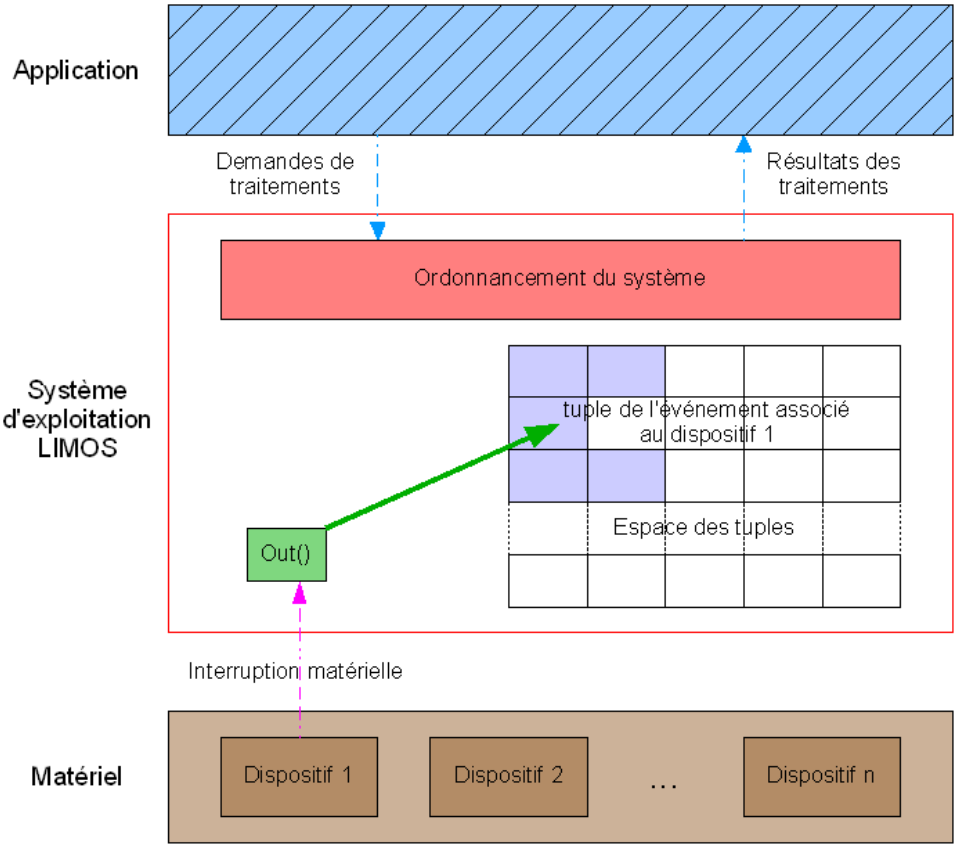

Figure 2.35 – Illustration de l'association interruption matérielle/événement

2.7.5 L'évaluation du système LIMOS

Pour répondre aux contraintes associées aux RCSF, un système d'exploitation doit posséder une faible empreinte mémoire adaptée aux fonctionnements des RCSF. Par conséquent, l'évaluation du système LIMOS va, dans un premier temps, consister à comparer sa taille mémoire avec celle des autres systèmes d'exploitation dédiés aux RCSF présentés dans ce chapitre.

Le système LIMOS a été conçu à partir des noyaux SDREAM et TinyOS. En comparant ces 3 systèmes, il est intéressant de voir où se situe le système LIMOS en terme de performance pure. Cette évaluation permettra de déterminer les autres apports de l'utilisation d'une architecture hybride au-delà d'une forte modularité du noyau.

2.7.5.1 L'empreinte mémoire

Dans cette section, l'empreinte mémoire du noyau LIMOS va être comparée à celle des différents systèmes d'exploitation dédiés aux RCSF présentés dans ce chapitre :

- Le système multitâche MANTIS ;
- Le système basé sur les événements TinyOS ;
- Le système centré sur les données AmbientRT ;
- Le système hybride Contiki.

Des informations plus ou moins détaillées selon les systèmes sont fournies sur la taille de leurs principaux composants et permettent d'introduire les configurations choisies lors de la comparaison (voir Table 2.3).

La partie gestion de processus du système MANTIS consomme 144 octets pour l'ordonnanceur, 120 octets pour une table de 12 processus à laquelle il faut ajouter 10 octets par entrée. Le système MANTIS offre la possibilité de configurer la taille de la pile associée à un processus avec une valeur minimale de 128 octets recommandée. Pour l'accès aux ressources partagées, chaque sémaphore utilise 5 octets.

La partie communication est un autre élément consommateur de mémoire. La taille des messages envoyés est variable et peut atteindre les 64 octets. Par conséquent, chaque buffer de communication doit avoir une taille supérieure à 64 octets qui est, en réalité, de 67 octets. Dans la configuration de base, 5 buffers sont réservés pour un total de 335 octets. L'entité COMM comprenant les couches physiques, MAC et, les interfaces radio, liaison série et de bouclage (« loopback ») représente 64 octets. Enfin, un module pour l'utilisation de l'inondation simple (« flooding ») ou bornée (« Broadcast ») de 30 octets peut être ajouté [Abrach 2003]. La taille du système varie de 831 octets pour 1 processus à 2349 octets pour 12 processus.

A la base, le système TinyOS a une empreinte mémoire de 478 octets répartie entre 432 octets pour le code et 46 pour les données. Le module de communication implique un ajout d'un peu moins de 1,5Koctets. Une configuration complète représente environ 3Koctets pour le code et 230 octets pour les données [Hill 2000].

Le système AmbientRT utilise 3800 octets soit environ 3.7Koctets de mémoire Flash pour une configuration comprenant l'ordonnanceur, le gestionnaire de données, un module d'allocation dynamique de mémoire et une couche d'abstraction du matériel. En terme de mémoire RAM, le système nécessite un minimum de 58 octets dont 26 octets sont dédiés à l'utilisation de la pile. De plus, chaque tâche requiert un supplément de 10 octets [Dulman 2004].

Le système d'exploitation Contiki a été implémenté sur deux architectures : l'une à base d'un microcontrôleur MSP430 de la société Texas Instruments et l'autre s'appuyant sur microcontrôleur AVR de la société Atmel Corporation. Le noyau du système a une taille de 810 et 1044 octets respectivement pour le MSP430 [MSP 2008] et l'AVR [AVR 2008]. Un système complet incluant la bibliothèque de gestion des processus a une empreinte mémoire de 1562 octets pour le MSP430 et de 1940 octets pour l'AVR mais, dans ce cas, sans le chargeur de programme. Pour une configuration hébergeant une application simple, la taille du système Contiki avoisine les 3800 octets [Dunkels 2004].

Pour le système LIMOS, nous nous sommes arrêtés sur une configuration complète de 3 événements, 5 processus et 8 tuples. L'empreinte mémoire de l'ensemble du code est de 3752 octets et 2228 octets pour les données.

Alors que tous les systèmes ont des empreintes mémoires comparables autour des 3,5Ko, le noyau MANTIS en utilise, en théorie, un tiers de moins. Toutefois, pour certaines applications, le système MANTIS a semblé plus volumineux que TinyOS [Dunkels 2004] [Wanner 2005] [Duffy 2006].

	LIMOS	MANTIS	TinyOS	AmbientRT	Contiki
Empreinte mémoire du code	3752 octets	2349 octets + 5 octets par sémaphore	~3Ko	3800 octets + 10 octets par processus	~3800 octets
Configuration	-5 événements -5 processus -8 tuples	-12 processus	-	-	-Sans application

Table 2.3 – Taille mémoire de différents systèmes dédiés aux réseaux de capteurs sans fil

En conclusion, le fait que le noyau LIMOS dispose d'une organisation structurelle plus complexe de par son architecture hybride ne le pénalise pas au niveau de la mémoire consommée par rapport aux autres systèmes dédiés aux RCSF.

2.7.5.2 Les performances générales du système

Le système LIMOS a été implémenté sur un microcontrôleur Atmel ARM7TDMI AT91SAM7S256 [ARM7 2007] dont les principales caractéristiques sont (voir Figure 2.36) :

- 1 microprocesseur de type RISC 32 bits de fréquence maximale de 55 MHz pour 0,9MIPS/MHz ;
- 2 ports USART ;
- 1 port USB 2.0 ;
- 1 interface SPI ;
- 1 convertisseur analogique-digital comportant 8 canaux (« chanel »), chaque canal ayant une précision de 10 bits ;
- 64Ko de mémoire SRAM ;
- 256Ko de mémoire Flash répartie dans 1024 pages de 256 octets.

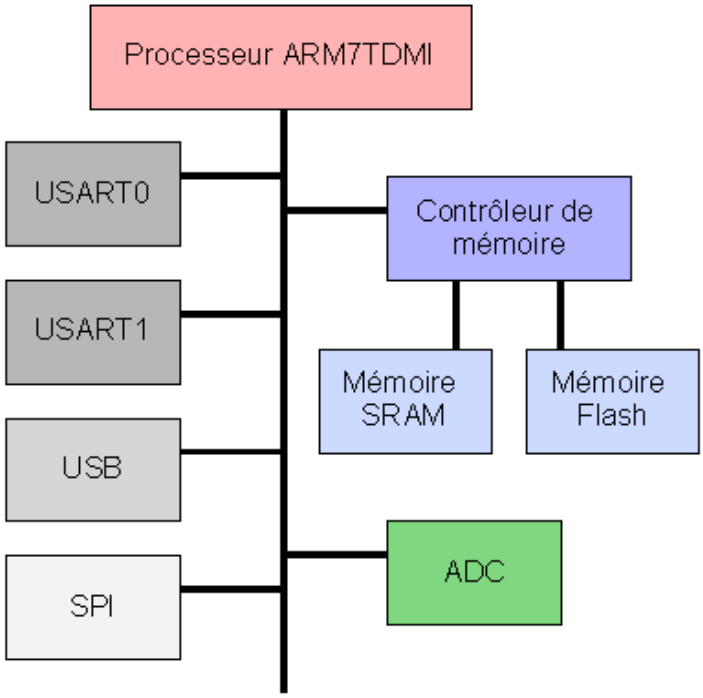

Figure 2.36 – Architecture du microcontrôleur AT91SAM7S256 d'Atmel Corporation

Au niveau de ses caractéristiques, ce microcontrôleur pourrait héberger un système d'exploitation embarqué temps réel classique mais l'utilisation qui en est faite est volontairement orientée vers les RCSF. Le choix s'est porté vers ce microcontrôleur de par les interfaces disponibles (USART, SPI, USB) et de par la possibilité de moduler la fréquence de fonctionnement du processeur de 5MHz à 55MHz. Cette dernière caractéristique permet d'adapter l'énergie consommée par le processeur en fonction de l'application

Le système LIMOS a été élaboré à partir des noyaux SDREAM et TinyOS. Comparer ces 3 systèmes permet de déterminer la place en termes de performance de l'utilisation d'une architecture hybride. Cette dernière autorise une plus grande flexibilité et adaptabilité aux applications au détriment de traitements plus complexes au niveau de l'ordonnancement.

Partant de ce constat, une comparaison par couple système/architecture de capteurs a été choisie plutôt que sur des systèmes seuls (voir Table 2.4).

	LIMOS	**SDREAM**	**TinyOS**
Ordonnancement	ARM7TDMI	MSP430F149	ATmega128
	Temps (en µs)	Temps (en µs)	Temps (en µs)
Evénements	13,86		11,5
Processus	15,72	14	

Table 2.4 – Comparaison entre les systèmes LIMOS, SDREAM et TinyOS

Le noyau TinyOS a été installé dans des capteurs de type « Mica Motes » abritant un microcontrôleur ATmega128. Le noyau SDREAM a équipé des capteurs évolués dédiés à la télémédecine [Zhou 2004a] utilisant un microcontrôleur MSP430F149. Les capteurs sans fil « LiveNodes » qui seront présentés plus en détails dans le chapitre 4 utilisent des microcontrôleurs AT91SAM7S256 de la société Atmel Corporation contrôlés par le système LIMOS.

Cette évaluation porte sur les temps moyen d'exécution au niveau de l'ordonnancement. Ainsi, le gestionnaire d'événements et l'ordonnanceur de processus du système d'exploitation LIMOS sont respectivement comparées avec ceux des noyaux TinyOS et SDREAM. Les performances affichées par le système LIMOS sont très proches de celles des deux autres systèmes.

2.8 Une synthèse sur les systèmes d'exploitation

Le choix du système d'exploitation qui doit équiper un capteur sans fil est une étape importante. La première idée nous amène logiquement vers les systèmes d'exploitation embarqués *classiques* tels que Windows CE .NET, RT Linux, QNX … Malheureusement, ces systèmes ne donnent pas totalement satisfaction en raison, entre autres, d'une empreinte mémoire supérieure à 5 voire 10 Ko.

Par conséquent, de nouveaux systèmes d'exploitation ont été élaborés pour être dédiés aux RCSF. Ces systèmes sont généralement répartis en deux catégories :

- Les systèmes multitâches;
- Les systèmes basés sur les événements.

Les noyaux MANTIS et AmbientRT sont des exemples de systèmes multitâches. Pour la seconde catégorie, on peut citer les systèmes TinyOS et Contiki, avec une particularité pour ce dernier, l'existence d'une bibliothèque système permettant un fonctionnement en mode multitâche. Il représente l'un des premiers prototypes de systèmes dit hybrides.

Sur le même principe, le système LIMOS est doté nativement d'une architecture hybride, basée à la fois sur les événements et sur les processus. Il a été conçu à partir des noyaux SDREAM et TinyOS. L'objectif recherché est la création d'un système capable d'une gestion intelligente des ressources (« resource awarness »). En effet, l'architecture hybride permet de combiner certains avantages de chacune des 2 catégories de systèmes à savoir une approche économe dans la consommation des ressources (mémoire essentiellement) tout en autorisant l'implémentation de traitements de longue durée. A ce titre, la gestion de la mémoire n'a pas été abordée jusqu'à présent. Elle le sera dans le chapitre suivant dont l'objet est la présentation d'un système de fichiers interrogatif LiveFile.

Enfin, le système LIMOS constitue également une base à d'autres développements en rapport avec les RCSF dans les domaines de la localisation [Kara 2007] et de la qualité de service (QdS) pour la communication [Amamra 2004] [Amamra 2008] et qui seront introduits dans les chapitres 3 et 4.

Chapitre 3

Le système de fichiers interrogatif LiveFile

Au sein des réseaux de capteurs sans fil (RCSF), une gestion intelligente des données est importante pour, d'une part préserver les ressources critiques des capteurs sans fil et, d'autre part offrir le comportement souhaité pour l'application supportée. A l'intérieur de chaque capteur et au sein du réseau, des données de divers types et en différentes quantités transitent quasiment en permanence. Elles proviennent des capteurs sans fil, fournissent des informations au système d'exploitation, donnent des indications pour permettre la communication au sein du réseau. Certaines données sont temporaires, d'autres, en revanche, demandent un stockage non volatile pour pouvoir être récupérées même après une panne système ou d'énergie.

Dans le chapitre précédent, le système d'exploitation LIMOS a été présenté sans aborder la problématique de la gestion de la mémoire et des données. Cette tâche est à la charge du système de fichiers LiveFile (LIMOS Versatile Embedded File System). Les principales fonctionnalités de ce système de fichiers interrogatifs vont être, tout d'abord, présentées. Puis, le module dédié à l'interrogation des données sera décrit en détails. Ensuite, la phase de déploiement fera l'objet d'une partie qui servira d'introduction aux applications du chapitre 4. Enfin, les principaux éléments à retenir constitueront la synthèse de fin de chapitre.

3.1 Présentation du système de fichiers LiveFile

Au départ, le rôle du système LiveFile était d'offrir un gestionnaire de mémoire évolué pour le système d'exploitation LIMOS. Dans un RCSF, le stockage des données n'est qu'une première étape avant la transmission de celles-ci vers un poste distant, la station centrale de collecte (« sink node »). Les besoins observés dans différentes applications ont permis d'établir les principales fonctionnalités dont devait disposer un système de fichiers dédié aux RCSF comme LiveFile.

3.1.1 Les fonctionnalités du système de fichiers LiveFile

La mémoire est une ressource critique des capteurs sans fil au même titre que l'énergie et la puissance de calcul. La première utilisation de la mémoire est plutôt classique et se résume au stockage des variables générées et brassées au sein du système d'exploitation. Comme mentionné dans le chapitre précédent, la gestion de la mémoire peut impliquer des opérations complexes comme, par exemple, l'allocation dynamique.

La seconde est relative à la communication et à la nécessité de mettre à disposition des modules associés des buffers pour stocker les données reçues ou à transmettre. En outre, la communication est l'une des fonctionnalités illustrant le lien qui existe entre les différentes ressources des capteurs sans fil et de l'équilibre qu'il est indispensable de trouver dans leur gestion et dans leur consommation. En effet, l'énergie consommée est proportionnelle au

nombre de bits transmis. Par conséquent, de manière optimale, seules les données intéressantes doivent être transmises. Or, les autres données auront peut-être un intérêt plus tard soit pour les informations qu'elles contiennent, soit pour contrôler le fonctionnement du capteur sans fil. De plus, l'importance des données n'est pas connue à l'avance et il est parfois difficile de la déterminer avant le déploiement in-situ. Dans tous les cas, cela oblige à stocker une certaine quantité de données temporairement ou de manière prolongée.

La relation entre gestion de la mémoire et de la communication apparaît également lorsque la communication est indisponible ou perturbée. Ainsi, les données qui auraient dû être envoyées doivent être stockées en attendant le rétablissement de la communication. La durée d'indisponibilité étant en général inconnue, il est indispensable de répertorier correctement les données pour qu'elles puissent être récupérées par la suite.

La plupart des capteurs sans fil dispose de microcontrôleurs équipés de mémoires de types SRAM et Flash. Ces dernières autorisent un stockage non volatile des données c'est-à-dire que, même après une coupure d'alimentation, le contenu de la mémoire reste intact. Les mêmes questions que précédemment se posent à savoir quelles données stocker et comment. De plus, l'écriture ou la programmation de la mémoire Flash implique certaines précautions qui seront présentées plus en détails dans la partie 3.2.

Les applications d'acquisition de données sont les premières ciblées par le développement d'un système de fichiers comme LiveFile. Même dans une application où les données collectées sont directement transmises, suivant la fréquence d'échantillonnage requise, un système tel que LiveFile peut avoir son utilité. Si la fréquence d'acquisition est extrêmement rapide, la disponibilité et les caractéristiques du module de communication peuvent requérir le stockage des données avant envoi. Inversement avec une fréquence d'acquisition lente, le module de communication peut être mis en mode veille pour économiser de l'énergie et être réveillé après la réception d'une certaine quantité de données à envoyer, d'où la nécessité de les stocker entre-temps.

Les applications de substitution de l'infrastructure fixe peuvent également requérir un système comme LiveFile. Si, un fichier volumineux est transféré à travers le réseau, celui-ci peut être découpé et envoyé par morceaux. L'objectif visé est la préservation du fonctionnement du réseau en s'adaptant à sa capacité pour éviter les phénomènes de congestion voire d'écroulement.

Dans une application de RCSF, les trois principales étapes sont :

- l'acquisition des données ;
- le stockage des données ;
- la transmission des données.

Une quatrième étape « l'agrégation des données » peut s'insérer entre les deux premières ou dernières. En effet, l'agrégation peut avoir lieu avant le stockage pour économiser de la mémoire ou juste avant la phase de transmission pour pouvoir envoyer une plus grande quantité d'information à la fois (voir Figure 3.1).

Si le dimensionnement du capteur le permet, l'ensemble des données acquises peuvent être stockées avant leur transmission. Les données intéressantes sont donc choisies au sein de la station centrale de collecte. Cependant, cette situation ne représente pas un cas général, bien au contraire. Une gestion des ressources disponibles est primordiale, voire obligatoire, et passe par la manipulation d'un nombre restreint de données. Ainsi, la mémoire et l'énergie sont préservées par moins de stockage et de transmission. De plus, la puissance de calcul

utilisée pour les opérations d'agrégation et de cryptage de données dépend de la quantité de données traitées.

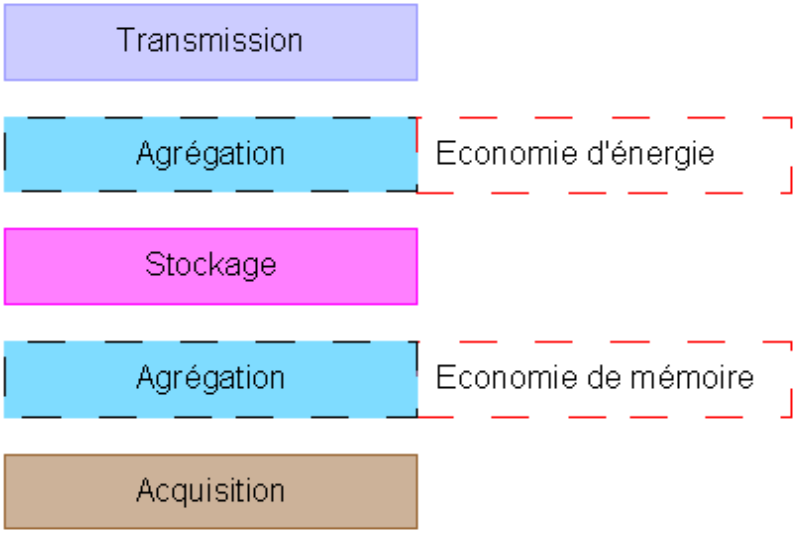

Figure 3.1 – Principaux traitements effectués au sein d'un capteur sans fil

Par conséquent, une sélection des données doit avoir lieu à chaque étape. A l'acquisition, les données n'ayant pas les critères souhaités ne sont pas retenues. Les principales données sont stockées en mémoire non volatile. Les autres sont disponibles durant une période de temps établie selon l'application ou les ressources en mémoire vive disponible. Les données anciennes sont remplacées par d'autres plus récentes. Pour réduire la quantité de données transmises à travers le réseau, une sélection doit être faite par l'intermédiaire d'interrogations provenant de la station centrale de collecte. De cette manière, seules les informations requises et donc les plus importantes sont diffusées au sein du réseau. Au regard des ressources disponibles, un capteur sans fil est peu propice à héberger un système de gestion de base de données. Les interrogations ou requêtes effectuées sur les données des capteurs doivent être ciblées et organisées pour être performantes tant au niveau de la précision et de la qualité du résultat qu'au niveau énergétique.

Les deux grandes fonctionnalités du système LiveFile sont :

- la gestion de la mémoire des capteurs sans fil ;
- l'interrogation des données collectées.

3.1.2 L'intégration dans le système d'exploitation LIMOS

Dans le chapitre précédent, le système LIMOS a été présenté en détails. Ces caractéristiques les plus importantes ont été mises en évidence. L'une de ses principales particularités est son architecture hybride basée sur les événements et les processus. L'autre est l'utilisation des concepts dérivés du système LINDA qui apporte un niveau d'abstraction pour la gestion des périphériques et la communication entre les processus et les événements. Ces différentes gestions s'appuient sur la notion de tuple qui représente un espace mémoire associé à un objet système (événement ou processus) manipulé par le système d'exploitation. L'accès à un tuple s'effectue à l'aide des primitives In() et Out() pour la lecture et l'écriture.

Tous ces principes offrent un cadre pour la gestion de la mémoire. En dérivant le fonctionnement de la primitive Out(), l'écriture dans la mémoire RAM ou Flash est possible. De la même manière, l'appel à la fonction In() autorise la lecture d'un emplacement mémoire.

Ces modifications permettent l'intégration du système LiveFile dans le noyau LIMOS. Elles ont l'avantage d'offrir une interface simple et répandue dans les systèmes utilisant le concept LINDA comme, par exemple, les noyaux DREAM et SDREAM.

Le fonctionnement de la primitive Out() ne se résume pas à placer une donnée à une adresse mémoire. Il prend en compte la particularité de la mémoire Flash et la nécessité d'une écriture page par page et de préserver celles qui ont été les plus utilisées. La primitive In() est appelée pour la recherche des données requises suite à une interrogation ou requête. Si le système l'autorise, l'allocation dynamique de mémoire vive est assurée par la fonction In() (voir Figure 3.2).

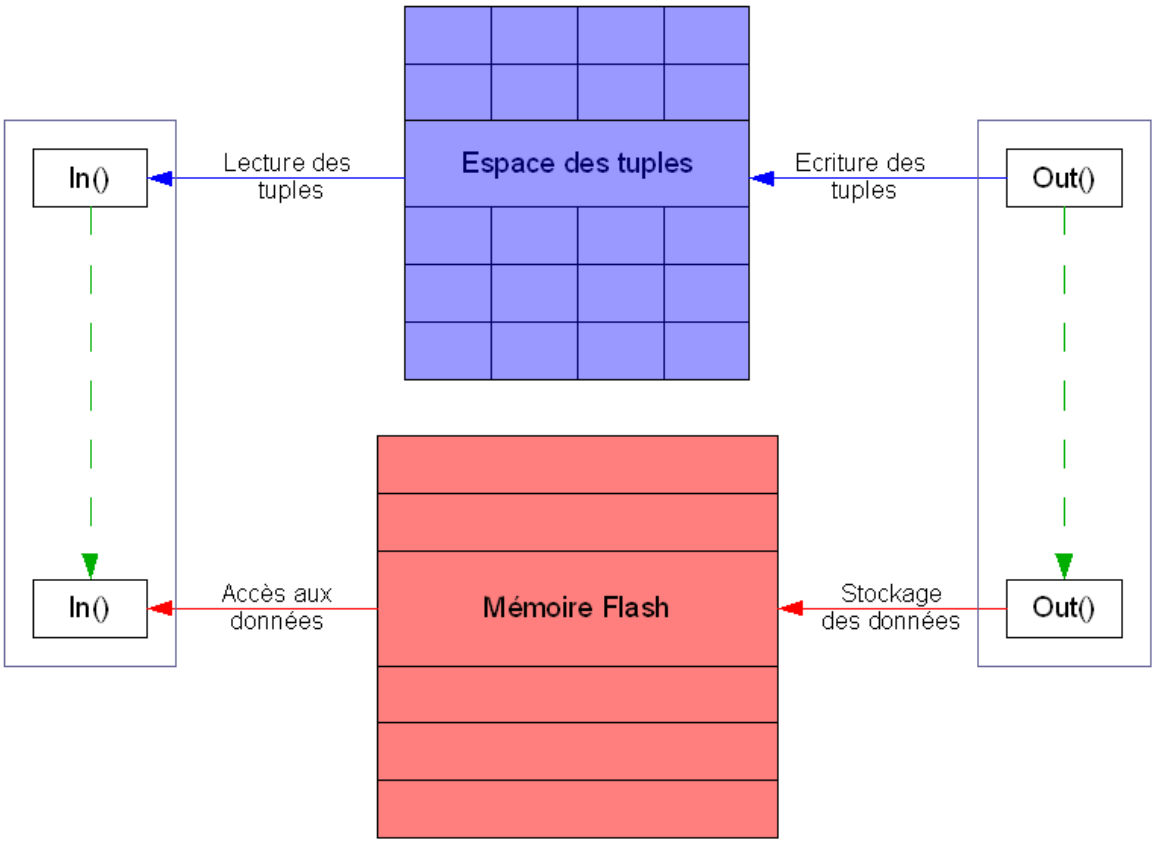

Figure 3.2 – Extension des fonctionnalités des primitives In() et Out()

L'unification des différents modules de gestion du noyau LIMOS au sein du concept LINDA étendu, constitue un point de départ dans la construction d'un intergiciel ou logiciel médiateur (« middleware ») dédié aux RCSF (voir Figure 3.3). Les intergiciels ont pour fonction principale l'interfaçage entre les dispositifs matériels et les applications d'un capteur sans fil. Plus précisément, leur rôle est d'offrir des services standardisés pour une manipulation efficace et adaptée des ressources matérielles disponibles. Ils viennent en réponse aux nombreux challenges issus des RCSF dont quelques uns ont été énoncés dans le chapitre 1 et où l'on trouve entre autres [Hadim 2006] :

- la gestion des ressources disponibles ;
- la gestion des données ;
- l'hétérogénéité des capteurs ;
- l'administration des capteurs à distance ;
- la reconfiguration des capteurs ;
- la sécurité.

La reconfiguration des capteurs est soit issue d'une demande à distance soit requise par un changement de topologie du réseau, la plupart du temps, subi.

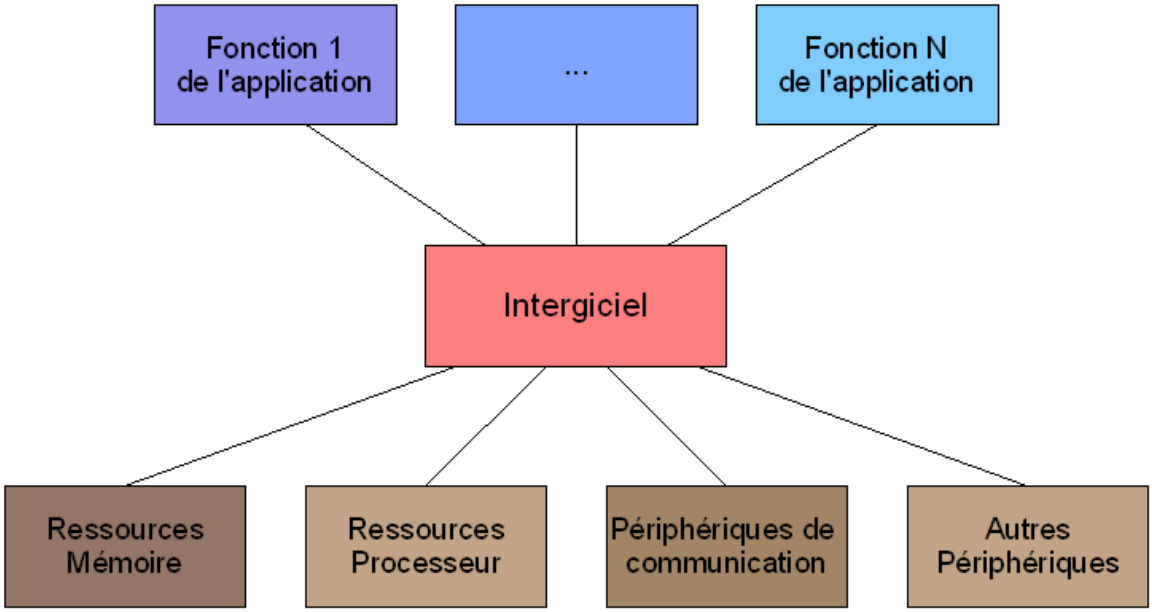

Figure 3.3 – La notion d'intergiciel

L'intérêt d'un intergiciel est de réunir tous les éléments d'un capteur et de concentrer leurs efforts dans un objectif commun qui est d'offrir le support nécessaire à l'application embarquée. Les principaux intergiciels sont TinyDB avec le système d'exploitation TinyOS [Madden 2005] et Cougar [Bonnet 2001]. Ils seront présentés dans la section 3.4.1 de ce chapitre.

3.2 La gestion de la mémoire Flash

Le système LiveFile est dédié à la gestion intelligente de la mémoire d'un capteur sans fil. Par rapport à la mémoire SRAM, celle de type Flash requiert plus de précautions au niveau de son utilisation. Cette partie 3.2 commence, d'abord, par la présentation des éléments importants à connaître sur la mémoire Flash. Puis, les particularités de la gestion de la mémoire Flash par le système LiveFile sont explicitées en les comparant à celles d'autres systèmes.

3.2.1. Présentation de la mémoire Flash

La plupart des microcontrôleurs actuels sont équipés d'une quantité plus importante de mémoire Flash que de mémoire SRAM comme c'est le cas pour le microcontrôleur AT91SAM7S256 d'ATMEL Corporation [ARM7 2007]. Dans le chapitre précédent, nous avons pu constater que les systèmes d'exploitation dédiés aux RCSF ont une empreinte mémoire inférieure à 6Koctets. Appliqué au microcontrôleur AT91SAM7S256 possédant 256Koctets de mémoire Flash, cela représente donc environ 250Koctets de mémoire Flash inutilisée. Pour certains microcontrôleurs, il est possible, après avoir retiré certains verrous de convertir la mémoire Flash interne en mémoire EEPROM telle qu'on la retrouve dans des périphériques de stockage amovibles.

Les caractéristiques de la mémoire Flash impose certaines règles pour la lecture et l'écriture. Cette mémoire est divisée en pages d'une taille préfixée. Une écriture ou une lecture ne peut s'effectuer que page par page. Deux familles de mémoire Flash existent :

- mémoire de type NOR ;
- mémoire de type NAND.

Cette répartition est liée au type de circuit « NOR » ou « NAND » utilisé pour le stockage d'un bit [Zeinalipour-Yazti 2005]. Les mémoires de type « NOR » autorisent des accès à la mémoire de manière aléatoire très rapides. Elles consomment, en règle générale, plus d'énergie que celles de types « NAND » pour les opérations de programmation mais elles sont, en revanche, plus fiables [Toshiba 2006] (voir Figure 3.4). Les technologies des mémoires sont en constante évolution. De nombreux travaux de recherche sont en cours dont l'objectif est la création de mémoire Flash plus économe et acceptant de plus en plus d'écritures [AMD 2001]. En outre, les mémoires Flash sont fortement pressenties pour remplacer les disques durs actuels au sein des ordinateurs personnels.

Leur grande capacité de stockage, leur mode d'utilisation basé sur un courant de faible intensité ainsi que leur consommation en énergie relativement modérée font des mémoires de type Flash une solution adaptée pour sauvegarder, de manière persistante, les données au sein d'un capteur sans fil [Mathur 2006a]. De par les améliorations successivement apportées au niveau de l'utilisation de la mémoire Flash, le stockage d'un bit consomme moins d'énergie que la transmission d'un bit à l'aide d'un module de consommation radio classique [Diao 2007b].

Toute écriture requiert un effacement au préalable de la page concernée. Par conséquent, on emploie plus le terme de *programmation d'une page* plutôt que d'écriture. La dernière et pas la moindre contrainte associée à la mémoire Flash est l'existence d'un nombre limite de programmation supportée par une page. Après avoir atteint cette limite, la page devient inutilisable.

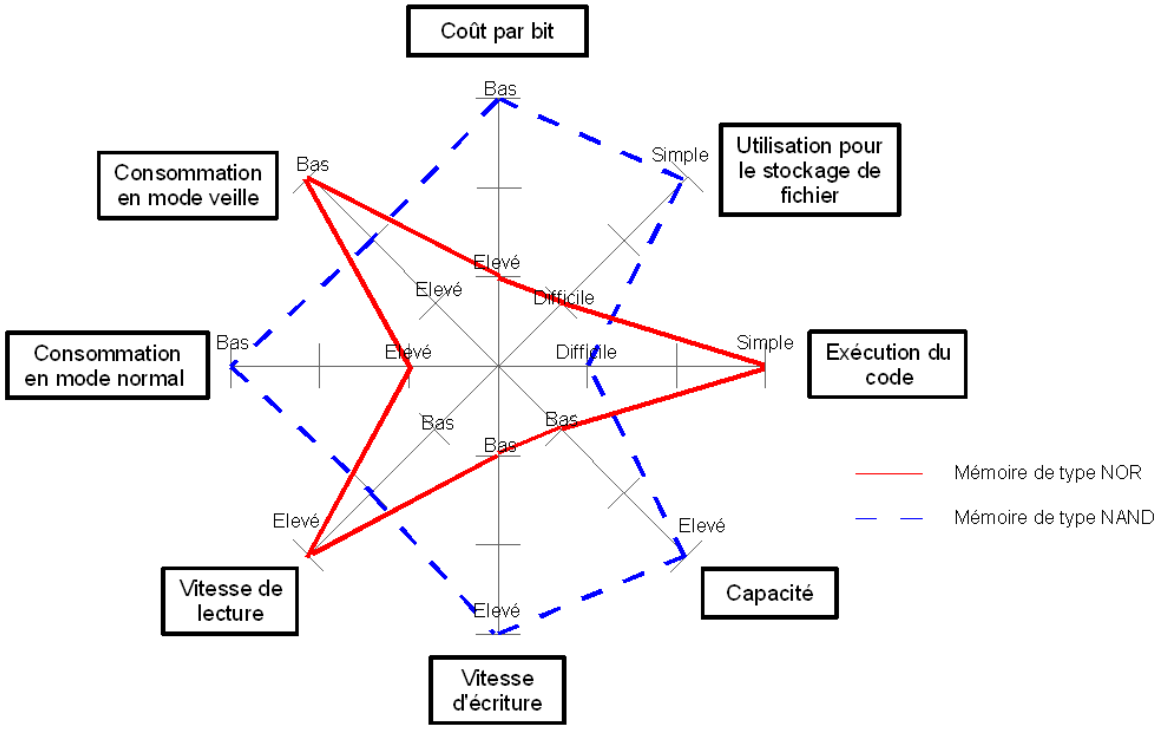

Figure 3.4 – Comparaison entre les mémoires Flash de type NAND et NOR [Toshiba 2006]

Dans certains dispositifs, la procédure de programmation est complexifiée par l'existence d'effacement par bloc de pages plutôt que par page seule. Quant au microcontrôleur AT91SAM7S256, il permet une programmation page par page bien qu'il soit nécessaire de sécuriser une région d'une taille de 16Ko pour protéger en écriture une page. A titre d'exemple, le microcontrôleur AT91SAM7S256 a une mémoire Flash découpée en 1024 pages de 256 octets. Chaque page a une « durée de vie » de 10.000 programmations. Enfin, le temps nécessaire à la programmation d'une page est de 6ms.

3.2.2 Les concepts définis pour la gestion des données

La gestion de la mémoire par le système LiveFile [De Sousa 2007a] [De Sousa 2007b] est régie par 3 grands concepts qui sont :

- l'adaptation des structures de données à l'application ;
- la gestion intelligente des ressources mémoires ;
- un niveau d'abstraction pour l'accès aux données.

3.2.2.1 Différents formats pour le stockage des données

La plupart du temps, dans les RCSF, le ou les types de données manipulées par une application sont connus, à l'avance, avant le déploiement in-situ. Les applications d'acquisition de données environnementales ou de détection et de suivi d'intrusions en sont des exemples.

La connaissance de ces éléments permet de préconfigurer le système pour accueillir des données dont la taille et les traitements associés sont connus. Le format des données est ainsi défini et intégré au système et permet un stockage et un accès adapté (voir Figure 3.5). Ces informations sont importantes pour organiser la mémoire Flash afin d'accueillir les différents éléments à stocker et réserver l'espace nécessaire.

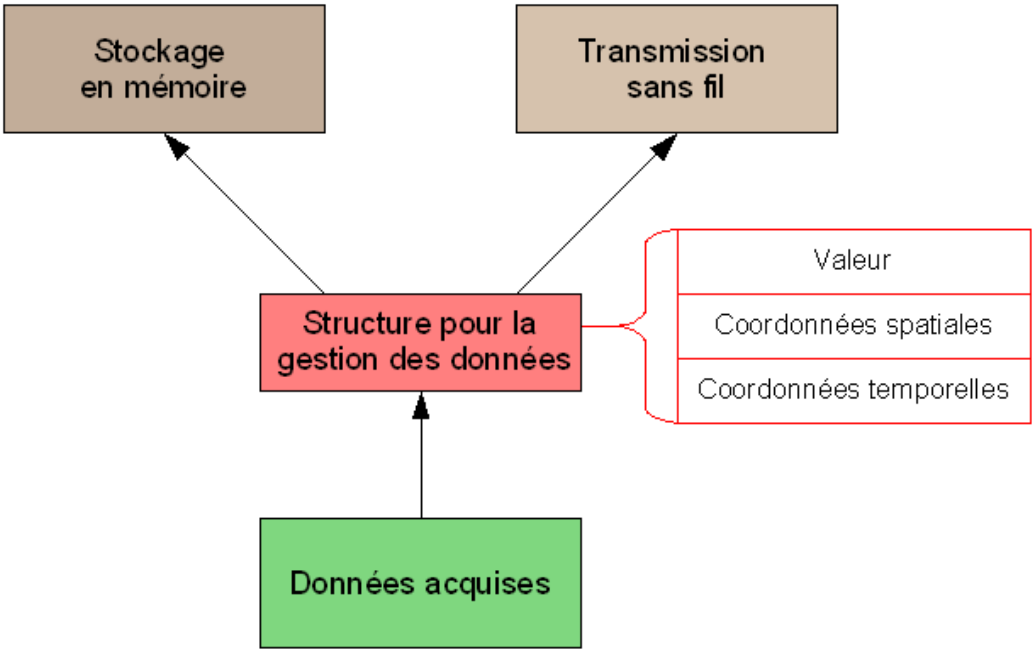

Figure 3.5 – Structures préétablies pour la gestion des données collectées

Une gestion intelligente de la mémoire Flash n'est possible qu'en disposant des renseignements sur chaque page. Les principaux sont l'état de la page à savoir occupée ou non et, le nombre d'écritures déjà subies par celle-ci. Selon le type d'application, d'autres renseignements peuvent être requis comme la date de la dernière écriture utile pour connaître la durée durant laquelle la page n'a pas été utilisée.

Un autre attribut est ajouté par rapport à la politique de gestion de la mémoire assumée au sein du système LiveFile. Cet attribut exprime les caractéristiques des données stockées dans la page concernée. Les 3 valeurs que peut prendre cet attribut sont :

- valeur « RECORD » ;
- valeur « FILE » ;
- valeur « CHECKPOINT ».

La valeur « RECORD » indique que la page contient un ensemble d'enregistrements dont le nombre dépend de la taille de ceux-ci. Les données enregistrées dans des fichiers clairement identifiés sont stockées dans des pages de type « FILE ».

Le rôle de la mémoire Flash est de conserver les données de manière non volatile. Ainsi, en cas de panne d'énergie ou d'un redémarrage système, les données ne sont pas perdues. Cependant, pour récupérer les données, il faut savoir où elles se situent et comment elles étaient organisées. Les pages « CHECKPOINT » contiennent les informations nécessaires à la restauration, au niveau des données gérées, du système dans l'état où il était avant son arrêt. Elles peuvent également contenir la liste des emplacements libres situés au sein des pages utilisées. Les pages sont remplies au fur et à mesure de la réception des données à stocker. Elles le sont de manière à laisser le moins d'espace inutilisé. La programmation d'une page est, par exemple, exécutée quand l'espace restant dans celle-ci est inférieure à la taille d'un enregistrement. Au bout d'un certain temps, des enregistrements d'une page peuvent être considérés comme obsolètes ou inutiles et donc libérés des emplacements qui pourront être utilisés plus tard. Des données ne sont pas immédiatement insérées dans ces espaces libres pour ne pas surexploiter certaines pages par rapport à d'autres (voir Figure 3.6).

La présence de cet attribut implique qu'une page ne peut stocker qu'un seul type de données. Ce choix s'explique, d'une part, par le fait que pour économiser les ressources disponibles, les traitements engendrés par le système LiveFile doivent être les plus simples et les plus performants possibles.

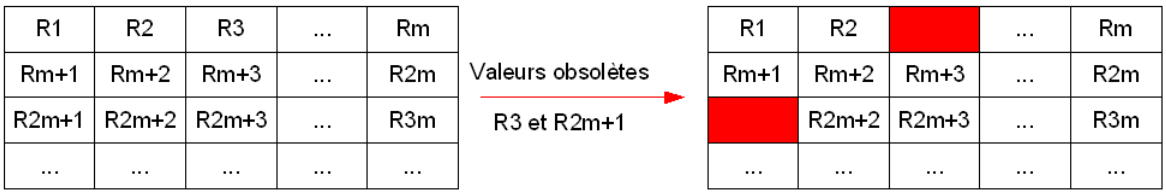

Figure 3.6 – Création d'un espace libre au sein de pages utilisées

Alterner entre la sauvegarde d'enregistrements et de fichiers au sein d'une même page accélère l'étape de programmation mais complexifie la récupération des données. En interdisant ce mode de fonctionnement, on peut établir une liste de pages de chaque type. D'autre part, la majorité des applications manipulent uniquement des enregistrements ou des fichiers. Il arrive parfois qu'une application stocke des enregistrements et, à un moment donné, qu'elle doive s'occuper de la sauvegarde d'un fichier. Ainsi, elle pourra créer un emplacement de stockage principal ou prioritaire pour les enregistrements et un autre pour les fichiers qui pourraient potentiellement être réceptionnés.

La liste principale des attributs d'une page sont donc :

- l'état de la page ;
- le type de données stockées ;
- le nombre d'écritures ;
- la date de dernière écriture.

Désormais, la nouvelle problématique mise en avant est de déterminer où conserver l'ensemble de ces renseignements. La mémoire Flash est limitée par le nombre d'écritures qu'elle peut supporter mais pas par le nombre de lectures. Par conséquent, le stockage de ces renseignements au sein de la page concernée semble être le mieux indiqué. Sous le système Livefile, les attributs d'une page sont stockés au début de celle-ci à la manière d'un en-tête de message réseau. A chaque opération de programmation, on récupère les données de l'en-tête de la page et on les modifie pour les adapter aux nouvelles données. C'est le cas du nombre d'écritures qui est ainsi transmis, au cours du temps, de programmations en programmations. Chaque application ou déploiement du système LiveFile implique d'établir la structure des attributs utiles de la page (voir Figure 3.7).

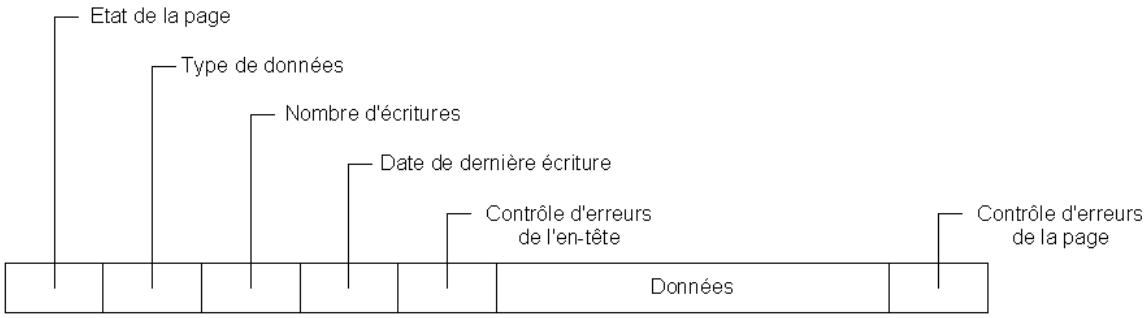

Figure 3.7 – Exemple d'attributs de l'en-tête d'une page

Une autre solution envisageable pour le stockage des informations d'une page est de les garder dans un bloc de pages situé à un emplacement connu en début ou en fin de l'espace mémoire disponible au sein de la mémoire Flash. En procédant de cette manière, la vision de l'occupation de la mémoire Flash est rapidement établie et ne nécessite pas d'accéder à toutes les pages. A l'inverse, pour réaliser la programmation d'une page, il faudrait accéder à au moins 2 pages plutôt qu'une. Le processus d'écriture en est donc complexifié et surtout certaines pages s'usent plus rapidement que d'autres (voir Figure 3.8).

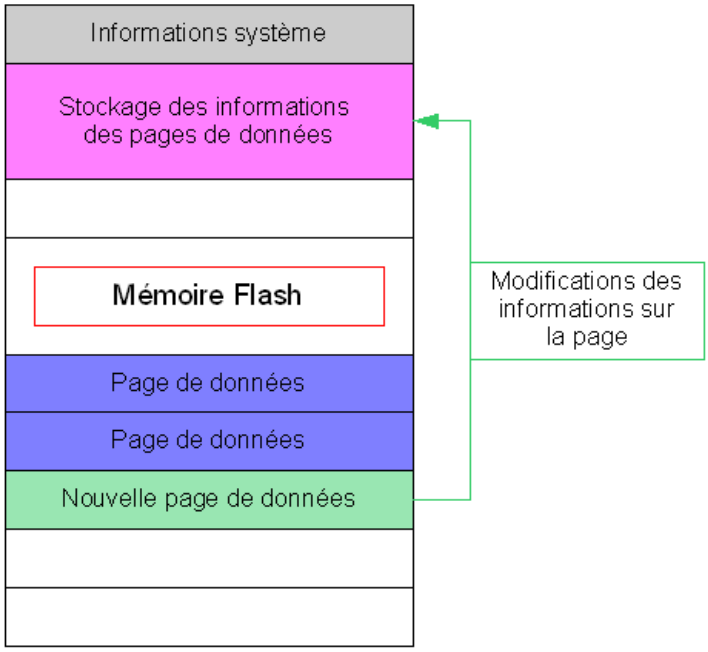

Figure 3.8 – Une autre solution pour le stockage des attributs d'une page

L'organisation de la mémoire Flash est adaptée par le système LiveFile aux besoins de l'application au cours du temps (voir Figure 3.9).

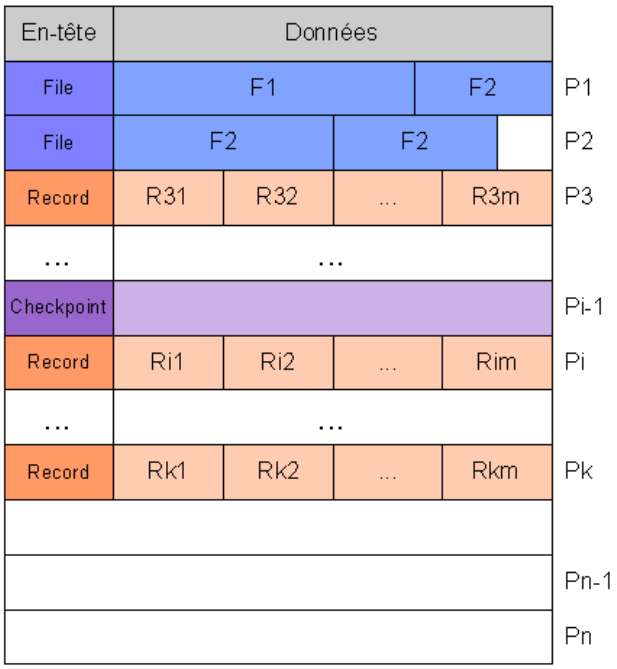

Figure 3.9 – Organisation générale de la mémoire Flash gérée par le système LiveFile

3.2.2.2 Gestion intelligente des ressources

La solution de l'en-tête pour le stockage des attributs d'une page a été retenue car elle minimise le nombre d'écritures dans la mémoire Flash. Elle est l'un des mécanismes de gestion économe de la mémoire Flash. Les autres portent sur les conditions de déclenchement d'une programmation et la préservation des pages les plus utilisées.

Dans une première approche, le stockage des données acquises est effectué de manière séquentielle, donnée par donnée, page par page. Or, en règle générale, une donnée a une taille plus petite voire beaucoup plus petite que celle d'une page. La mémoire se retrouve ainsi considérablement fragmentée. En arrivant à la fin de la mémoire Flash, on peut facilement revenir au début de la mémoire et rajouter des données. Une rotation dans l'écriture des pages s'effectue naturellement et permet leur préservation (voir Figure 3.10).

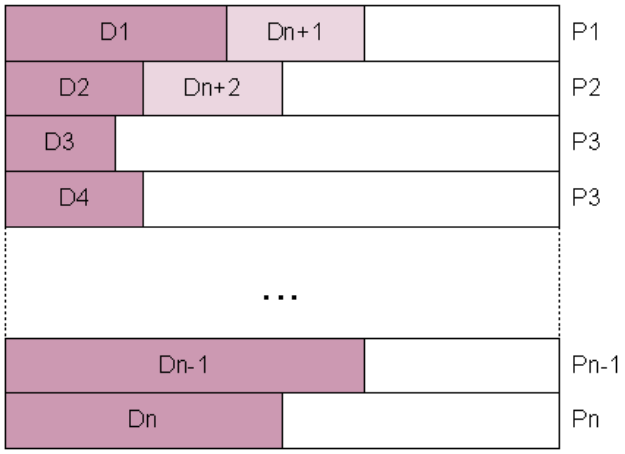

Figure 3.10 – Ecritures séquentielles des données dans la mémoire Flash

Cependant, certaines applications autorisent de différer la sauvegarde des données en mémoire Flash. Elles permettent l'intégration d'une étape intermédiaire matérialisée par le stockage des données dans un espace mémoire ou « buffer » réservé dans la mémoire RAM. La taille de ces espaces mémoires peut être inférieure ou égale à celle d'une page de mémoire Flash. En effet, il peut être inférieur, si au lieu de considérer la donnée seule, on considère une séquence significative composée de plusieurs données. Par exemple, la taille d'une page de mémoire Flash d'un microcontrôleur AT91SAM7S256 est de 256 octets. Les données peuvent donc être écrites par bloc de 256 octets dans la mémoire Flash après avoir été stockées dans un buffer de mémoire SRAM. La mémoire Flash est ainsi à la fois moins sollicitée et moins fragmentée. Chaque page remplie ne possède pas 256 octets de données car il faut prendre en compte l'espace mémoire occupé par l'en-tête (voir Figure 3.11).

Les conditions du déclenchement de l'écriture dans la mémoire Flash doivent être établies suivant les contraintes de l'application. Le stockage dans la mémoire Flash peut avoir lieu plus fréquemment que par bloc de 256 octets (en intégrant l'en-tête de la page) pour limiter la quantité de données perdues en cas de problème d'alimentation. En effet, les données stockées dans la mémoire SRAM seront effacées en cas de redémarrage du système. Un compromis entre la préservation de la mémoire Flash et la quantité de données que l'on peut potentiellement perdre existe et doit être trouvé pour chaque application considérée. Pour les applications où toutes les données sont critiques, le stockage de manière séquentielle dans la mémoire Flash reste la solution la plus simple mais peut être remplacé par une collaboration entre plusieurs capteurs sans fil. Après avoir acquis une donnée, le capteur sans fil stocke celle-ci en mémoire SRAM et en envoie une copie à ses capteurs voisins. Ainsi,

même en cas de panne, la donnée n'est pas perdue. Ce mode de fonctionnement est abordé dans la section 3.5.1 dédié à la sauvegarde des données.

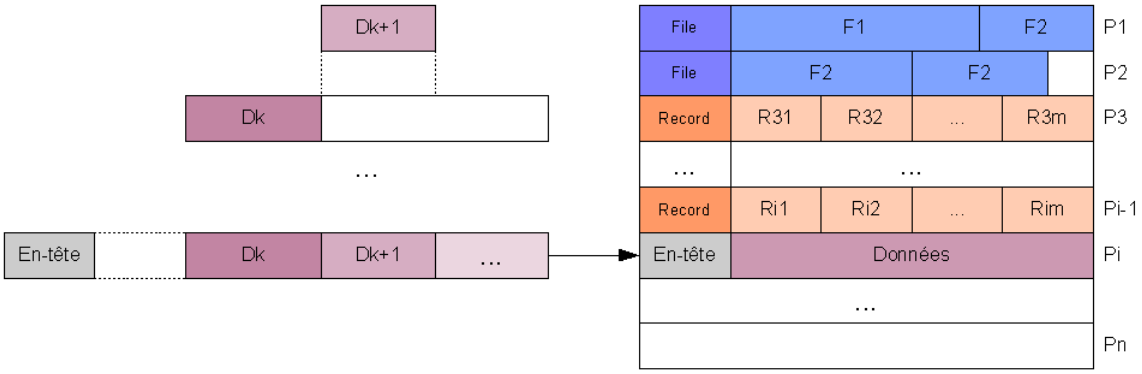

Figure 3.11 – Utilisation de la mémoire Flash sous le système LiveFile

Suivant la fréquence d'acquisition des données, le nombre de buffer de mémoire SRAM varie. Si cette fréquence est plus longue que le temps total de stockage des données dans la mémoire Flash qui comprend les étapes de recherche et de programmation d'une page, 2 buffers seulement sont nécessaires : l'un pour la programmation, l'autre pour la lecture de la mémoire Flash. Pour les applications avec des fréquences d'acquisition plus fortes, 2 buffers d'écriture peuvent être requis : l'un pour réaliser l'opération de programmation et l'autre pour stocker les données nouvellement acquises. La durée de la programmation d'une page étant incompressible, le temps nécessaire à une écriture dépend de la durée de recherche et de mise-à-jour des attributs de l'en-tête d'une page. Finalement, si la mémoire Flash est à la fois très sollicitée et fragmentée avec plus aucune page entièrement vide disponible, un quatrième buffer est obligatoire. Son rôle est de stocker temporairement la page sélectionnée pour ajouter de nouveaux éléments en fin (voir Figure 3.12).

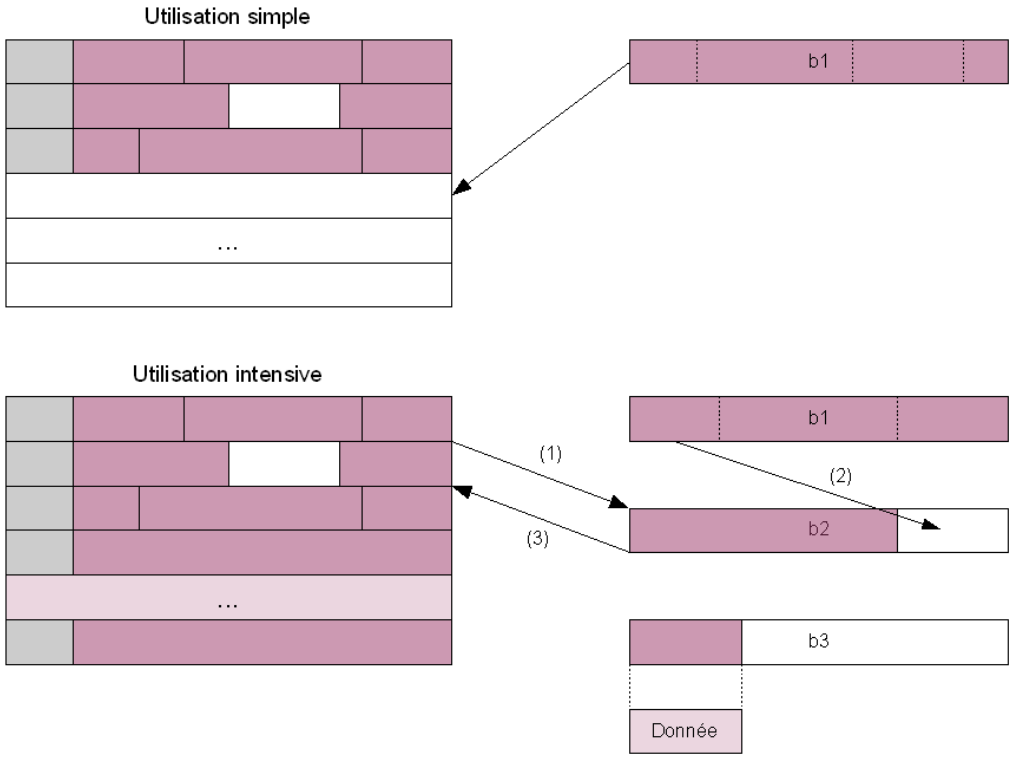

Figure 3.12 – Nombres de buffers nécessaires suivant l'application supportée

L'écriture d'une page de mémoire Flash est une opération assez longue et consommatrice d'énergie. Limiter le nombre d'écritures permet donc d'économiser les ressources du système. Dans le microcontrôleur AT91SAM7S256, la durée de la programmation d'une page est de 6ms et doit être prise en compte si le nombre d'écritures planifié pour une application est important. Le fonctionnement du système d'exploitation pourra être adapté pour répondre à ce type de sollicitation.

Le nombre d'écritures déjà subies par une page est connu grâce à son en-tête et autorise l'élaboration d'une technique de rotation de programmation (« wear-levelling »). L'objectif d'un tel dispositif est de préserver les pages qui auraient été utilisées de manière plus intensives par rapport aux autres. Au départ, la mémoire Flash va être programmée dans l'ordre de numérotation de ses pages. Une page qui serait libérée pour cause de données stockées devenues obsolètes, ne sera réutilisée que si la mémoire Flash est remplie. En effet, si les pages libérées étaient directement reprogrammées, elles auraient deux écritures de plus par rapport à d'autres pages (voir Figure 3.13).

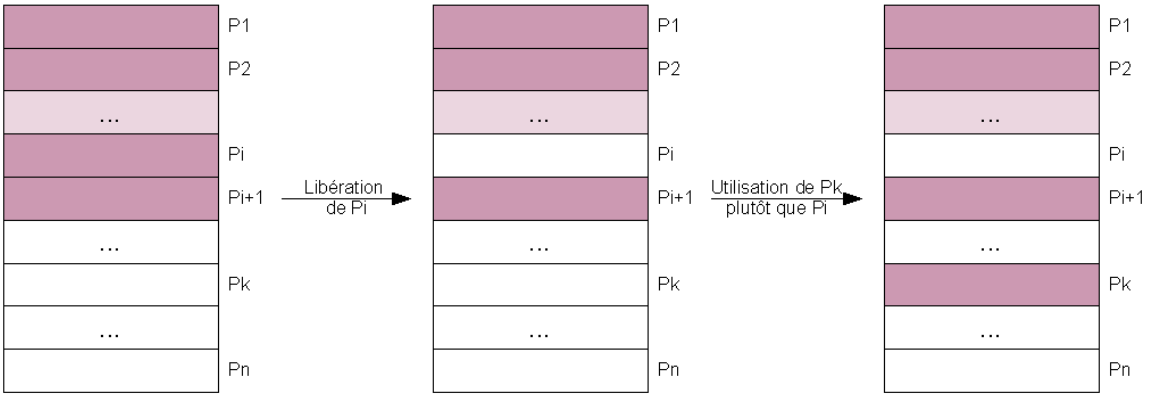

Figure 3.13 – Premières écritures dans la mémoire Flash

Au fur et à mesure de l'application, la mémoire Flash va être de plus en plus fragmentée. Cela implique d'une part de stocker les emplacements mémoires vides et d'autre part de savoir pour chaque page si elle doit ou non être préservée. La première partie est réalisée, en stockant dans la mémoire SRAM, la liste des pages non utilisées pour éviter d'avoir à parcourir l'ensemble de la mémoire Flash pour les rechercher. Cette liste sera également stockée, en temps voulu, dans une ou plusieurs pages de type « CHECKPOINT » pour qu'en cas de redémarrage ou panne d'alimentation du système, les données stockées soient retrouvées. Dans le cas des enregistrements au format défini préalablement, la gestion des emplacements mémoire libres peut être affinée en considérant non seulement les pages intégralement libres mais également les espaces mémoires vides présents au sein des pages (voir Figure 3.14).

Cependant, toutes les pages libres ne vont pas faire partie de cette liste. Les pages les plus utilisées vont être reversées dans une liste dite de préservation. Pour déterminer dans quelle liste, une page va être répertoriée, un indicateur d'usure est défini. Celui-ci fixe le nombre d'écritures limites que peut subir une page avant d'être préservée. La valeur de cet indicateur est fixée au départ suivant les contraintes de l'application supportée. Cependant, pour certaines applications, cet indicateur ne doit pas être constant pour éviter la pénurie de pages libres au profit des pages préservées.

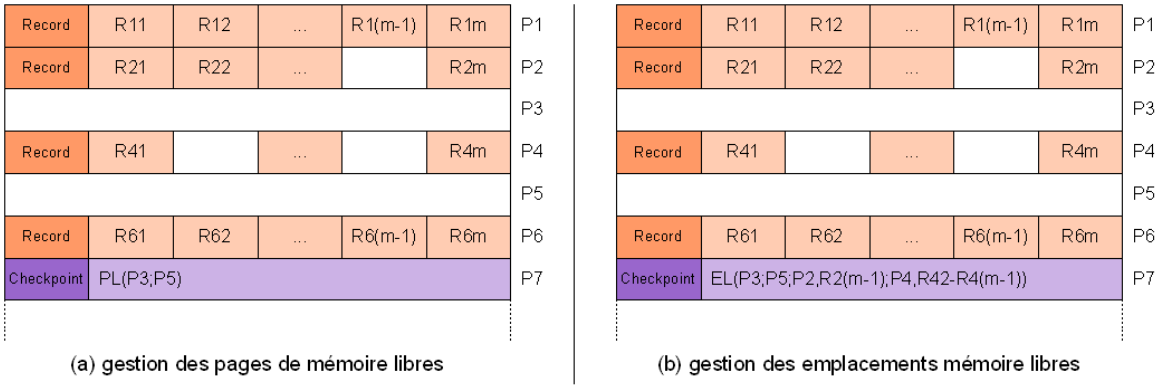

Figure 3.14 – Gestion des emplacements libres

Par conséquent, quand le nombre de pages préservées atteint un certain seuil, cet indicateur est incrémenté. Ainsi, les pages préservées sont donc toutes reversées dans la liste des emplacements libres (voir Figure 3.15).

```
/* Recherche d'une page libre */
nouvelle_page ← recherche_page_libre();

/* Initialisation du compteur de changement du seuil d'usure */
compt_chgt_seuil ← 0;

/* Recherche d'une page pouvant être encore utilisée */
TANT QUE (nouvelle_page.nbre_ecriture > seuil_usure)
ET (compt_chgt_seuil < chgt_seuil) FAIRE
        compt_chgt_seuil ← compt_chgt_seuil + 1;

        /* Préservation de la page trop utilisée */
        preservation(nouvelle_page);

        nouvelle_page ← recherche_page_libre();
FINTANTQUE

SI (comp_chgt_seuil = chgt_seuil) ALORS
        /* Modification de la valeur du seuil d'usure */
        seuil_usure ← seuil_usure + μ;

        /* Réintégration des pages auparavant préservées */
        reintegration(liste_page_preservees);
FINSI
```

Figure 3.15 – Fonctionnement du mécanisme de préservation des pages

En utilisant cette technique, pour une acquisition de données constamment renouvelées, toutes les pages dédiées au stockage ont quasiment le même nombre de programmations subies. Pour une application plus complexe où des pages entières sont conservées par rapport à certaines de leurs propriétés, cette répartition du nombre de programmations n'est pas uniforme sur l'ensemble de la mémoire Flash mais l'est au niveau du groupe des pages les plus utilisées. Une page peut être peu utilisée pour diverses raisons. La principale est que celle-ci contient des informations systèmes utiles pour la récupération des données après une panne. Ces informations sont, par exemple, le début et la fin de la mémoire Flash réellement allouée aux stockages des données acquises. Une page qui

stockerait des données acquises importantes par rapport aux phénomènes qu'elles caractérisent pourra rester inchangée tout au long du fonctionnement du capteur. Par exemple, dans une application de surveillance d'une forêt, les événements marquants comme des incendies seront stockés au sein de pages de la mémoire Flash pour, a priori, un certain temps. Pour pallier ce problème, ces renseignements peuvent être déplacés dans des pages très utilisées pour les économiser voire dans des pages préservées.

De manière synthétique, la gestion intelligente des ressources s'appuie sur les trois mécanismes suivants :

- la programmation de pages différée grâce à l'utilisation de buffer de mémoire SRAM ;
- la rotation de la programmation entre les différentes pages ;
- la préservation des pages les plus utilisées.

Suivant l'application, ces mécanismes pourront être utilisés à différents niveaux. Pour des applications simples avec une faible quantité de données à stocker, la préservation des pages ne sera pas forcément utile et viendra alourdir le fonctionnement du système. De plus, la gestion des ressources, par l'intermédiaire de ces mécanismes, ne se résume pas seulement à la mémoire. L'énergie du capteur est d'une part économisée en limitant le nombre de programmations dans la mémoire Flash. D'autre part, cette gestion n'implique pas des traitements lourds qui monopoliseraient le processeur pendant des laps de temps conséquents.

3.2.2.3 Niveau d'abstraction pour l'accès aux données

Le système LiveFile possède des mécanismes évolués qui en font un module utilisable dans différents systèmes d'exploitation dédiés aux RCSF afin de gérer la mémoire disponible au sein des capteurs utilisés. Cependant, le système LiveFile a été élaboré pour répondre à des besoins concernant les fonctionnalités du système d'exploitation LIMOS. L'interface d'utilisation du système LiveFile a été conçue pour permettre son intégration complète au sein de ce système d'exploitation.

Dans le chapitre précédent, le concept LINDA et sa place dans le système LIMOS ont été présentés. Les gestions des périphériques, des ressources partagées, de la communication et de la synchronisation sont assurées par une version adaptée de ce concept. L'intégration du système LiveFile va donc être réalisée en étendant de nouveau la portée de ce concept. Au sein du système LIMOS, celui-ci repose essentiellement sur les primitives In() et Out(). Ces deux fonctions vont donc être modifiées pour faire appel aux mécanismes issus du système LiveFile (voir Figure 3.16).

La primitive Out() possède à la base 3 paramètres :

- « key » : identifiant du tuple sélectionné pour recevoir des données ;
- « data » : données à insérer dans le tuple ;
- « size » : taille des données à insérer.

La primitive Out() est modifiée pour assurer l'écriture dans la mémoire Flash. Ainsi, le paramètre « key » contient soit l'identifiant du tuple soit les consignes pour le stockage des données dans la mémoire Flash. Dans ce dernier cas, le paramètre « key » est séparé en 3 éléments par un caractère séparateur (« : »). Le premier est le caractère « f » pour indiquer qu'il s'agit d'une écriture dans la mémoire Flash. Le second est le type de données stockées c'est-à-dire « RECORD », « FILE » ou « CHECKPOINT ». Afin de simplifier et d'accélérer le processus de recherche de données, une page ne peut stocker des données que d'un seul type. Au moment de l'écriture, ce mode de fonctionnement peut nécessiter ponctuellement la

réservation de plusieurs buffers d'écriture : un pour les données « RECORD » et un autre pour celle de type « FILE » par exemple.

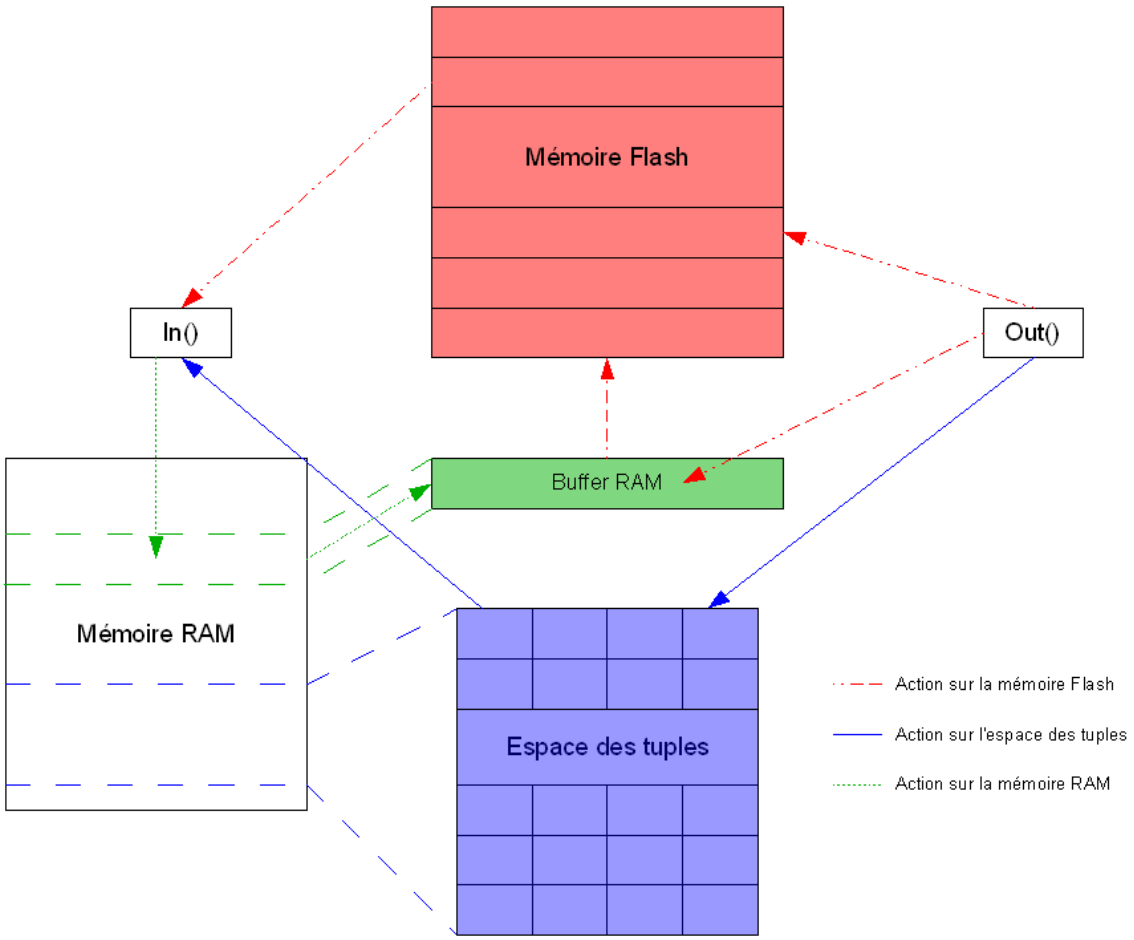

Figure 3.16 – Modification des primitives In() et Out()

Le dernier élément est le lieu de stockage c'est-à-dire soit localement, soit sur un autre capteur. Cette fonctionnalité sera présentée plus en détails dans le paragraphe 3.5.1 de ce chapitre. Quand un paramètre « key » au format que nous venons d'énoncer est transmis à la primitive Out(), les traitements propres au système LiveFile intégrés au sein de cette primitive vont être appliqués aux données stockées dans le paramètre « data ».

La primitive Out() fait appel à des fonctions associées à chaque type de données :

- record_insert() pour les enregistrements ;
- file_insert() pour les fichiers ;
- save_insert() pour les sauvegardes des informations à récupérer après une panne de système.

La fonction « record_insert() » s'appuie sur la structure de données associée aux enregistrements. Les attributs de cette structure sont, par exemple, liés à la datation, à la localisation et aux différentes valeurs de l'enregistrement. Le format des informations « CHECKPOINT » est également propre à l'application.

La primitive In() a, quant à elle, 3 fonctions. La première concerne le fonctionnement classique du système d'exploitation LIMOS avec l'accès à un message d'un tuple. La deuxième est la réservation de buffers à partir de la mémoire RAM. A l'initialisation du système d'exploitation LIMOS, un bloc de mémoire RAM peut être réservé pour permettre l'allocation dynamique de buffers pour réaliser certaines actions. Les buffers de programmation et de lecture de la mémoire Flash peuvent être ainsi alloués suivant les besoins apparaissant au cours de l'application. Cependant, pour optimiser le fonctionnement du système, l'allocation dynamique de la mémoire peut ne pas être utilisée. L'ensemble des besoins nécessaires en termes de mémoire sont déterminés a priori.

La fonction In() permet également d'accéder aux données présentes dans la mémoire Flash. Le type de données recherché et le mode d'accès souhaité sont indiqués dans l'argument « key » de cette primitive :

- « f : Pk : id_capteur » : récupération de toutes les données de la page « k » du capteur « id_capteur » ;
- « f : Rk : id_capteur » : récupération de l'enregistrement « k » du capteur « id_capteur » ;
- « f : PiRj : id_capteur » : récupération du $j^{ème}$ enregistrement de la page « i » du capteur « id_capteur » ;
- « f : Fk : id_capteur » : récupération du fichier « k » du capteur « id_capteur ».

La plupart du temps, la recherche de données s'effectuant localement, l'élément « id_capteur » est soit omis soit mis à la valeur « 0 ». Suivant l'application, certaines procédures de recherche comme, par exemple, celles liées aux fichiers sont inutiles et ne sont donc pas présentes pour ne pas alourdir le fonctionnement général du système. Le nombre et la complexité des traitements associés à la gestion de la mémoire peuvent être ainsi adaptés aux contraintes rencontrées.

A terme, de la même manière que pour le système LiveFile, l'ajout ou la suppression de modules au sein du système LIMOS pourraient être obtenus par mise-à-jour des primitives In() et Out(). Une fois déployés, les capteurs gérés par le système LIMOS pourraient recevoir de nouveaux modules à installer avec les versions associées de ces deux primitives. Plus généralement, le concept LINDA étendu tel qu'il vient d'être présenté pourrait être utilisé pour administrer le capteur et l'ensemble de ses composants logiciels associés.

3.2.3 Comparaison avec les autres systèmes

Dans le cadre des RCSF, différents systèmes ont été développés pour gérer la mémoire Flash des dispositifs utilisés. Ces systèmes prennent en compte les ressources limitées d'un capteur sans fil et abordent la gestion des données suivant différentes approches. L'apparition de mémoire Flash dont la consommation d'énergie est inférieure à celle d'un module de communication classique a donné lieu à des développements récents dans ce domaine. L'évolution des capteurs sans fil et l'augmentation des cas d'utilisation des RCSF a également mis en évidence la nécessité d'une telle fonctionnalité. Quatre systèmes sont présentés dans ce qui suit.

3.2.3.1 Le système Capsule

La nécessité de disposer de technologies pour le stockage non volatile de données est illustrée par les nombreux domaines d'application des RCSF. Ceux-ci vont du simple archivage de données jusqu'au stockage de composants logiciels utilisés pour une complète reprogrammation du capteur. De plus, les récentes innovations au niveau des mémoires Flash ont permis de réduire l'énergie nécessaire à leur utilisation et font d'elles une alternative non

négligeable à la communication. Dans un capteur sans fil sans unité de stockage non volatile, toutes les données qui ne sont pas transmises seront perdues en cas de panne et de redémarrage du système d'exploitation. L'utilisation de la mémoire Flash offre donc la possibilité soit de retarder l'envoi des données pour éviter, par exemple, la congestion du réseau soit de n'envoyer au fur et à mesure que les données demandées par l'utilisateur. Ce dernier peut toujours accéder, s'il le souhaite, aux anciennes données collectées et non transmises stockées en mémoire Flash. Le système Capsule a été développé à partir de ces différents constats [Mathur 2006b] en se focalisant sur les mémoires de type « NAND » moins consommatrices d'énergie que celle de type « NOR »

La première particularité du système Capsule est que les données ne sont pas stockées uniquement dans des fichiers mais également dans des structures de données plus ou moins complexes. Les formats de stockage de base sont les suivants, classés par ordre de complexité au niveau de leur gestion :

- les piles ;
- les files ;
- les flux ;
- les tableaux.

La présence de ces différents formats de stockage s'explique par le mode de fonctionnement associé généralement aux RCSF qui consiste à collecter des informations à une fréquence donnée. Le système Capsule fait partie des systèmes de fichiers dont la politique de gestion du stockage des données est inspirée de celle associée à la gestion des logs (« log-structured file system ») [Rosenblum 1991]. Schématiquement, l'ensemble de la mémoire Flash est considéré comme un fichier de logs où les données représentant les différents événements observés sont ajoutées les unes après les autres. Une donnée insérée ne peut être modifiée, elle peut être uniquement effacée. Un fichier peut être utilisé pour le stockage avec ajout, en début ou en fin, des données nouvellement recueillies. Si l'étape de collecte ne pose pas de problèmes majeurs, celle de recherche de données avec certaines caractéristiques est plus complexe que dans le cadre de l'utilisation d'un tableau, d'une liste ou d'une file.

Les piles sont la structure de données à la gestion la plus simple. Le système dispose d'un pointeur vers le dernier élément inséré. Chaque pile a, dans son en-tête, un pointeur vers l'élément précédent. L'empilement se résume donc à indiquer l'adresse de l'élément précédent dans la pile dans l'en-tête du nouvel élément inséré, à stocker ce dernier dans un emplacement de mémoire Flash libre et à mettre-à-jour le pointeur courant du système. Le dépilement consiste à modifier le pointeur courant du système.

Les files sous le système Capsule ont un fonctionnement très proche de celui des piles, l'opération d'empilement étant identique. La modification de l'en-tête des éléments déjà insérés pour qu'ils puissent pointer sur les nouveaux éléments n'est pas conseillée sous peine d'user prématurément la mémoire Flash. Par conséquent, le dépilement est plus complexe et oblige à traverser, à chaque fois, l'ensemble des éléments d'une file à la recherche du premier élément.

Les flux sont des structures hybrides combinant une pile et une file c'est-à-dire autorisant le parcours des données dans un sens comme dans l'autre.

Sous le système Capsule, les tableaux sont construits par regroupement d'un ensemble fixe d'index. Ces derniers sont constitués d'une donnée et, d'une clé prise dans un intervalle fixé au moment de la compilation. L'utilisation d'index permet d'obtenir une hiérarchie à plusieurs niveaux. Le premier niveau est pour les données, le deuxième ainsi que les suivants sont établis à partir d'agrégation d'index.

Ces différentes structures peuvent être combinées pour en créer deux autres entités de stockage plus élaborées :

- les flux de données indexés ;
- les fichiers.

Les flux de données indexés sont l'application du principe d'index à des éléments d'un flux plutôt qu'à des données. Il est ainsi possible de marquer un élément du flux représentant un événement particulier observé pour pouvoir y accéder plus rapidement.

Les fichiers sont également obtenus à l'aide d'index définis suivant plusieurs niveaux. Chaque fichier est découpé en bloc pointé par un index. Les données ajoutées sont associées au fichier par l'intermédiaire des index. Le nombre de niveaux d'index pour un fichier donné dépend de sa taille.

Chaque structure de stockage possède une interface avec l'ensemble des opérations que l'on peut leur appliquer. Le stockage de données directement dans la mémoire Flash sans passer par ces structures est possible.

Différentes couches sont définies au sein du système Capsule (voir Figure 3.17) :

- la couche d'abstraction de la mémoire Flash ;
- la couche « objet » ;
- la couche « application ».

Les méthodes d'accès et de gestion bas niveau de la mémoire Flash font partie de la première couche. Celle-ci fournit les primitives pour utiliser le périphérique matériel qu'est la mémoire Flash. En outre, des techniques de préservation de la mémoire Flash sont requises. L'emploi de la programmation différée de la mémoire en fait partie. Toute programmation (ou écriture) passe par un buffer intermédiaire dont la taille est variable et n'est donc pas obligatoirement identique à celle d'une page. En revanche, la lecture s'effectue directement sur la mémoire Flash sans utilisation de buffer supplémentaire.

Le système Capsule est muni d'un dispositif de nettoyage de la mémoire qui s'appuie sur 3 principes visant à limiter les ressources consommées. Tout d'abord, le nettoyage n'est déclenché que lorsqu'il est vraiment nécessaire et est donc retardé jusqu'au dernier moment. Pour une application peu consommatrice de mémoire non volatile, un dispositif de nettoyage fonctionnant en permanence et en tâche de fond a peu d'intérêt. En revanche, si un tel dispositif est nécessaire, il faut l'exécuter avant que la mémoire Flash ne soit trop utilisée et n'implique des traitements trop lourds. De plus, une fonction a été élaborée pour chaque type d'objet ou de structure de données pour permettre leur regroupement (« data compaction »). L'objectif est de pouvoir diviser la mémoire en deux parties avec d'un côté les emplacements occupés par des données et de l'autre les emplacements vides. Enfin, pour le nettoyage proprement dit, une technique simple est privilégiée par rapport à d'autres plus sophistiquées. Elle consiste à parcourir une seule fois la mémoire Flash pour marquer les blocs anciens ou inutilisés de manière à pouvoir les effacer ensuite en temps voulu.

La couche « objet » contient l'ensemble des structures permettant la gestion des données et qui ont été présentées ci-dessus. Pour en faciliter la gestion, chaque objet composé d'une donnée et de sa structure de stockage associée est muni d'un en-tête. Les informations contenues dans cet en-tête sont, par exemple, la taille de l'objet et un mécanisme de contrôle d'erreurs de type « checksum ».

Dans cette couche, un mécanisme de recouvrement après panne de type « checkpoint/rollback » est également présent. La mémoire Flash a, de par ses

caractéristiques, un fonctionnement de type statique. Cela correspond à la fois à un avantage et à un inconvénient dans un processus de recouvrement après panne. Un avantage car toute donnée supprimée ne le sera pas immédiatement. Il faudra attendre une prochaine écriture à l'emplacement même cette donnée pour qu'elle soit supprimée réellement. Tant que cette suppression n'est que virtuelle, la donnée est toujours accessible au prix d'un parcours d'une partie plus ou moins grande de la mémoire Flash. Cette qualité fait également office d'inconvénient. En effet, en cas de perte des informations systèmes, la récupération des seules données valides est complexe voire impossible. Le mécanisme de « checkpoint » s'occupe donc de sauvegarder dans la mémoire Flash un instantané des informations système présentes en mémoire SRAM.

La couche « application » regroupe l'ensemble des applications qui requièrent l'utilisation de la mémoire Flash. Les traitements de type agrégation ou non des données, le protocole de routage, la reprogrammation complète du système en sont des exemples.

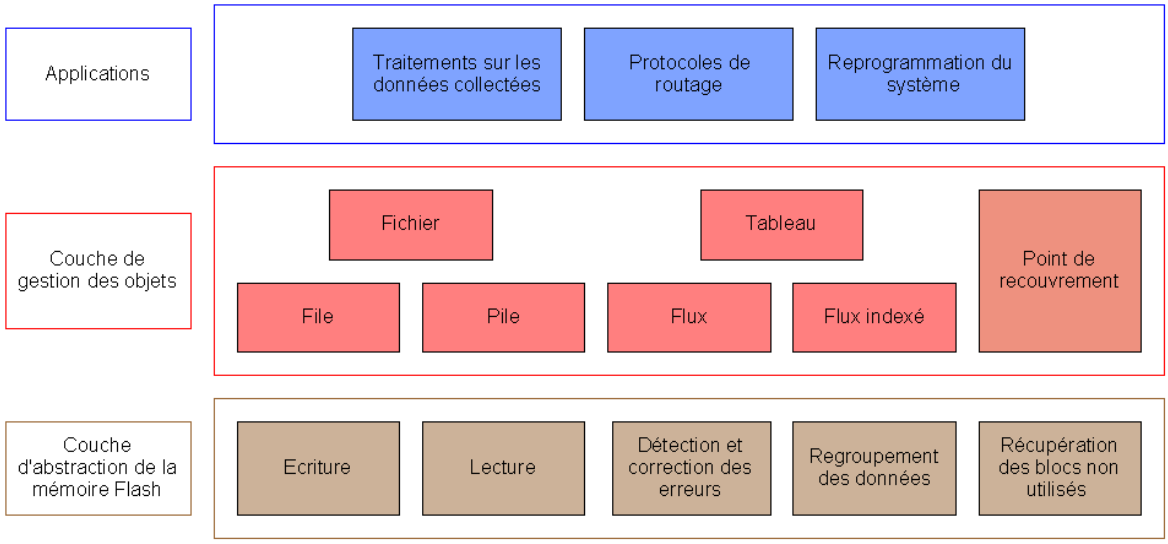

Figure 3.17 – Les différentes couches du système Capsule [Mathur 2006b]

3.2.3.2 Le système ELF

L'approche adoptée par le système ELF (Efficient Log-structured Flash File system) est différente de celle du système Capsule même si ce système utilise une politique de stockage de type gestion des logs [Dai 2004]. Dans le système ELF, la notion de fichier est très proche de celle présente dans les systèmes UNIX. Un log dans le système ELF est composé d'une séquence de nœuds physiques dont le type indique s'il s'agit d'un répertoire (« ELF_DIR »), d'un fichier (« ELF_FILE ») ou d'une opération portant sur les deux précédentes entités citées (« ELF_COMMON »). Les opérations disponibles pour les fichiers sont :

- la création ;
- l'ouverture ;
- l'ajout de données en fin ;
- l'écriture (ou modification) ;
- la lecture (ou recherche) ;
- le renommage ;
- la suppression ;
- la troncation.

Ces opérations sont celles classiquement associées à la gestion de fichiers ou de répertoires. Une différence existe au niveau de la troncation. Celle-ci dans le système Capsule vise non pas à supprimer les données en fin de fichiers mais celles en début qui sont les plus anciennes. Habituellement, une entrée de log ou nœud physique devrait être créée pour chaque opération sur la mémoire (voir Figure 3.18). Or, pour ne pas trop fragmenter la mémoire Flash, les données issues d'opérations d'ajout successives peuvent être regroupées dans une même entrée. Plus précisément, quand une nouvelle programmation est requise, la première étape consiste à rechercher un log existant permettant l'ajout de données.

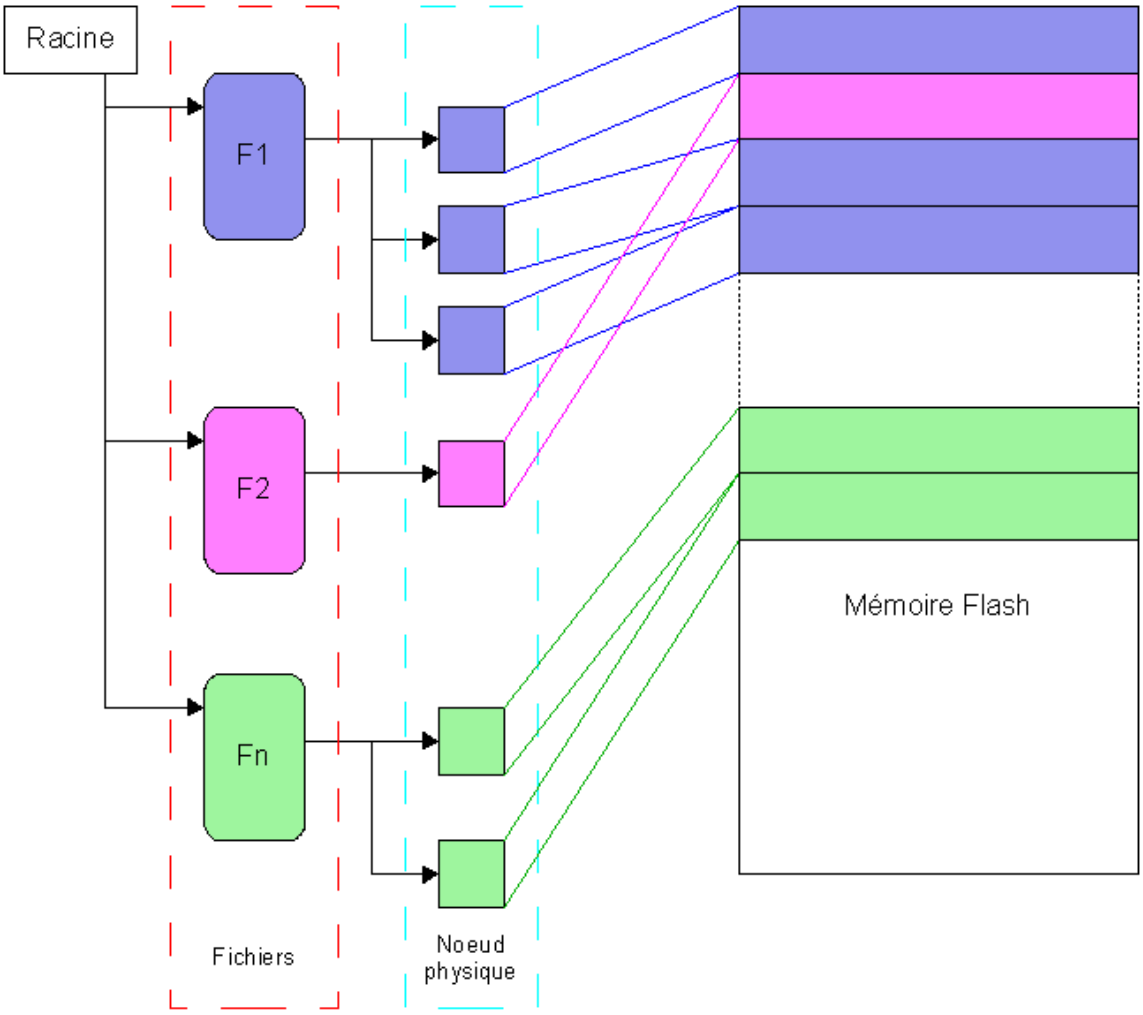

Figure 3.18 – La notion de nœuds physiques dans le système ELF [Dai 2004]

Le système ELF a été élaboré en considérant trois grands types de données que l'on retrouve généralement dans les applications de RCSF :

- les données collectées ;
- les données de configuration ;
- les images binaires pour la reprogrammation.

Des traitements et des niveaux de fiabilité différents sont requis pour chacun de ces types de données. Les données collectées sont issues du processus d'acquisition s'appuyant sur les capteurs de grandeur physique. Elles sont recueillies à une fréquence donnée et sont

stockées séquentiellement, les données les plus anciennes pouvant être écrasées par d'autres plus récentes.

Les données de configuration sont des informations système pouvant subir des modifications durant le déploiement du capteur. La durée séparant deux modifications n'est pas fixe et est assez grande. Ce type de données requiert un niveau supérieur de fiabilité à celui des données collectées.

Pour la reprogrammation du système, des images binaires sont utilisées. Ces images peuvent contenir l'ensemble ou des parties du nouveau système. Dans tous les cas, la transmission et le stockage de ces données ne doivent pas donner lieu à des erreurs qui entraineraient un arrêt complet du système et une inutilisation du capteur sans fil.

Une reprogrammation du système étant une opération peu courante, un mécanisme optionnel a été développée pour garantir l'intégrité des données d'un fichier. Différents procédés permettent de préserver la mémoire Flash. Les modifications apportées à des données existantes sont, d'une part, stockées dans une page différente. D'autre part, chaque page est munie d'un ensemble d'attributs dont fait partie le nombre de programmations subies. Grâce à cette information, le module dédié au nettoyage de la mémoire Flash peut décider de préserver certaines pages en ne les remettant pas directement dans le circuit des pages libres. Comme dans le système Capsule, ce module est appelé quand le nombre de pages libres descend en dessous d'un certain seuil.

Un attribut est dédié à la détection et à la correction des erreurs pouvant apparaître dans une page. Un autre correspond à un pointeur stockant l'adresse de la page suivante associée, par exemple, au même fichier. Ce mode de fonctionnement implique parfois une double écriture de certaines pages : la première pour l'écriture des données, la seconde pour modifier le pointeur suivant de la page.

Le système ELF dispose d'un mécanisme de recouvrement après une panne à plusieurs niveaux. Le premier effectue un instantané du système à un moment donné et consiste à sauvegarder l'état des nœuds physiques. Il s'agit d'un mode de fonctionnement de type « checkpoint/rollback ». De cette manière, ne sont pris en compte que les états stables et consistants associés à chaque élément. Le second niveau permet d'avoir également des informations sur les opérations en cours d'exécution au moment où la panne a lieu. Ainsi, les traces d'opérations mêmes incomplètes sur un fichier sont conservées. Dans le meilleur des cas, il est ainsi possible de poursuivre des opérations qui n'ont pas pu aboutir à cause de la panne.

Pour un fonctionnement optimal du système ELF, l'utilisation de mémoire de type EEPROM est préconisée en plus de celle de type FLASH pour le stockage non volatile de données (voir Figure 3.18). Le principal intérêt des mémoires EEPROM mis en évidence est une durée de vie supérieure à celle des mémoires de type FLASH. Cependant, l'arrivée de mémoire Flash de nouvelle génération autorisant plus d'un million d'écritures pourrait remettre en question l'utilisation des mémoires EEPROM [AMD 2001]. Cette configuration est adaptée à une implémentation du système ELF dans un capteur sans fil « mica2 ».

3.2.3.3 Le système MicroHash

Dans le système MicroHash, les aspects indexation et recherche de données sont considérés au même titre que la gestion de la mémoire Flash [Zeinalipour-Yazti 2005]. Le système propose différents mécanismes pour la gestion de la mémoire Flash de type « NAND » et pour l'interrogation des données stockées. L'approche adoptée consiste à collecter à la source le plus grand nombre de données mais de ne transmettre que les plus

intéressantes ou importantes pour l'utilisateur situé à l'autre bout de la chaîne d'acquisition. L'énergie des capteurs est ainsi économisée en évitant les transmissions inutiles de données.

Des hypothèses sont définies au niveau des données collectées. Celles-ci sont collectées à un instant donné et stockées sous forme d'enregistrements de type numérique avec des valeurs situées dans des intervalles connus. En règle générale, les données générées par un capteur de grandeur physique ont un format fixe. Par conséquent, la taille des enregistrements peut également être considérée comme fixe.

La mémoire Flash est utilisée comme une file circulaire. Les programmations commencent au début de la mémoire. Les données sont ainsi insérées progressivement et sont donc triées automatiquement par date d'acquisition. Quand la mémoire est pleine, la programmation reprend au début de celle-ci. Ce mode de fonctionnement est obtenu à l'aide de deux compteurs. Le premier « idx » joue plus un rôle de pointeur vers la page courante à programmer. Il est initialisé avec le numéro de la première page pouvant être programmée et est ensuite incrémenté au fur et à mesure des écritures. Le second « cycle » indique le nombre de fois où le compteur « idx » est remis à zéro ce qui correspond au nombre de fois où la mémoire Flash a été complètement utilisée. Ainsi, les écritures sont réparties uniformément entre toutes les pages. De plus, comme pour les autres systèmes présentés précédemment, un buffer de mémoire vive est utilisé pour accumuler une certaine quantité de données avant d'effectuer une programmation.

Ces techniques sont plus adaptées aux ressources limitées des capteurs sans fil en comparaison avec celles basées sur des adresses virtuelles. Une couche supplémentaire d'abstraction est ajoutée pour convertir une adresse réelle en adresse virtuelle. Après une modification, les données restent associées à la même adresse virtuelle mais sont stockées dans une autre page physique.

Chaque page est munie d'un en-tête de 8 octets avec les attributs suivants :

- Type de page (3 bits) ;
- Code de contrôle d'erreurs de type CRC (16 bits) ;
- Nombre d'enregistrements stockés dans la page (7 bits) ;
- Pointeur vers la page précédente (23 bits) ;
- Nombre d'écritures subies par la page (15 bits).

Les différents types de page sont : « Root », « Repertory », « Index » et « Data ». La partie « donnée » d'une page est spécifique au type de celle-ci. Les pages de type « Root » contiennent des informations relatives au fonctionnement de la mémoire Flash comme le type d'index utilisé, la position de la dernière écriture et l'adresse des différentes pages « Directory ». Ces dernières ont pour rôle la gestion des groupements de pages. Des répertoires de grande taille peuvent être obtenus grâce à l'attribut « pointeur vers la page précédente » des en-têtes de pages. Sous le système MicroHash, les index sont construits à partir des informations issues de pages « Directory » et « Index ». Un nombre fixe d'index vers les données ainsi que la date d'acquisition du dernier enregistrement indexé sont stockés dans les pages « Index ». Il est ainsi possible de se repérer plus rapidement durant les requêtes temporelles et de déterminer s'il faut se diriger vers la page précédente ou suivante. Les index sur les enregistrements sont de deux types avec ou sans « offset » mais comprennent tous les deux l'identifiant de la page où les données sont stockées. L'« offset » correspond au nombre d'octets séparant le début de la partie « donnée » d'une page du début de l'enregistrement indexé. Le code de contrôle des erreurs s'applique à la partie « donnée » de chaque page. Les pages « Data » contiennent un nombre fixé d'enregistrements. Au sein de la mémoire Flash, les pages de différents types sont intercalées les unes avec les autres.

Le fonctionnement du système MicroHash est divisé en quatre phases (voir Figure 3.19) :

- la phase d'initialisation (« The Initialization Phase ») ;
- la phase de croissance (« The Growing Phase ») ;
- la phase de répartition (« The Repartition Phase ») ;
- la phase de suppression (« The Deletion Phase »).

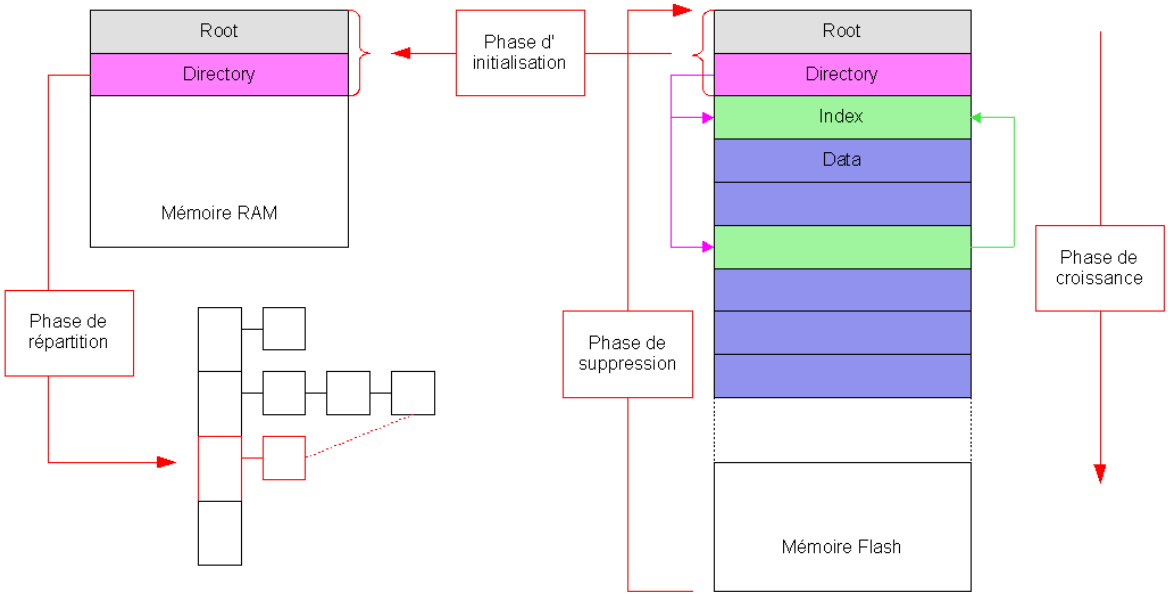

Figure 3.19 – Fonctionnement du système MicroHash [Zeinalipour-Yazti 2005]

La phase d'initialisation consiste à charger, dans la mémoire RAM, la page « Root » et une partie des pages « Directory ». Elles sont stockées en mémoire vivre pour en faciliter et accélérer l'accès. Elles sont sauvegardées périodiquement dans la mémoire Flash. La première étape consiste à localiser la page « Root » dans la mémoire Flash. En règle générale, la page « Root » est présente dans la « page 0 » de la mémoire Flash. Si la « page 0 » est arrivée en fin de vie, la page suivante est utilisée et ainsi de suite. Un bloc de pages est réservé en début de mémoire Flash pour stocker les pages « Root ».

La phase de croissance correspond à l'ajout progressif de pages « Data » et « Index » pour la gestion des données collectées. Les compteurs « idx » et « cycle » interviennent dans cette phase. A chaque écriture dans la mémoire Flash, un index est créé. Pour les enregistrements suivants {[1000,50], [1001,52], [1002,52]} au format « date, valeur », les index « ir = [idx, offset] » associés sont {[0,0], [0,12], [0,24]}. Les pages « Directory » servent à stocker l'adresse des pages « Index ». Chaque répertoire lié à une ou plusieurs pages « Directory » est découpé en plusieurs sous-ensembles appelés « buckets ». Ces sous-ensembles sont construits à partir des valeurs minimale « lb » (« lower bound ») et maximale « ub » (« upper bound ») potentiellement récoltables et un nombre défini au départ de classes de répartition des données. Chaque sous-ensemble est donc établi en découpant l'intervalle des valeurs possibles en sous-intervalles de même dimension. Si la répartition des données collectées est uniforme, chaque sous-ensemble aura quasiment le même nombre d'enregistrements à indexer.

Durant la phase de répartition, ces sous-ensembles sont ensuite redéfinis pour que les données soient, si nécessaire, mieux réparties. Les index avec leurs pages associées (pages « Index » et « Directory ») sont réorganisés durant cette phase. Les « buckets » répertoriant un grand nombre d'index par rapport aux autres sont, dans un premier temps, découpés. Plus

précisément, ceux stockant un nombre d'index supérieur à un seuil fixé par l'utilisateur sont, à leur tour, redécoupés. Dans un second temps, les « buckets » non utilisés depuis un certain temps sont stockés dans la mémoire Flash. Cela permet de libérer de la mémoire SRAM pour pouvoir stocker de nouveaux « buckets ».

La phase de suppression se résume en un appel au module ou fonction de nettoyage de la mémoire. Ce module est exécuté après chaque remplissage complet de la mémoire Flash et supprime les anciennes données, bloc par bloc, suivant les besoins. Ces suppressions sont répercutées au niveau des index grâce à un algorithme de recherche dédié. Il est également fait en sorte que les index qui pourraient être supprimés ne pointent pas vers des données valides.

Le système MicroHash permet d'indexer les données temporelles pour autoriser deux types de requêtes :

- les requêtes « égalité » (« equality queries ») ;
- les requêtes sur des intervalles de temps (« time-based range »).

Une requête « égalité » équivaut à comparer un attribut et une valeur donnée. La formalisation de ce type de requête est la suivante « Q(vi, a) » où « vi » est un attribut et « a » la valeur spécifiée pour le test. La requête « q=(temperature, 95F) » en est un exemple et vise à retourner les informations sur les enregistrements où la température est égale à 95F. Les requêtes sur des intervalles de temps sont définies de la manière suivante « Q(t,a,b) » où l'on va rechercher les enregistrements qui ont été collectés entre les instants a et b, avec la possibilité que a et b soient égaux.

En prenant en compte l'organisation propre au système MicroHash, les requêtes « égalité » impliquent différentes étapes. La première consiste à trouver le « bucket » approprié dans lequel on peut extraire l'adresse de la dernière page « Index » référençant les données répondant potentiellement à la requête exprimée. Puis, la suivante est la lecture de l'ensemble de ces pages « Index » en mode page par page pour trouver la ou les pages « Data » contenant les données recherchées. Enfin, l'extraction des enregistrements constitue la dernière étape.

Pour accélérer les requêtes temporelles, la recherche dichotomique a été implémentée sous deux différentes variantes appelées « LBSearch » et « ScaleSearch ». Dans la version « LBSearch », une borne inférieure « pessimiste » de l'intervalle de recherche est redéfinie récursivement jusqu'à atteindre la valeur souhaitée. Celle de type « ScaleSearch » exploite la connaissance sur la distribution des données des pages « Data » et « Index ». Quand la distribution des données est uniforme au niveau de la mémoire Flash, la méthode « ScaleSearch » est meilleure que celle de type « LBSearch ».

3.2.3.4 Le système TFFS

Les traitements sur la mémoire Flash du système TFFS (Transactional Flash File System) s'applique aux blocs plutôt qu'aux pages [Gal 2005]. Ces blocs correspondent au plus petit groupe de pages effaçables en une seule fois. Ces blocs sont appelés « unité d'effacement » (« erase unit »). Le système a été au départ développé pour utiliser la mémoire de type « NOR » embarquée dans la plupart des microcontrôleurs. L'objectif visé est la conception d'un système de fichiers adapté aux ressources limitées des RCSF sans être obligatoirement conforme à la norme POSIX. Les opérations sur les fichiers les plus courantes ainsi que d'autres non présentes dans la norme sont disponibles dans le système TFFS pour répondre aux contraintes rencontrées.

La principale particularité du système TFFS est la possibilité d'effectuer des opérations de lecture et d'écriture, soit directement, soit en passant par des transactions. Ces dernières sont utilisées d'une part pour gérer les accès concurrentiels aux données et d'autre part comme mécanisme de recouvrement après panne. La principale cause de panne mise en évidence est un manque ou une fluctuation brutale de l'énergie disponible.

Pour gérer les fichiers et les transactions, une interface de type API a été développée avec un large panel de fonctions parmi lesquelles :

- FD CreateFile(type, name, long_name, properties, parent_dir, tid) ;
- FD Open(parent, name, tid) ;
- int ReadBinary(file, buffer, length, offset, tid) ;
- int WriteBinary(file, buffer, length, offset, tid) ;
- int ReadRecord(file, buffer, length, offset, tid) ;
- int AddRecord(file, buff, length, tid) ;
- int CloseFile(file, tid) ;
- int DeleteFile(file) ;
- TID BeginTransaction() ;
- int CommitTransaction(tid) ;
- int AbortTransaction(tid).

Les fonctionnalités non supportées délibérément sont la troncation, le changement des attributs d'un fichier tel que son nom, le parcours des répertoires en utilisant des chemins relatifs. Trois types de fichiers sont présents sous le système TFFS :

- les fichiers « enregistrement » (« record ») ;
- les fichiers binaires ;
- les répertoires.

La gestion de ces différentes entités passe par une organisation particulière au niveau de la mémoire Flash et l'utilisation de structure de données à base d'arbres. L'unité d'effacement, considérée dans le système TFFS comme élément de base pour l'utilisation de la mémoire Flash, est divisée en quatre parties : l'en-tête, le tableau de descripteurs des secteurs, les secteurs et un espace libre situé entre les descripteurs et les secteurs (voir Figure 3.19). Un secteur correspond à un bloc mémoire de taille variable alloué au sein de l'unité d'effacement pour stocker des données. Le descripteur contient l'« offset » c'est-à-dire l'adresse de début du secteur qu'il référence ainsi qu'un bit « valide » et un autre « obsolète » qui permettent de spécifier l'état du secteur. Plus précisément, le bit « valide » indique si le champ offset a été correctement rempli et le bit « obsolète » que le secteur actuellement référencé est obsolète ou non.

Le système TFFS a été conçu pour pouvoir déplacer les données à la manière d'un mécanisme classique de défragmentation. Plusieurs unités d'effacement dont la plupart des données sont obsolètes peuvent être réquisitionnées comme espace de mémoire libre. Avant de pouvoir réutiliser ces unités, les données encore valides doivent être déplacées et regroupées dans une même et nouvelle unité. Au-delà du simple transfert d'octets d'un emplacement à un autre, se pose le problème du référencement des secteurs déplacés. En effet, que ce soit les fichiers d'enregistrements, binaires ou répertoires, ceux-ci sont construits à l'aide d'arbres qui pointent sur différents secteurs. Un secteur stocke soit des enregistrements soit des ajouts de données selon qu'il est associé à un fichier respectivement d'enregistrements ou binaire.

Les feuilles de l'arbre d'un répertoire contiennent, quant à elle, les métadonnées associées aux fichiers :

- l'identifiant du fichier au niveau du répertoire ;
- le type du fichier ;
- le nom au format long du fichier ;
- les droits d'accès au fichier.

Pour pallier ce problème, le système TFFS utilise des pointeurs logiques à la place de pointeurs physiques. Chaque pointeur logique comprend deux champs : un numéro logique d'unité d'effacement et un autre de secteur. Le premier numéro est utilisé comme index dans la table des unités d'effacement et le second comme index au niveau du tableau des descripteurs de secteurs.

Une table est utilisée pour stocker la correspondance entre unité d'effacement logique et physique. Elle masque donc les changements effectués durant les déplacements de secteurs. Cette table est stockée en mémoire de type soit SRAM soit Flash. Durant un déplacement, l'offset d'un secteur peut changer mais pas sa place dans le tableau des descripteurs. TFFS copie les secteurs non obsolètes dans des unités complètement vides en les collant le plus possible les uns aux autres de manière à libérer le plus d'espace libre. Ainsi, au sein de chaque unité d'effacement, les secteurs peuvent être répartis suivant qu'ils sont de nouveaux secteurs ou des secteurs recopiés. Deux indices au niveau des descripteurs ont été définis pour différencier les éléments de chaque catégorie. Les indices « lr » et « ln » concernent respectivement le descripteur du secteur recopié en dernier et le descripteur vers le nouveau secteur le plus récemment inséré.

Au niveau de l'organisation des fichiers, le système TFFS utilise une nouvelle structure de données appelée « arbres efficaces à plusieurs versions de recherche » (« efficient versioned search trees »). Cette structure a été élaborée à partir de celles des arbres persistants de recherche (« persistent search trees »). Un arbre est utilisé pour représenter les associations entre les différents fichiers comme on peut le retrouver dans le système UNIX. Au gré des opérations effectuées sur les fichiers, l'arbre subit des modifications (voir Figure 3.20).

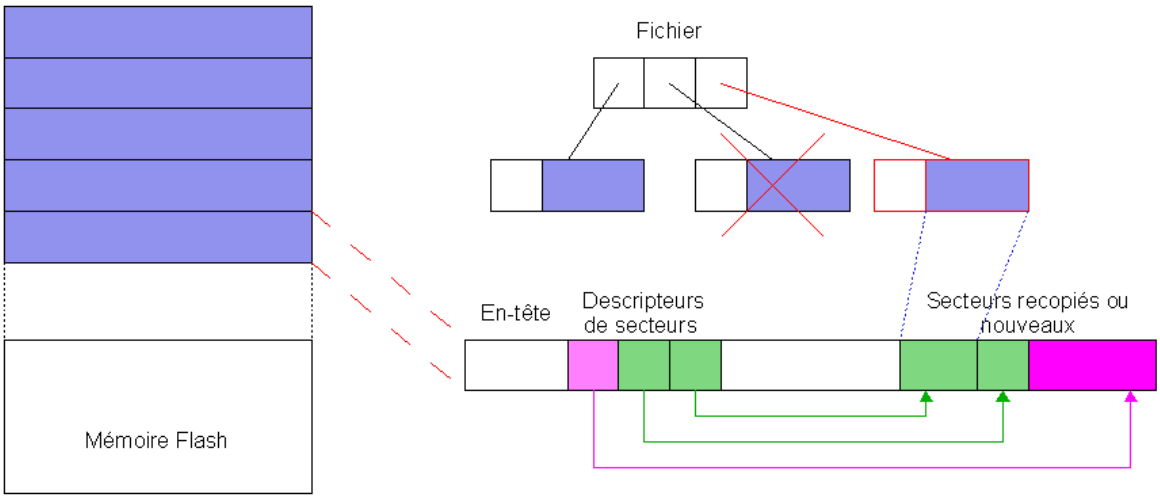

Figure 3.20 – Gestion d'un fichier d'enregistrements sous le système TFFS [Gal 2005]

Une version de l'arbre correspond à un état où un ensemble complet de modifications sur celui-ci a été validé. A partir de ce moment, la version courante de l'arbre devient accessible uniquement en lecture et une autre version est créée pour recueillir les nouvelles

modifications. La plupart du temps, seules la version courante et celle immédiatement précédente de l'arbre sont utilisées. La version courante correspond à un état où une transaction est en cours. De plus, pour réduire l'espace mémoire consommée, la version précédente peut être élaguée pour ne conserver que les toutes dernières modifications validées. On parle à ce moment-là d'arbres élagués à plusieurs versions (« pruned versioned trees »).

Chaque transaction dispose d'un identifiant TID (« Transaction IDentifier ») qui est fourni au moment de sa création. Cet identifiant correspond à un marqueur temporel. Plus cet identifiant est petit, plus la transaction associée est ancienne. Ce mode d'identifiant autorise le fonctionnement suivant : une transaction « t » peut utiliser les résultats des transactions validées allant de « 0 » à « t-1 » mais ne peut faire de même avec ceux issus des transactions supérieures ou égales à « t + 1 » validées ou non.

Chaque transaction crée une nouvelle version d'un arbre en le modifiant. Cette version reste active tant que la transaction n'a pas été validée (« committed ») ou considérée comme avortée (« aborted »). En attendant l'aboutissement de l'une ou l'autre situation, des pointeurs dits de rechange (« spare pointers ») sont utilisés pour stocker les modifications ayant eu lieu au niveau de l'organisation de l'arbre. Si la transaction échoue, ces pointeurs ne seront pas pris en compte. Des règles sont définies au niveau des interactions entre les transactions pour éviter l'accès en lecture ou en écriture à des données inconsistantes ou incohérentes. Pour faire respecter ces règles, trois identifiants de transaction sont liés et stockés dans une version d'arbre :

- l'identifiant de transaction de la dernière modification validée ;
- l'identifiant de la dernière transaction qui a effectué un accès en lecture ;
- l'identifiant de la transaction qui modifie actuellement l'arbre.

Un mode de fonctionnement sans transaction est également disponible. Les opérations sur les fichiers sont exécutées sans vérifier si elles ont abouti ou avorté. L'avantage de ce procédé est de nécessiter moins de traitements complexes et d'être plus direct.

Un mode de fonctionnement de type log est également présent au sein du système TFFS et peut contenir des informations sur les transactions, les opérations atomiques et les opérations de maintenance atomiques. Chaque entrée du log comporte les quatre attributs suivant :

- un bit « valide/obsolète » ;
- un identifiant construit différemment suivant le type d'entrée ;
- un identifiant de transaction ;
- un pointeur logique.

La présence des deux derniers attributs dépend du type d'entrées manipulé. La principale fonction du log est de répertorier les modifications effectuées par une transaction portant essentiellement sur la racine des différents arbres manipulés.

3.2.3.5 Bilan sur les comparaisons réalisées

Dans le cadre de la gestion de la mémoire, en particulier celle de type Flash, le système LiveFile dispose de la plupart de fonctionnalités présentes dans les autres systèmes (voir Table 3.1). En effet, le système LiveFile a été conçu en prenant en compte et en adaptant les travaux existants sur la gestion de la mémoire Flash dans les RCSF. Le système MicroHash propose, par exemple un stockage de type flux de données indexées proches du format de type « RECORD ». En revanche, la méthode utilisée sous le système MicroHash

pour répartir uniformément les programmations dans la mémoire Flash ne permet pas la conservation de données anciennes mais toujours intéressantes vis-à-vis de l'événement qu'elles représentent. Des fichiers classiques peuvent également être gérés sous le système LiveFile, les opérations disponibles étant moins nombreuses que celles définies dans le système ELF.

Caractéristiques	LiveFile	Capsule	ELF	MicroHash	TFFS
Type de mémoire	NOR	NAND	NOR	NAND	NOR
Techniques de préservation	-Programmation différée -Préservation des pages anciennes	-Programmation différée	-Préservation des pages anciennes	-Gestion de la mémoire comme une file circulaire	-Adressages physiques et logiques -Recopie des données modifiées
Nettoyage de la mémoire	-Etablissement d'un ensemble de pages libres -Module de nettoyage optionnel	-Regroupement de données -Mécanisme de nettoyage à déclenchement retardé	-Mécanisme de nettoyage à déclenchement retardé	-Nettoyage implicite par écrasement des données	
Types de données gérées	-Enregistrement -Fichier	-Pile, file, flux, tableaux -Flux indexés, fichiers	-Fichier	-Enregistrement -Groupe d'enregistrements	-Enregistrement -Fichier -Répertoire
Opérations sur les données	-Création, lecture, écriture (insertion), suppression	-Principales opérations associées à chaque type de données	-Création, suppression, ouverture, lecture, écriture -Ajout en fin, renommage, troncation	-Insertion, suppression, indexation	-Création, ouverture, lecture, écriture
Mécanisme de recouvrement après une panne	-Instantané du système	-Instantané du système	-Instantanée du système -Indications sur les opérations en cours		-Mécanisme complet avec gestion de transactions
Autres fonctionnalités	-Interrogation de données		-Contrôle des erreurs CRC	-Contrôle des erreurs CRC -Interrogation des données	-Organisation structurée des fichiers d'un répertoire
Empreinte mémoire	2Ko + 256 octets par buffers en plus + 8 à 16 octets d'en-tête par pages	2Ko 200 octets de SRAM	Non Spécifié	3Ko + buffers pour la gestion des index	0,2Ko de SRAM +6 octets d'en-tête par page

Table 3.1 – Comparaison des principaux systèmes de gestion de la mémoire Flash

Par défaut, les buffers de mémoire SRAM, servant au stockage intermédiaire des données avant leur programmation dans la mémoire Flash, ont une taille équivalente à celle d'une page. Cependant, comme dans d'autres systèmes, cette taille est configurable sous le système LiveFile suivant l'importance des données manipulées. Le système LiveFile a été élaboré de manière à minimiser les ressources consommées au niveau mémoire, puissance de calcul mais également énergie. Cet aspect est d'autant plus important qu'il a été implémenté pour gérer une mémoire de type « NOR », généralement plus consommatrice d'énergie que celles de type « NAND ».

Le système LiveFile dispose également d'un mécanisme performant de recouvrement après panne, bien que moins complet que celui de TFFS. Ce dernier n'utilise pas, contrairement à tous les autres, de buffers intermédiaires avant programmation et use donc plus rapidement la mémoire Flash.

Au final, le système LiveFile possède, d'une part, les fonctionnalités généralement requises pour permettre une utilisation et une gestion intelligente de la mémoire Flash. D'autre part, son association avec le système LIMOS offre un dispositif adaptable à différents types d'application de RCSF. Grâce à son architecture hybride, le système d'exploitation LIMOS offre un large panel de configurations permettant de répondre au mieux aux contraintes de performances rencontrées, le système LiveFile pouvant être également modelé suivant l'application supportée.

Dans cette section, les aspects liés à l'interrogation de données ont été introduits par l'intermédiaire du système MicroHash. La manière avec laquelle cette problématique est abordée dans le système LiveFile est présentée dans la partie 3.4.

3.3 La gestion avancée des données

Certaines applications de RSCF, de par leurs caractéristiques, autorisent l'emploi de techniques avancées améliorant la gestion des données collectées. Elles permettent principalement soit d'optimiser le stockage, soit d'accélérer la recherche de données, mais pas seulement. L'objet de cette partie est donc la présentation de plusieurs techniques avec leurs différents apports.

3.3.1 La compression de données

Pour obtenir un stockage plus compact, la première solution envisagée est la compression de données. Une telle fonctionnalité peut être requise par manque de mémoire ou pour archiver une partie des données qui doivent être conservées durant une période assez longue. Différents algorithmes de compression sont disponibles mais tous ne sont pas adaptés pas aux RSCF. L'objectif visé n'est pas uniquement de minimiser la mémoire utilisée. Il consiste également à réduire l'énergie consommée soit en limitant le nombre de programmations de la mémoire Flash soit en diminuant les quantités d'informations transmises.

Pour pouvoir être utilisé dans une application de RCSF, un algorithme de compression doit respecter un ensemble de contraintes associées aux ressources disponibles. La principale source de limitation est, a priori, la puissance et le temps de calcul nécessaire pour exécuter correctement l'algorithme [Sadler 2006]. Cependant, la mémoire requise pour stocker les calculs intermédiaires, ainsi que l'énergie consommée durant l'exécution de l'algorithme, sont également à prendre en considération [Puthenpurayil 2007].

Le choix de l'algorithme est également conditionné à la qualité des données compressées recherchées c'est-à-dire à l'utilisation d'une compression avec ou sans perte. Cette qualité dépend généralement du format des informations échangées. Pour des images, du son, voire des vidéos, la compression avec perte est, dans la plupart des cas, utilisée. En revanche, pour les flux ou les fichiers de données classiques, une compression sans perte est quasi obligatoire. Des exemples de techniques de compression avec perte sont :

- Compression d'images
  - JPEG (Joint Photographic Experts Group)
- Compression vidéo
  - MPEG (Moving Picture Experts Group)
- Compression audio
  - MP3 (MPEG-1 Audio Layer 3)

Quant aux techniques de compression sans perte, on peut citer :

- Compression d'images
  - PNG (Portable Network Graphics)
- Compression de flux de données
  - GZIP (GNU Zip) [Deutsch 1996a]

Ces dernières font appel à des algorithmes de compression sans perte tels que Huffman [Huffman 1952], LZW (Lempel-Ziv-Welch) [Welch 1984] ou DEFLATE [Deutsch 1996b]

Le système LiveFile a été développé pour répondre aux deux grandes catégories d'applications de RCSF :

- les applications d'acquisition de données ;
- les applications de substitution d'une infrastructure fixe de réseau.

Les applications de cette deuxième catégorie peuvent impliquer la transmission de flux audio et vidéo. Cependant, dans ce cas, le RCSF sert de support à la communication. Toute compression de données avec perte ne peut avoir lieu qu'au niveau des entités faisant appel au RCSF. Par conséquent, une étude a été menée pour équiper le système LiveFile d'un algorithme de compression sans perte.

3.3.1.1 Le codage par plages RLE

L'une des premières solutions envisagée est l'utilisation du codage par plages ou « run-length encoding » (RLE). Le principe de fonctionnement de ce codage consiste à considérer les séquences de « n » caractères identiques successifs. Plus ces séquences sont longues, plus le taux de compression est important (voir Figure 3.21). Il est généralement utilisé pour compresser des images. L'intérêt d'une telle méthode de compression pour les RCSF existe pour des applications de surveillance de grandeurs peu variables. Par exemple, pour détecter les incendies de forêt, des capteurs de température peuvent être utilisés et effectuer des relevés à des fréquences très élevées (toutes les minutes). Le nombre de valeurs successives identiques est, dans cette situation, très important.

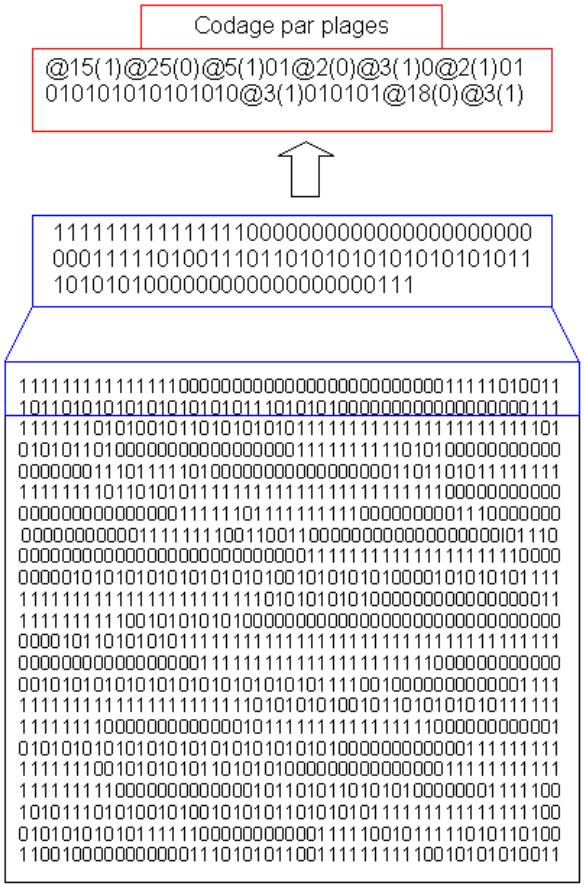

Figure 3.21 – Illustration du codage RLE

Cependant, certaines applications nécessitent de stocker, en plus de la valeur acquise, un ensemble de renseignements sur la date et le lieu de prélèvement. Pour ce faire, dans le système LiveFile, le type « RECORD » est, la plupart du temps, associé à une structure de données complète listant les différents attributs associés à la donnée acquise. La répétition de valeurs identiques successives en est donc assez limitée. Cette solution a l'avantage d'être l'une des moins consommatrice de ressources et de permettre de compresser, sans conversion, différents formats de données (caractères, chiffres, pixels, etc.).

3.3.1.2 Le codage d'Huffman

Le codage d'Huffman est une méthode de compression de type statistique. Son fonctionnement repose sur le principe suivant : plus un symbole est fréquent, plus son codage doit être court. Inversement, un symbole rare a un codage plus long. Par exemple, si l'on considère un texte en français, la lettre « e » aura un codage plus petit que la lettre « z ».

La première étape du codage d'Huffman consiste à déterminer la distribution des différents symboles présents dans l'élément à compresser. Elle implique la construction d'un arbre binaire dit d'Huffman. Chaque feuille de celui-ci contient un symbole avec son nombre d'occurrences faisant office de poids. Les nœuds de l'arbre sont ensuite créés, de manière itérative, par la réunion des feuilles de poids les plus faibles restantes (voir Figure 3.22). A chaque niveau de l'arbre, l'ordre d'assemblage est propre à l'alphabet considéré.

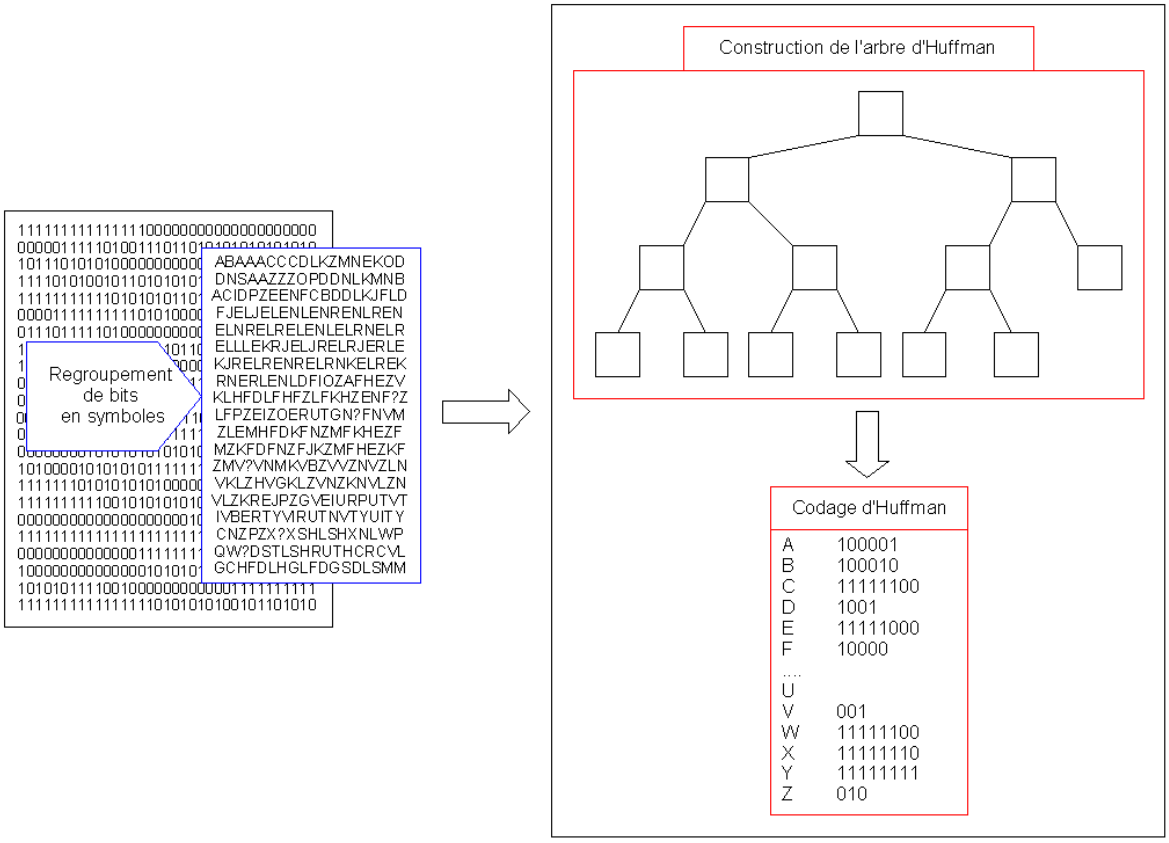

Figure 3.22 – Illustration du codage d'Huffman

Le codage d'Huffman existe sous 3 formes différentes suivant le mode de création de l'arbre choisi :

- création statique de l'arbre ;
- création semi-adaptative de l'arbre ;
- création adaptative de l'arbre.

Dans le premier cas, le codage de chaque symbole est connu dès le départ et est issu d'un arbre élaboré préalablement. Pour réaliser la compression et la décompression des données, il faut posséder la table de conversion dérivée de cet arbre. Pour la création semi-adaptative, un ensemble de données ou un fichier est pris comme référentiel pour construire l'arbre utilisé. Chaque nouveau flux de données reçues par la suite sera compressé à l'aide des informations issues de cet arbre. Le dernier mode implique la construction d'un arbre pour chaque ensemble de données à compresser.

3.3.1.3 L'algorithme LZW

L'algorithme LZW est une méthode de compression utilisant un dictionnaire qui est totalement ou en partie construit dynamiquement. Généralement, le dictionnaire contient au départ le code des symboles simples. Il est ensuite progressivement enrichi à partir des codes créés pour compresser des mots composés de plusieurs symboles simples.

L'exemple le plus souvent utilisé pour présenter l'algorithme LZW est la compression d'une chaîne de caractères. Le dictionnaire possède au départ le code ASCII de tous les caractères simples allant de 0 à 255. Par conséquent, le prochain élément rencontré constitué d'un couple de caractères sera codé avec le nombre 256 (voir Figure 3.23). Un caractère au format ASCII étant codé sur 8 bits, avec cette méthode, deux caractères voire plus le seront sur 9 bits au lieu de 16. Au fur et à mesure de la rencontre de nouveaux mots, le codage sur 9 bits peut ne pas suffire. Ce problème requiert soit un bon dimensionnement du codage au préalable soit des mécanismes pour indiquer un changement de format au niveau du codage.

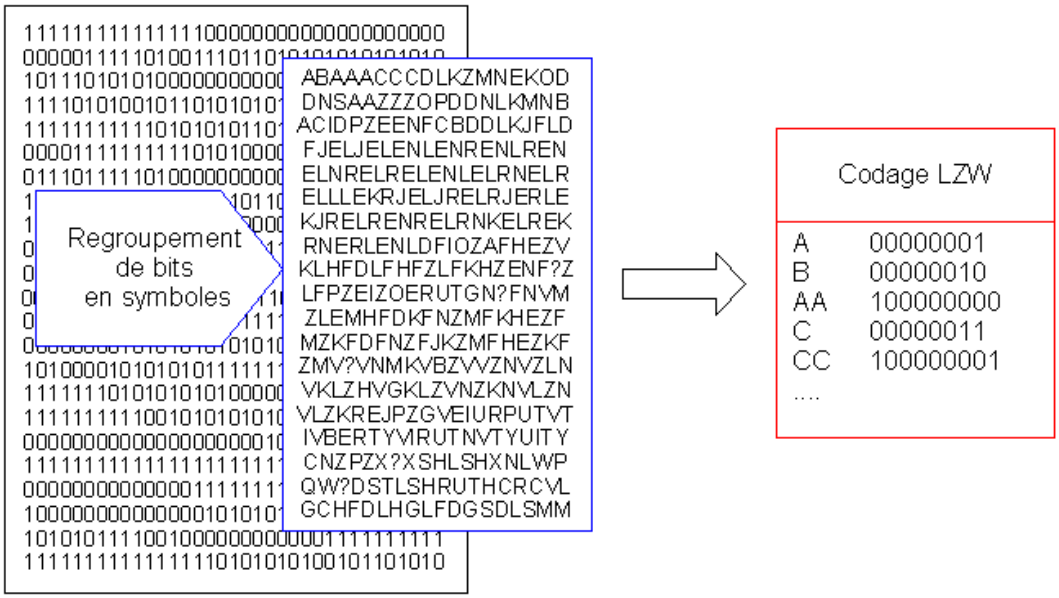

Figure 3.23 – Illustration de l'algorithme LZW

3.3.1.4 Bilan sur la compression de données

Les méthodes étudiées font appel à des techniques de compression de données sans perte assez différentes. En effet, par exemple, le codage de Shannon-Fano n'a pas été présenté car son principe de fonctionnement est proche de celui du codage d'Huffman. Chacun des algorithmes présentés dispose d'avantages et d'inconvénients par rapport à une utilisation dans un RSCF.

Pour comparer ces 3 méthodes, des tests de compression sur différents types de fichiers sont envisageables. Cependant, la méthode qui affichera les meilleures performances globalement, pourrait être pénalisée sur certains jeux de données. La notion d'entropie peut être utilisée pour déterminer l'algorithme avec potentiellement le taux de compression le plus performant [Shannon 1948]. Cette notion tend à estimer la quantité d'informations contenues dans un texte. Elle donne donc des indications sur la limite théorique de compression. De par son mode de fonctionnement statistique, le codage d'Huffman est celui qui se rapproche le plus de l'entropie.

Cependant, le critère du taux de compression n'est pas le seul à prendre en considération au niveau des RSCF. A ce titre, les inconvénients de la méthode d'Huffman sont l'utilisation d'un codage de taille variable, la nécessité dans certains cas de la

construction d'un arbre assez volumineux. De plus, l'arbre doit être transmis pour effectuer la décompression ce qui n'est pas le cas si l'on utilise le code ASCII pour initier l'algorithme LZW. Le code RLE ne nécessite, quant à lui, aucune transmission de dictionnaire ou de table de conversion (voir Table 3.2).

Algorithmes	**Avantages**	**Inconvénients**
Codage par plages	Puissance de calcul nécessaire	Faible niveau de performance
Codage d'Huffman	Niveau de performance élevé	Construction de l'arbre Diffusion du codage
Algorithme LZW	Niveau de performance élevé	Construction du code

Table 3.2 – Comparaison des différentes méthodes de compression de données

Toutes ces méthodes ont été développées dans un cadre général. Or, dans les applications d'acquisition de données, les grandeurs observées sont connues avant le déploiement du RCSF. Cette connaissance ainsi que certaines des caractéristiques des grandeurs observées sont utilisables pour optimiser la méthode de compression. Le contexte d'application peut donc intervenir dans l'élaboration du dictionnaire de codage.

3.3.2 Les métadonnées

Les métadonnées sont un concept de plus en plus étudié. L'approche abordée dans cette partie est centrée sur les RCSF. L'objectif visé est l'amélioration de la gestion des données au sein du système LiveFile à l'aide de métadonnées.

3.3.2.1 Définition et utilisation dans le système LiveFile

Les métadonnées sont des données ou des informations relatives à d'autres données. Le champ d'application de cette définition n'étant pas restreint, des travaux de recherche sur les métadonnées ont été initiés dans différents domaines. Beaucoup se rapporte logiquement aux systèmes de gestion de bases de données.

Avec la démocratisation du GPS ou d'autres outils de localisation, des informations sur le lieu et le moment de l'acquisition d'une donnée ont pu être récupérées en plus de la grandeur observée. La prise en compte et la gestion de ces métadonnées spatiales et temporelles est ainsi devenue une problématique très étudiée. Les aspects abordés se situent à la fois au niveau de la modélisation et de l'intégration de ces métadonnées dans des systèmes d'information géographiques [Gutiérrez 2007].

Dans [Huynh 2000], dans le cadre de l'utilisation des métadonnées dans les entrepôts de données, différentes classifications ont été proposées. La première consiste à classer les métadonnées suivant le moment où elles interviennent dans le processus de gestion des données :

- les métadonnées associées aux insertions de nouvelles données ;
- les métadonnées associées au stockage des données ;
- les métadonnées associées aux interrogations ou requêtes sur les données.

La première classe comprend les métadonnées qui donnent des informations sur les traitements effectués sur les données avant leur stockage. La donnée peut, par exemple, être corrigée pour prendre en compte l'incertitude du dispositif d'acquisition. La table et les vues

auxquelles la donnée appartient font partie des métadonnées de la deuxième classe. Enfin, les informations facilitant l'accès aux données au niveau de l'interprétation des requêtes appartiennent à la troisième classe.

Une autre classification sépare les métadonnées en 2 groupes :

- les métadonnées statiques ;
- les métadonnées dynamiques.

Les métadonnées statiques sont celles qui restent, a priori, identiques tout au long de la durée de vie de la donnée au sein de l'entrepôt. A l'inverse, celle de type dynamique sont modifiées suivant des événements prédéfinis. Un exemple de métadonnée dynamique est le nombre d'accès en lecture à une donnée. Cette information peut avoir son utilité que ce soit pour un entrepôt de données ou un RCSF. Les entrepôts de données grâce à la technologie OLAP (On-Line Analytical Processing) permettent des analyses complexes avec différents niveaux d'agrégation. Dans les entrepôts, les temps de traitements sont favorisés au détriment de la mémoire utilisée. Ainsi, pour accélérer la vitesse d'exécution des interrogations complexes, des prétraitements sont réalisés préalablement. Connaître et accélérer l'accès aux données les plus demandées peut donc représenter un avantage. Dans les RCSF, l'emploi d'un tel mécanisme ne peut se faire sans prendre en compte la consommation des ressources qu'il implique. Ces différentes classifications ne s'appliquent pas uniquement aux entrepôts de données. Elles peuvent l'être aussi aux systèmes d'information géographiques avec la valorisation des métadonnées spatiales et temporelles.

Au sein du système LiveFile, les métadonnées interviennent à différents niveaux car elles permettent de définir le contexte de l'application. L'association des systèmes LIMOS et LiveFile forment un dispositif adaptatif à l'application supportée. L'architecture hybride du système d'exploitation LIMOS est d'une part adaptée au fonctionnement de l'application. D'autre part, dans le système LiveFile, différents formats de stockage des données sont définis pour répondre également efficacement aux besoins de l'application.

En règle générale, les métadonnées fournissent des informations complémentaires sur les données. Elles apportent des précisions sur différents points. Dans le cas de données collectées, les informations contenues dans les métadonnées portent, par exemple, sur le lieu et la date d'acquisition. Les métadonnées définies de cette manière sont présentes au niveau du stockage de type « RECORD ».

Cependant, dans le cadre du système LiveFile, elles ne sont pas le seul type de métadonnées utilisées. Des métadonnées dites « contextuelles » ont été ajoutées. Dans les applications de RCSF, l'élément le plus important est l'information ou la somme d'informations contenues dans les données acquises plutôt que les données brutes en elles-mêmes. Pour surveiller une forêt vis-à-vis des départs d'incendies, des capteurs de température peuvent être déployés. L'objectif n'est pas de récolter une liste de températures mais de détecter une brusque élévation avec l'endroit où elle a été observée. A travers le RCSF, ne va transiter que l'information liée à un départ d'incendie ou au dépassement d'un certain seuil au niveau des températures. Les températures resteront, quant à elles, sauvegardées au sein de chaque capteur pour pouvoir, si nécessaire, interpréter les causes de fausses alertes. Ce principe peut être étendu au stockage des données. La grandeur « température » n'est pas stockée, seule l'information pour savoir si elle est supérieure ou inférieure à un certain seuil, un bit suffisant pour contenir cette information. Le but de l'utilisation des métadonnées contextuelles est le remplacement des données brutes par d'autres contenant les informations utiles et nécessitant moins de ressources à la fois pour le

stockage et la communication. Les métadonnées contextuelles vont être présentées plus en détails dans ce qui suit.

Différentes informations sont obtenues à partir de l'étape d'acquisition. Ces informations peuvent être modélisées au sein d'un système d'information à l'aide du modèle relationnel [Codd 1970]. Plus précisément, un ensemble d'attributs « A_i », i allant de 1 à n, est associé à chaque relation « r » constituant son schéma. Chaque attribut « A_i » a un ensemble de valeurs possibles « V_i » qui forment son domaine noté « $Dom(A_i)$ ». On définit également le domaine actif de « A_i » comme le sous-ensemble des valeurs prises par « A_i » dans « r » et on le note « $ADom(A_i,r)$ ». Le domaine actif de la relation « r » noté « ADom(r) » est :

$$ADom(r) = U_{Ai,\ i=1\ à\ n}\ ADOM(A_i,r)$$

Pour certaines applications de RCSF, le domaine actif d'une relation « r » peut être établi et utilisé pour configurer le système LiveFile avant le déploiement des capteurs. L'intervalle des valeurs relevées par un capteur de température positionné dans un champ peut être déterminé plus ou moins précisément. Les informations collectées sont amenées à être intégrées dans un processus d'aide à la décision. Les attributs représentant ces informations subissent donc des traitements pour répondre aux besoins des utilisateurs finaux. Ces traitements modifient le domaine actif de la relation « r ». Dans notre cas, l'objectif est d'élaborer des traitements représentant différents niveaux d'interprétation des données brutes afin de réduire la taille mémoire nécessaire pour stocker les informations issues du domaine actif d'une relation « r ».

En reprenant l'exemple de l'application de surveillance d'incendie, l'étape d'acquisition consiste à récolter les informations suivantes :

- « température » : la température acquise ;
- « localisation » : la position du capteur au moment de l'acquisition ;
- « datation » : la date de l'acquisition.

A un premier niveau d'interprétation, l'information « température » est contenue dans un attribut de type nombre réel. L'élément « localisation » est décomposé en plusieurs attributs suivant les coordonnées GPS obtenues. Finalement, une structure permettant de stocker une date au format « jour, mois, année, heure : minute : seconde » est utilisée pour l'information « datation ». Le niveau le plus haut pourrait correspondre pour l'information « température » à un intervalle, pour « localisation » à une zone et pour « datation » à une période de l'année ou de la journée (voir Figure 3.24).

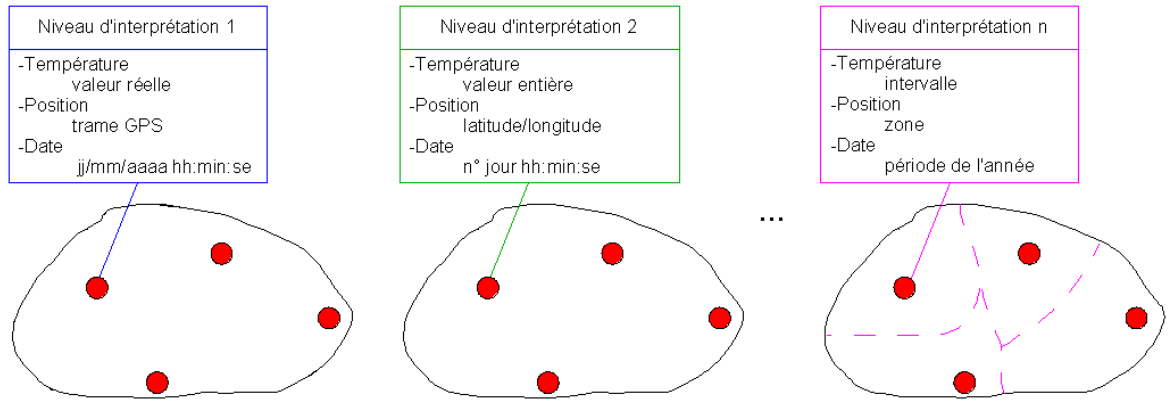

Figure 3.24 – Différents niveaux d'interprétation des données d'une application de surveillance d'incendie

Sur l'exemple de la figure précédente, pour la « température », 2 seuils préétablis avant le déploiement permettent de définir 3 intervalles codés sur 2 bits :

- « Intervalle1 (01) » ;
- « Intervalle2 (10) » ;
- « Intervalle3 (11) ».

5 zones géographiques sont définies :

- « zone A (001) » ;
- « zone B (010) » ;
- « zone C (011) » ;
- « zone D (100) » ;
- « zone E (101) ».

La période correspond au jour de l'année qui peut être codé sur 9 bits du « 1er Janvier (000000001) » au « 31 Décembre (101101101) ou (101101110) ». L'ensemble requiert 14 bits auxquels une méthode simple de correction d'erreurs telle que le contrôle de parité peut être ajoutée pour que chaque acquisition ait une taille de 2 octets. Pour des capteurs fixes, l'information sur la zone géographique peut ne pas être sauvegardées ou transmises systématiquement. La taille d'une acquisition est donc, dans certains cas, variable. Au final, le domaine actif des attributs associés aux informations est ainsi réduit par classification ou groupement des valeurs en différents sous-ensembles.

Dans le système LiveFile, le rôle des métadonnées a été étendu pour permettre l'élaboration et l'utilisation des différents niveaux d'interprétation liés aux attributs. Dans l'exemple précédent, si l'on considère une métadonnée « MTtemp » associée à la « température », elle indique quelles opérations doivent être réalisées sur cette grandeur (Figure 3.25). Ainsi, à chaque métadonnée contextuelle MT_k sont associés des traitements $T_k()$ allant d'un simple test sur une valeur à l'utilisation d'une fonction d'agrégation plus ou moins complexe.

```
SI MTtemp = 'T0' ALORS
        /* valeur simple */
        retourner(valeur)
FINSI

SI MTtemp = 'T1' ALORS
        /* valeur arrondie */
        retourner(arrondie(valeur)) ;
FINSI

SI MTtemp = 'T2' ALORS
        /* Association avec un intervalle */
        retourner(recherche_intervalle(valeur)) ;
FINSI
```

Figure 3.25 – Exemple de métadonnées

La relation suivante peut être définie, où DMT_k est la nouvelle donnée obtenue après l'utilisation des traitements $T_k()$ associés à la métadonnée MT_k sur la donnée D :

$$DMT_k = T_k(MT_k, D)$$

L'ensemble des couples (métadonnée MT_k, traitements associés $T_k()$) constitue le dictionnaire d'interprétation « Dict » :

$$Dict = \{(MT_k, T_k()), k \text{ allant de 1 à n}\}$$

Soit Qb() donnant la quantité de bits nécessaire pour représenter une donnée brute D de type entier, réel, chaînes caractères, etc. :

$$Qb(D) \rightarrow \mu$$

Avec le scalaire μ indiquant le nombre de bits

L'utilisation des métadonnées contextuelles pour le stockage est sujette à la contrainte suivante :

$$Qb(\bigcup DMT_k) + Qb(Dict) \leq Qb(\bigcup D), k \text{ allant de 1 à n}$$

Cependant, en stockant dès le départ le dictionnaire d'interprétation sur chaque capteur, il n'est pas nécessaire de l'envoyer à chaque transmission de données. Par conséquent, pour la transmission, la contrainte à respecter est la suivante :

$$Qb(\bigcup DMT_k) \leq Qb(\bigcup D), k \text{ allant de 1 à n}$$

Au fur et à mesure des applications supportées, le système LiveFile va être enrichi de nouvelles métadonnées contextuelles. Celles-ci pourront être réutilisées partiellement ou totalement au niveau d'applications au fonctionnement similaire. Le prélèvement à l'aide d'un RCSF de données sur la pression fait appel à des traitements que l'on retrouve dans une application d'acquisition de températures.

Pour réduire l'espace mémoire nécessaire au stockage des informations, l'utilisation des métadonnées contextuelles peut être couplée à celle d'un algorithme de compression. La ressource mémoire est ainsi privilégiée vis-à-vis des ressources processeur disponibles. Le déploiement du RCSF devra donc être précédé de l'élaboration du dictionnaire d'interprétation et d'une étude sur l'algorithme de compression le mieux adapté. Au final, deux modes de compression de données sont définissables :

- la compression à l'aide d'un algorithme ou d'un codage ;
- la compression par métadonnées.

3.3.2.2 La transmission des données

En équipant l'ensemble des capteurs avec un même dictionnaire d'interprétation, les quantités de données transmises sont réduites à chaque échange. Sachant que l'énergie consommée est quasi-proportionnelle au nombre de bits transmis, cette technique permet une économie non négligeable. On constate de nouveau que les ressources d'un capteur sans fil sont liées entre elles. Ainsi, préserver une ressource va généralement s'effectuer aux dépens d'une autre. Dans ce cas, économiser de l'énergie pour la transmission va s'accompagner d'une consommation plus importante de la mémoire pour stocker le dictionnaire d'interprétation. Les ressources de type processeur ou puissance de calcul seront également plus utilisées.

Sécuriser, au niveau communication, un RCSF est une opération complexe. La problématique associée est la conception d'une méthode offrant une protection performante tout en consommant le moins de ressources possibles. Les métadonnées contextuelles peuvent offrir un mécanisme de sécurisation des transmissions. Son action se situe au niveau de la confidentialité des données et non de l'authentification. La méthode élaborée s'appuie d'une part sur la non-distribution du dictionnaire d'interprétation. D'autre part, sur le fait que les

données obtenues après application des traitements associés aux métadonnées n'entre dans aucune logique par rapport aux informations contenues dans les données brutes. Schématiquement, une zone géographique peut être représentée par un caractère alors qu'une autre peut l'être par un entier.

Certaines métadonnées peuvent être dédiées à la mise-à-jour du dictionnaire d'interprétation. Tout au long du fonctionnement du RCSF, des modifications peuvent intervenir sur le dictionnaire de départ pour améliorer la robustesse de cette solution vis-à-vis de certaines attaques. La protection des données peut être accrue par la présence de plusieurs dictionnaires d'interprétation. Un capteur peut posséder certains ou la totalité des dictionnaires. Les données peuvent circuler à travers le réseau sans que tous les capteurs ne disposent de tous les dictionnaires. En revanche, la communication entre deux capteurs est conditionnée aux dictionnaires que dispose chacun d'entre eux. Des sous-groupes d'échanges peuvent être ainsi créés (Figure 3.26).

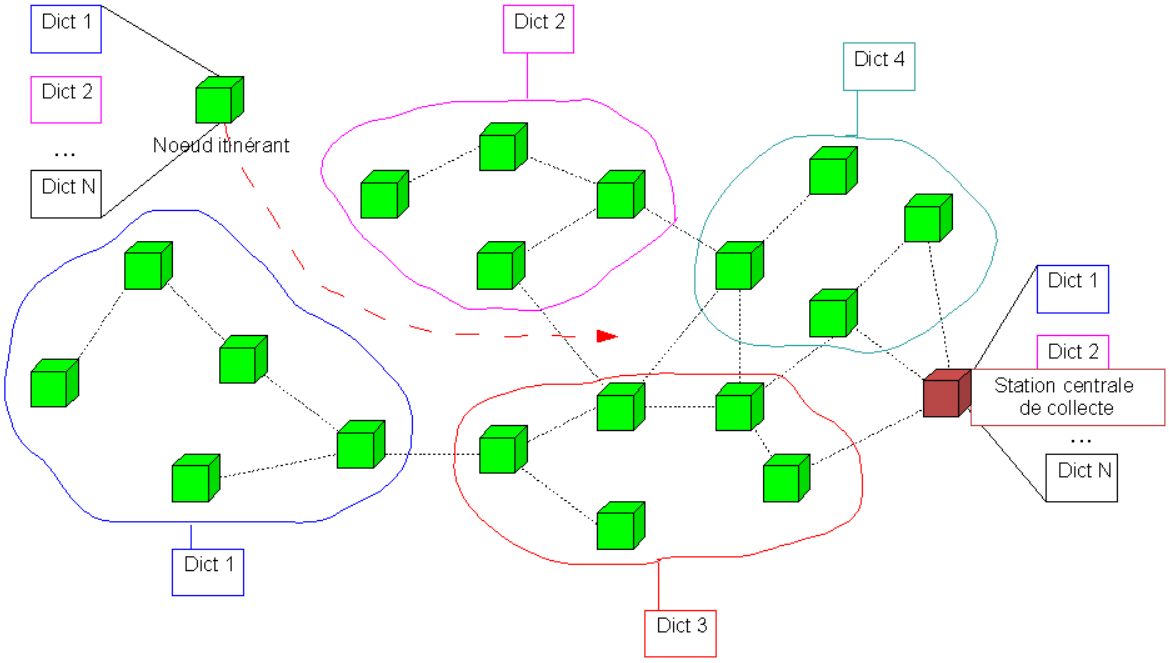

Figure 3.26 – Utilisation des métadonnées contextuelles dans la transmission

Les dictionnaires d'interprétation peuvent être stockés de différentes manières. La première solution envisagée est un stockage au même endroit que les données c'est-à-dire en mémoire Flash. Ainsi, en cas de panne du capteur, les données peuvent être directement récupérées à partir du capteur à condition de connaître l'emplacement du dictionnaire d'interprétation.

Une autre solution est le stockage en mémoire RAM avec copie temporaire en mémoire Flash en cas de redémarrage programmé. Le capteur est déployé avec le dictionnaire en mémoire Flash. Après l'avoir copié en mémoire RAM, il efface correctement celui-ci de la mémoire Flash. Quand le niveau d'énergie est bas, le capteur peut décider ou non de stocker le dictionnaire dans la mémoire Flash. Ce fonctionnement implique de disposer d'une copie du ou des dictionnaires d'interprétation utilisés qui serait, a priori, localisée dans la station centrale de collecte.

Des méthodes pour la transmission de données compressées existent [Hollenbeck 2004] [Friend 2004]. Elles combinent le protocole TLS (« Transport Layer Security ») et

l'algorithme de compression LZS (Lempel-Ziv-Stac). Ces méthodes sont assez complètes puisque le protocole TLS opère au niveau de l'authentification. Ce protocole repose sur une architecture client/serveur et nécessite une étape de négociation en plusieurs échanges, donc assez consommatrice d'énergie. Ces éléments font de ces méthodes des solutions complexes à mettre en place au sein d'un RCSF.

3.4 L'interrogation des données

Le système LiveFile a été, au départ, conçu comme une couche d'abstraction pour l'accès en écriture et en lecture à la mémoire Flash. Le concept de métadonnée contextuelle a ensuite été ajouté pour améliorer la gestion des ressources des capteurs sans fil. Ce système dispose également de fonctionnalités pour l'interrogation des données stockées qui vont être présentées dans cette partie.

3.4.1 Les systèmes existants

Différents travaux ont été menés pour développer des systèmes de gestion de bases de données (SGBD) dédiés au RCSF. Deux approches sont, au départ, envisageables au niveau des RCSF [Bonnet 2001] :

- l'approche centralisée ;
- l'approche distribuée.

L'approche centralisée est généralement illustrée par les entrepôts de données matérialisés. Dans une approche « entrepôts de données », les données produites et gérées par les sources que sont les capteurs seraient intégrées au sein d'une même base de données. Cette intégration est soit matérialisée soit virtuelle. Dans le premier cas, les données de chaque capteur sont recopiées dans une base de données centrale. Dans le second cas, une interface contenant des informations sur les données de chaque capteur avec des détails sur leur organisation fait office de base de données centrale.

Appliquée au RCSF, l'approche distribuée a quelques similitudes avec celle des entrepôts de données virtuels. D'autant plus si une station centrale de collecte est utilisée, non seulement pour interroger le RCSF, mais également pour centraliser des renseignements sur les données présentes dans les capteurs. Cependant, la station centrale est généralement en charge uniquement de récupérer le résultat des requêtes et n'intervient ni dans l'organisation du RCSF ni au niveau des échanges de données. Chaque capteur dispose de sa propre base de données et gère celle-ci à leur façon suivant leurs ressources disponibles. Les requêtes initiées à partir d'une des entrées du RCSF sont ensuite diffusées à travers le réseau pour être traitées par chaque capteur sans fil.

Au niveau des RCSF, l'approche distribuée est celle qui est majoritairement retenue comme l'atteste les deux principaux systèmes existants dédiés à l'interrogation de données que sont :

- le système TinyDB ;
- le système Cougar.

Ces systèmes vont être présentés en détails dans ce qui suit.

3.4.1.1 Le système TinyDB

Le système TinyDB fait partie des développements réalisés sur les RCSF par le projet WEBS (« Wireless EmBedded Systems ») initié par l'Université Berkeley de Californie [Madden 2005]. Ce système a été développé pour équiper le système d'exploitation TinyOS [Hill 2000] d'un module d'interrogation des données distribué. L'approche considérée initialement dans la conception du système TinyDB est que, pour réduire la consommation d'énergie d'un capteur sans fil, l'acquisition et l'interrogation de données doivent être gérées étroitement au sein d'un même système. Dans TinyDB, les données des capteurs sont stockées sous forme de tuple dans la table « sensors ». Cette dernière est en fait répartie à travers tous les capteurs du réseau. Le système TinyDB utilise une syntaxe basée sur le

langage SQL (Structured Query Language) pour l'interrogation. La possibilité d'effectuer des requêtes de type « SELECT … FROM … WHERE … GROUP BY … » implique la présence des opérateurs relationnels de sélection, de projection et de jointure auxquels des fonctions d'agrégation sont ajoutées.

Cette syntaxe a été étendue pour intégrer des traitements liés au processus d'acquisition des données. La clause « SAMPLE PERIOD f FOR d » permet d'acquérir des données à une fréquence « f » durant une durée « d ». Par exemple, la requête 1 (voir Figure 3.27) demande à chaque capteur ou nœud du RCSF d'effectuer des relevés de luminosité et de température toutes les secondes pendant une durée de 10 secondes et ensuite de transmettre ces informations à la station centrale de collecte. Un protocole simple de synchronisation temporelle est utilisé au sein du RCSF pour que l'étape d'acquisition débute au même moment au niveau de chacun des nœuds.

Le système TinyDB permet de définir des requêtes dont la fréquence d'échantillonnage sera déterminée suivant l'énergie disponible sur le capteur sans fil. Dans ce cas, l'utilisateur ne spécifie que la durée en heure, en jour voire en mois pendant laquelle la requête doit s'exécuter (voir Requête 2 de la Figure 3.27). La fréquence d'échantillonnage utilisée va être estimée à l'aide d'une formule ayant pour paramètres : le coût de récupération des données issues des capteurs de grandeur physique, les opérateurs utilisés dans la requête, le taux de transmission souhaité, le niveau de batterie actuel. Le rôle de relayeur que peut jouer le capteur pour ces voisins est également pris en compte. Au niveau du routage, l'hypothèse considérée est que les capteurs du réseau forment un arbre, la cadence de transmission étant donnée par la racine de celui-ci. L'utilisateur peut fournir une borne inférieure pour la fréquence d'échantillonnage. Si l'estimation donne une fréquence inférieure à cette borne, la transmission de données s'effectue à un rythme moins soutenu que celui de l'acquisition.

Pour ce type de requête, certaines opérations comme la jointure ou le tri ne peuvent être appliquées directement du fait du flux continu de données arrivant dans le nœud. Leur utilisation est donc interdite tant que l'acquisition de données n'est pas terminée. Pour palier à ce problème, le système TinyDB définit des *points matérialisés* (« materialization points ») qui consistent en des buffers stockant une quantité fixée de données issues d'un processus d'acquisition. La requête 3 (voir Figure 3.27) est un point matérialisé créé sur l'exemple précédent. Ces points matérialisés peuvent être ensuite utilisés dans d'autres requêtes avec la possibilité de faire des jointures (voir requête 4 de la Figure 3.27).

Les requêtes à exécuter sont généralement envoyées à partir de la station centrale de collecte matérialisée par un ordinateur personnel relié par un lien de communication sans fil au RCSF. Les rôles affectés à cette station centrale sont d'une part d'analyser et d'optimiser les requêtes de l'utilisateur avant leur transmission et d'autre part regrouper et afficher correctement les résultats de ces requêtes provenant des différents nœuds du RCSF.

Aux requêtes qui viennent d'être présentées, le système TinyDB en permet d'autres, réparties de la manière suivante :

- les requêtes d'agrégation simples ;
- les requêtes d'agrégation temporelles ;
- les requêtes basées sur les événements.

Les requêtes d'agrégation simples sont présentes au sein d'un SGBD classique. Sous le système TinyOS, elles ont été adaptées au fonctionnement des RCSF. La principale

différence réside dans le fait qu'une requête donne lieu à un flux d'enregistrement au format « (group_id, aggregate_value) » plutôt qu'une seule donnée.

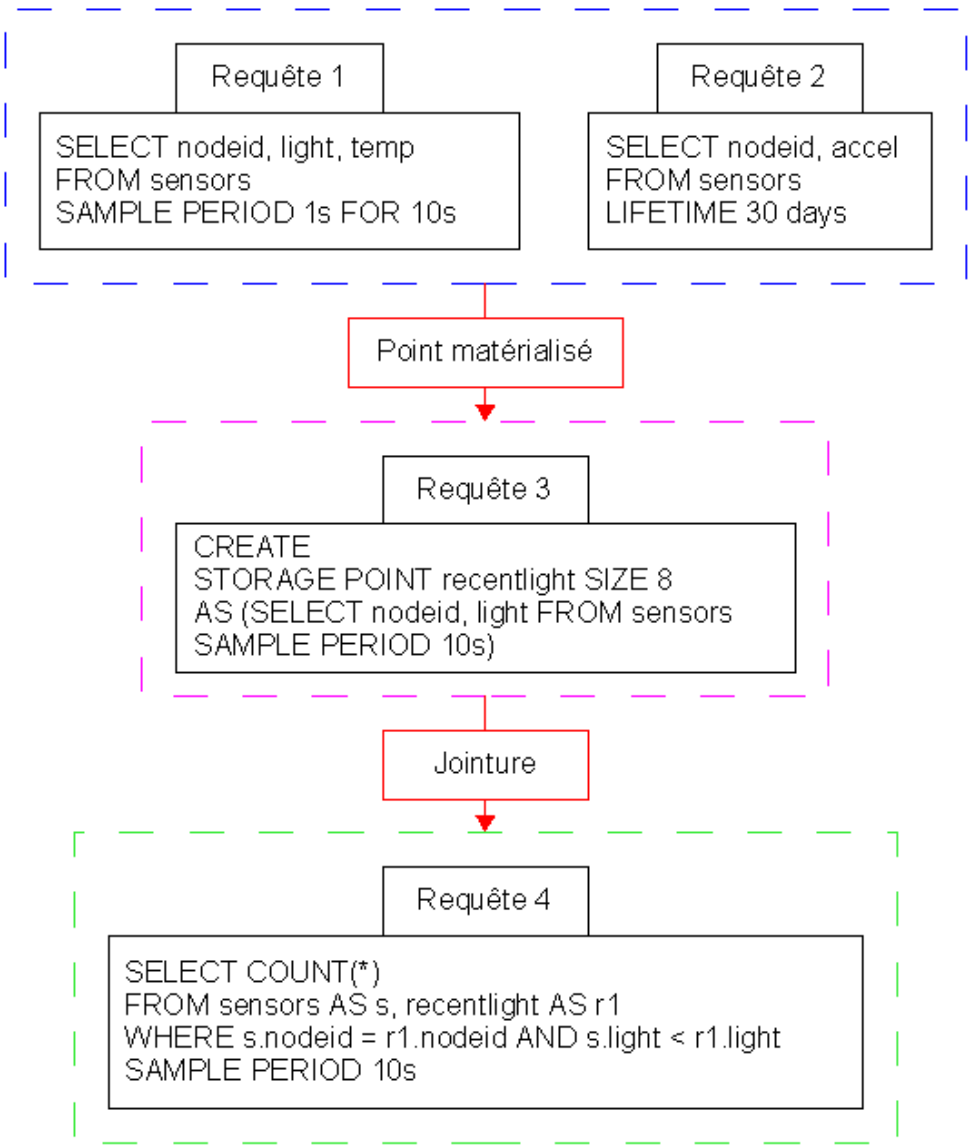

Figure 3.27 – Requêtes simples dans le système TinyDB [Madden 2005]

La synchronisation temporelle est de nouveau un élément important si l'on considère la requête 1 de la Figure 3.28. Celle-ci fournit le niveau sonore de chaque pièce situé au 6ème étage d'un bâtiment. Plusieurs capteurs étant déployés dans une même pièce, chaque relevé d'un capteur doit être réalisé au bon moment et ne doit être compté qu'une seule fois.

Au lieu que la requête tout entière soit soumise à des contraintes temporelles, seule la partie d'agrégation peut l'être. La requête 2 de la Figure 3.28 calcule la moyenne du volume sonore sur les dernières 5 secondes à partir de prélèvements effectués toutes les secondes et pendant une durée de 30 secondes. Cette requête construit, en fait, une fenêtre glissante de taille 5 secondes sur un intervalle de 30 secondes.

A partir du fonctionnement du noyau TinyOS, des requêtes basées sur les événements ont pu être définies au sein du système TinyDB. Elles sont de deux types. Le premier

regroupe les requêtes qui sont déclenchées par un événement. En plaçant des capteurs dans et autour d'un lieu habituellement fréquenté par les oiseaux, l'événement déclencheur pourra être la détection de la présence d'un oiseau (voir requête 3 de la Figure 3.28). Inversement, une requête peut servir à scruter des grandeurs physiques et générer un événement en cas d'observation d'une valeur particulière (voir requête 4 de la Figure 3.28). Ces traitements sont locaux même si, dans certains cas, l'événement généré peut être propagé par le système d'exploitation.

Une requête peut intégrer des commandes système permettant des actions directes sans passer explicitement par un événement. Par exemple, l'observation d'une température supérieure à un seuil donné peut entraîner la mise en marche d'un ventilateur (voir requête 5 de la Figure 3.28).

Des requêtes peuvent également être utilisées pour donner des informations sur l'état du capteur ou du réseau. Les requêtes sur la durée de vie utilisent des données sur le niveau de la batterie, attribut qui peut être de nouveau utilisé pour donner l'état des voisins d'un capteur (voir requête 6 de la Figure 3.28).

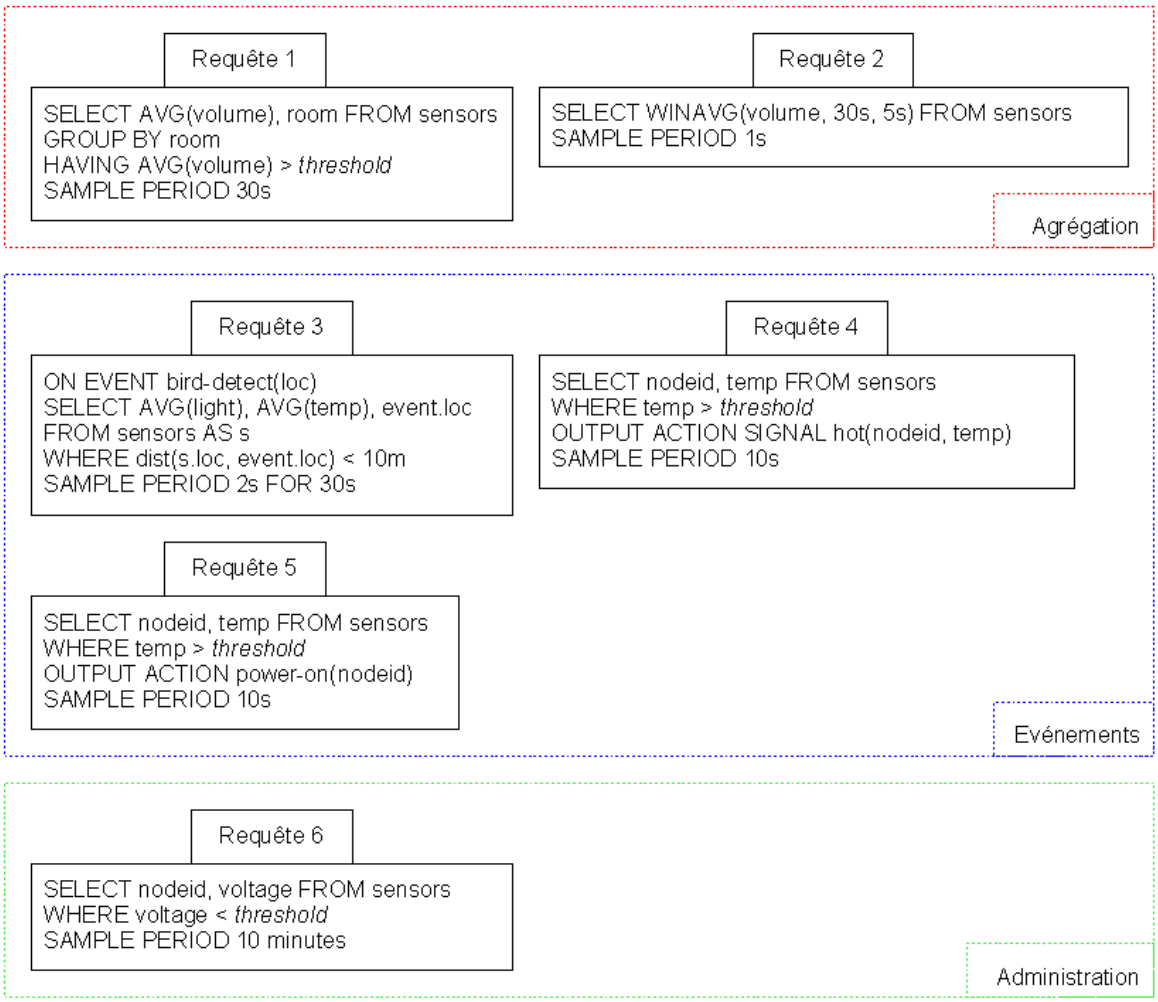

Figure 3.28 – Requêtes complexes dans le système TinyDB [Madden 2005]

Le système TinyDB autorise l'emploi de la valeur « NULL » pour indiquer l'absence d'une donnée collectée par un capteur. En revanche, les requêtes imbriquées telles qu'on les retrouve dans un SGBD classique ne sont pas disponibles sous le système TinyDB. Cette

limitation est liée au fait que le résultat d'une requête est un flux de données plutôt qu'une relation. Un contournement possible à ce problème est l'utilisation d'un point matérialisé au niveau de la sous-requête. Enfin, un identifiant est associé à chaque requête et permet d'interrompre celle-ci à tout moment via une commande de type « STOP QUERY id ». Le système TinyDB dispose ainsi d'un ensemble conséquent d'opérateurs pour réaliser différents traitements en relation avec l'interrogation des données collectées (voir Figure 3.29)

```
[ON [ALIGNED] EVENT    event-type [{paramlist}]
                                  [boolop event-type{paramlist} ... ]]
SELECT [NO INTERLEAVE] <expr> | agg(<expr>) | temporal_agg(<expr>), ...
FROM [sensors | storage-point], ...
[WHERE {<pred>}]
[GROUP BY {<expr>}]
[HAVING {<pred>}]
[OUTPUT ACTION [command | SIGNAL event({paramlist}) | (SELECT ... )]
[INTO STORAGE POINT bufname]
[SAMPLE PERIOD seconds
            [[FOR nrounds] | [STOP ON event-type [WHERE <pred>]]]
            [COMBINE { agg(<expr>)}]
            [INTERPOLATE LINEAR]]
[ONCE]
[LIFETIME seconds [MIN SAMPLE RATE seconds]]
```

Figure 3.29 – Ensembles des mots clés disponibles sous le système TinyDB [Madden 2005]

Dans chaque capteur sans fil, des métadonnées sont définies pour spécifier les attributs, les événements et les fonctions propres à l'utilisateur, utilisés dans le système TinyDB. Ces métadonnées sont intégrées de manière statique dans le système au moment de la compilation. Les métadonnées associées à un événement sont : un nom, un profil ou une signature, et une fréquence estimée de déclenchement. En ce qui concerne les attributs, les métadonnées donnent des informations réparties en deux catégories. La première comprend les renseignements servant à optimiser les requêtes. Ceux-ci sont le coût et le temps de récupération de l'attribut ainsi que l'intervalle des valeurs prises par celui-ci. La seconde catégorie est constituée des propriétés sémantiques des attributs qui interviennent dans les aspects agrégation et transmission des résultats issus des requêtes. Le système TinyDB est muni de différents mécanismes pour minimiser la consommation d'énergie au niveau des étapes d'acquisition, d'interrogation et de transmissions des données.

Le système TinyDB complète l'ensemble de ces fonctionnalités sur la gestion des données au sein de chaque capteur par d'autres en rapport avec la diffusion des requêtes et de leur résultat à travers le RCSF. Dans un souci de préservation de l'énergie consommée au niveau du réseau, la dissémination des requêtes doit être réalisée de manière intelligente et se limiter aux capteurs concernés. L'hypothèse considérée est que la majeure partie des requêtes envoyées contiennent soit l'identifiant du ou des capteurs concernés soit une localisation géographique.

Une technique appelée « arbres de routage sémantiques » (« Semantic Routing Trees ») a été élaborée. Son principe est de fournir assez de renseignements à chaque capteur pour qu'il puisse décider s'il doit ou non transmettre une requête à ces nœuds voisins. Cette décision est prise en se basant sur les critères de sélection de la requête. Les nœuds du réseau

sont organisés suivant une structure de type arbre. Le nœud racine à plusieurs nœuds fils qui sont, à leur tour, pères d'autres nœuds. Cette organisation se répète jusqu'à atteindre des nœuds sans fils qui sont les feuilles de l'arbre. Pour un attribut donné, un nœud parent va préalablement récupérer l'intervalle de valeurs disponibles au sein de ces nœuds fils. Quand arrivera une requête sur ce même attribut, chaque nœud père pourra, grâce aux informations qu'il détient, décider s'il doit ou non diffuser la requête à ses nœuds fils.

3.4.1.2 Le système Cougar

Le système Cougar part du principe que les données collectées au niveau d'un RCSF ne peuvent être gérées de la même manière que celles stockées dans un SGBD classique [Bonnet 2001]. Les données collectées d'un capteur sans fil définissent une vue matérialisée sur une période de temps donné plutôt qu'une relation à part entière. Une différence est donc faite entre les données stockées comprenant les informations sur le capteur sans fil et son fonctionnement, et les données collectées. Une base de données dite « de capteur » (« sensor database ») est en charge de gérer l'ensemble de ces données.

Le système Cougar manipule deux types d'objets pour la gestion et l'interrogation :

- les relations ;
- les séquences.

Les données stockées le sont dans des relations. Les données collectées sont elles associées aux séquences. Les requêtes relationnelles sont matérialisées par l'intermédiaire de la syntaxe SQL. Les opérateurs tels que la sélection, la projection et la jointure sont définis ainsi qu'un certain nombre de fonctions d'agrégation. Cette syntaxe, comme pour le système TinyDB, a été complétée pour prendre en compte les requêtes dites de séquences. Des opérateurs de séquence ont ainsi été ajoutés. A la manière des relations, le résultat d'une requête dite de séquence est une séquence. Certains opérateurs permettent le passage d'une relation à une séquence. Par exemple, l'opérateur relationnel de projection peut s'appliquer à une séquence et renvoyer une relation. La jointure entre une relation et une séquence est également possible, les comparaisons se faisant au fur et à mesure de l'acquisition de nouvelles données. Enfin, des traitements de type agrégation ou tri sur les séquences ne peuvent être réalisés qu'après la fin de la phase d'acquisition.

Les séquences sont alimentées par des données fournies par les différents capteurs de grandeur physique présents au niveau du RCSF. Ainsi, chaque type de capteur de grandeur physique est modélisé à l'aide d'un type d'abstraction de données spécifiques (« Abstract Data Type (ADT) »). A chaque abstraction, des fonctions retournant des données collectées sont associées. Les attributs associés aux ADT vont être utilisés au niveau des clauses SELECT, FROM et WHERE pour réaliser des requêtes sur des séquences plutôt que sur des relations. Un exemple de schéma d'une base de données de capteur est « R(loc point, floor int, s sensorNode) ». L'attribut « loc » est un ADT de type « point » stockant la position du capteur. L'entier « floor » indique l'étage où est positionné le capteur. Enfin, « sensorNode » est l'ADT d'un capteur de température. Cet ADT a des fonctions associées : « getTemp() » pour demander l'acquisition d'une température et « detectAlarmTemp(*threshold*) » pour renvoyer les températures qui dépasseraient le seuil « *threshold* ». Des exemples de requêtes sont fournis dans la Figure 3.30.

Dans certains cas, les attributs et les fonctions ADT ne peuvent être utilisés directement. Le fait que, généralement, les données collectées soient marquées temporellement ou horodatées constitue la première raison. Ainsi, dans le système Cougar, les

données collectées sont issues de fonctions de traitements du signal générant des valeurs au cours du temps. Le temps est divisé en intervalles discrets qui correspondent chacun à une position donnée. Ainsi, chaque donnée est affectée à la position associée à l'intervalle de temps durant lequel elle a été générée. Si plusieurs valeurs sont générées dans le même intervalle de temps, elles sont affectées à la même position. Ce mode de fonctionnement demande d'une part que les capteurs soient synchronisés temporellement. D'autre part, si une valeur n'est pas générée durant un intervalle donné, la valeur « NULL » est associée à la position liée à l'intervalle. Or, dans les requêtes fournies en exemple, le temps apparaît durant le processus d'acquisition mais n'est plus présent par la suite. L'autre principale raison est la nécessité de pouvoir conserver dans certains cas l'identifiant du capteur sans fil tout au long des étapes d'acquisition, de transmission et de présentation des données. Dans certains cas, l'utilisateur peut chercher à connaître l'identifiant du capteur qui a effectué des relevés qui lui paraissent intéressants.

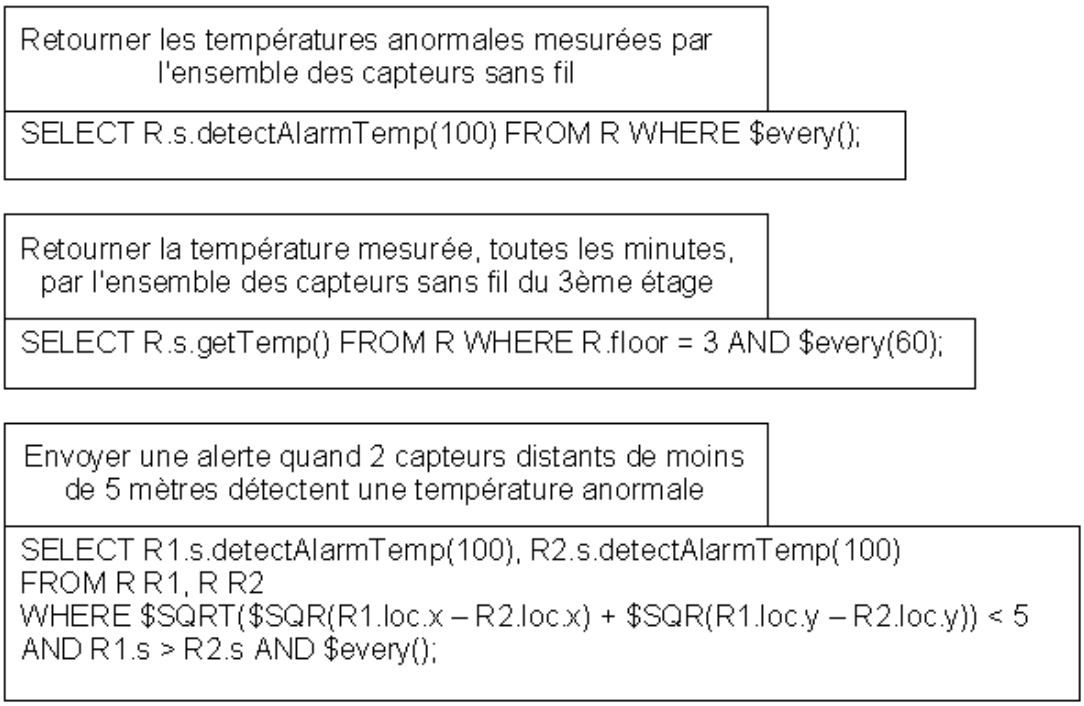

Figure 3.30 – Exemples de requêtes sous le système Cougar [Bonnet 2001]

La notion de « relation virtuelle » (« virtual relation ») a donc été introduite pour pallier ces manques observés. Une relation virtuelle complète à l'aide d'autres données les informations issues des attributs et fonctions de base ADT. Par exemple, quand la fonction ADT « detectAlarmTemp(*threshold*) » renvoie une seule donnée à chaque fois, la relation virtuelle « VRdetectAlarmTemp() » indique en plus l'identifiant du capteur sans fil auteur du prélèvement et la date à laquelle celui-ci a eu lieu.

Contrairement à une relation classique, une relation virtuelle n'autorise ni les mises-à-jour ni les suppressions mais seulement les ajouts de nouveaux éléments. Pour ne pas renvoyer des données inconsistantes, la lecture d'un nouvel enregistrement d'une relation virtuelle est une opération bloquante. Un pointeur ou curseur est utilisé pour indiquer la fin de l'acquisition de nouveaux enregistrements au sein d'une relation virtuelle.

3.4.2 Les mécanismes d'interrogation classiques

Le système LiveFile a été conçu pour répondre aux contraintes des RCSF et des besoins généralement rencontrés au niveau des applications. Le premier objectif a été de réduire la taille des données à la fois pour le stockage et la transmission des données. Différentes solutions ont été proposées dans les parties précédentes. Le deuxième objectif est la récupération des données et leur transmission soit vers un autre capteur soit vers une station centrale de collecte. Cette fonctionnalité appartient plutôt aux applications d'acquisition de données stockant les différents relevés sous le format de type « RECORD ».

Le système LiveFile a donc été muni d'un moteur d'interrogation pour accéder aux données recherchées. Comme pour les systèmes présentés au paragraphe 3.4.1, la sémantique d'interrogation employée est inspirée de celle du modèle relationnel et du langage SQL associé. De nombreux concepts sont définis au sein du modèle relationnel :

- les relations et les attributs ;
- les clés primaires avec les dépendances fonctionnelles et les différentes formes normales associées ;
- les clés étrangères et les dépendances d'inclusion.

Les relations et les attributs ont déjà été mentionnés pour introduire les métadonnées contextuelles. La plupart des applications d'acquisition s'intéressent à une seule grandeur qui aboutit généralement à l'utilisation d'une unique table ou relation. Les données sont stockées sous forme soit brutes soit d'enregistrements avec utilisation ou non des métadonnées contextuelles. Cette relation, si elle est unique, a le même nom que l'identifiant du capteur qui l'héberge. La clé primaire la plus souvent utilisée est le numéro de l'acquisition ou de l'enregistrement. Celle-ci n'est pas obligatoire sachant que les interrogations ou requêtes portent généralement sur les caractéristiques des données collectées. L'utilisation d'une seule table réduit la possibilité d'employer les autres concepts énoncés. Cependant, une dépendance d'inclusion voire une contrainte de clé étrangère est utile pour établir une hiérarchie ou un ordre entre les données d'une même relation.

De la même manière, différents opérateurs relationnels ont été définis dans le modèle relationnel :

- la sélection ;
- la projection ;
- la jointure ;
- la division.

Ceux-ci sont complétés par des opérateurs ensemblistes :

- le produit cartésien ;
- l'union ;
- la différence ;
- l'intersection.

La traduction de ces opérateurs en langage SQL n'est pas toujours identique d'un SGBD à un autre. La « différence » est représentée, selon les cas, par la clause « MINUS » ou « EXCEPT ». Quant à l'opérateur « division », il n'a pas de traduction simple en langage SQL. Il est parfois obtenu à l'aide d'une formule composée des opérateurs « projection », « sélection » et « produit cartésien ». Considérons deux relations « r » avec deux attributs « a,b » et « s » avec un attribut « b ». La division de « r » par « s » suivant « b » correspond à

la recherche des valeurs de l'attribut « a » qui, au niveau de la relation « r », sont associées à toutes les valeurs « b » présentes dans la relation « s ».

L'emploi d'une seule table rend les opérateurs ensemblistes inutiles. L'intérêt des opérateurs « jointure » et de « division » est d'une part également limité dans un tel cas. D'autre part, les traitements associés à ces opérateurs sont complexes et requièrent énormément de ressources (voir Figure 3.31). Ainsi, seuls les opérateurs de « sélection » et de « projection » sont présents de manière standard dans le système LiveFile, les autres pouvant être rajoutées.

Projection(r)
Parcours de l'ensemble des tuples d'une relation r
    Affichage des valeurs d'attributs recherchés

Sélection(r)
Parcours de l'ensemble des tuples d'une relation r
    Test sur le (ou les) critère(s) de sélection
    Affichage des valeurs d'attributs recherchés

Jointure(r, s)
Parcours de l'ensemble des tuples d'une relation r
    Recherche de correspondances dans une autre relation s suivant le critère de jointure
    Affichage des valeurs des attributs des tuples qui sont en correspondance

Division(r, s)
Parcours de l'ensemble des tuples d'une relation s
    Liste des valeurs distinctes pour l'attribut du critère de division
Parcours de l'ensemble des tuples d'une relation r
    Comparaison avec la liste de valeurs distinctes de l'attribut
    Affichage des valeurs des attributs des tuples associés à toutes les valeurs de s

Figure 3.31 – Fonctionnement des opérateurs relationnels

Des fonctions mathématiques sont associées à ces 2 opérateurs :

- « count(A) » : pour compter les valeurs de l'attribut « A » considéré ;
- « sum(A) » : pour sommer les valeurs de l'attribut « A » considéré ;
- « avg(A) : pour effectuer la moyenne des valeurs de l'attribut « A » considéré ;
- « min(A) » et « max(A) » : pour donner respectivement le minimum et le maximum des valeurs de l'attribut « A » considéré.

La clause « ORDER BY » est implémentée par l'intermédiaire d'un algorithme de tri sous le système LiveFile. Le tri utilisé par défaut peut être remplacé suivant les contraintes et les caractéristiques de l'application supportée.

Au niveau des RCSF, la clause « GROUP BY » est intéressante. Elle peut être utilisée pour connaître la température moyenne observée dans chacune des zones depuis l'application. Soit la relation « r » avec comme attributs « temp, zone, numjour », cette interrogation est traduite en langage SQL sous la forme suivante :

```
SELECT moy(r.temp) FROM r GROUP BY r.zone ;
```

Cependant, celle-ci, ainsi que la clause « HAVING », n'ont pas encore été implémentées, de manière optimale, pour une utilisation dans le cadre des RCSF. On déconseillera donc son utilisation au niveau du système LiveFile.

Le langage SQL définit également des opérateurs pour insérer, supprimer et mettre à jour les données respectivement :

- « INSERT INTO …VALUES(…) ; » ;
- « DELETE FROM … WHERE … ; » ;
- « UPDATE … SET … WHERE … ; ».

Les deux premiers sont présents dans le système LiveFile. Sachant que, dans un RCSF, les capteurs sont les sources de données, la mise-à-jour d'une donnée n'est pas un fonctionnement logique. Plutôt que de modifier une donnée existante, il est recommandé de la supprimer et d'en insérer une nouvelle corrigée.

Les dernières clauses concernent la création (« CREATE TABLE »), la modification (« ALTER TABLE ») et la suppression (« DROP TABLE ») d'une relation. Quand le format de type « RECORD » est utilisé, le schéma de la table est intégré dans la structure de l'enregistrement où peuvent apparaître ou non les métadonnées contextuelles. L'hypothèse considérée est que la ou les relations d'un capteur sont créées avant le déploiement et ne sont plus modifiées.

Les systèmes TinyDB et Cougar intègrent dans leurs requêtes des commandes pour exécuter différentes actions au niveau de l'acquisition des données. Notre approche est différente et vise à séparer les traitements propres aux systèmes d'exploitation et ceux liés au gestionnaire de données. Plus précisément, les tâches de fonctionnement du système d'exploitation LIMOS ne peuvent pas être programmées à partir du système LiveFile. Considérons la question suivante : « Quelles sont les températures supérieures au seuil « s » qui vont être acquises à une fréquence « f », à partir de maintenant pendant une durée « d » ? ».

Sous le système TinyDB, cette question est transformée en la requête suivante : « SELECT temp FROM sensors WHERE temp > s SAMPLE PERIOD f FOR d ; ». Des appels à des primitives du système d'exploitation se cachent derrière cette requête pour effectuer des relevés périodiques. Dans notre cas, pour répondre à une telle question, le système d'exploitation LIMOS va être configuré avant le système LiveFile. Deux configurations sont possibles. La première est la création d'un événement avec deux processus, l'un pour l'acquisition et, l'autre pour l'accès et le test de la donnée avec utilisation du système LiveFile. La deuxième est la définition de deux événements, l'un pour l'acquisition, l'autre pour l'interrogation (voir Figure 3.32). Un troisième processus ou événement peut être ajouté pour l'étape de communication.

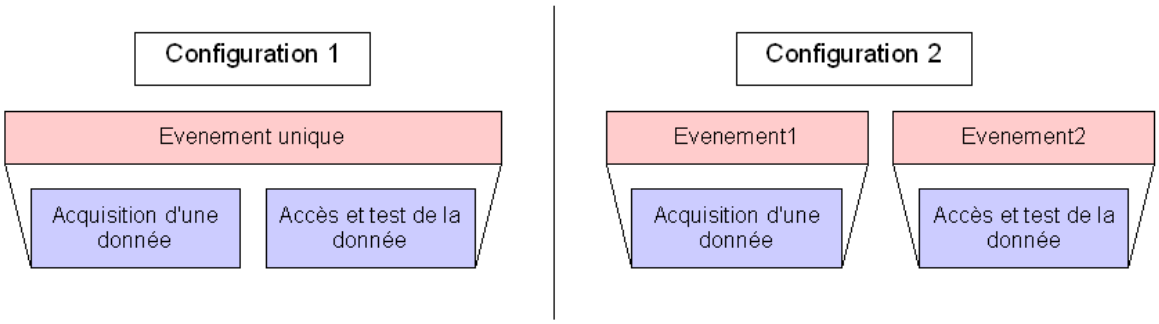

Figure 3.32 – Configuration système pour des interrogations temps réel

Pour diverses raisons et hypothèses, tous les opérateurs et clauses issus du langage SQL ne sont pas définis au sein du système LiveFile. Par conséquent, le système LiveFile est

considéré plus comme un microsystème de fichiers interrogatif dédié au RCSF plutôt qu'un SGBD.

3.4.3 L'accès rapide aux données

Le moteur d'interrogation du système LiveFile a été conçu pour répondre aux contraintes des RCSF et aux besoins généralement rencontrés au niveau des applications. Le premier objectif est de réduire la taille des données à la fois pour le stockage et pour la transmission des données. Différentes solutions ont été proposées dans les parties précédentes. Le second objectif est la récupération des données et leur transmission que ce soit vers un autre capteur ou vers une station centrale de collecte.

Pour accéder plus rapidement aux données recherchées, la principale technique consiste à indexer les données. Les RCSF étant une thématique de recherche assez récente, les travaux sur l'indexation des données adaptée sont peu nombreux même si 3 solutions d'indexations sont proposées dans [Noël 2006]. Celles-ci sont centrées sur la gestion des données spatio-temporelles et s'appuient sur des techniques à base d'arbres.

Notre approche est différente et dérive des métadonnées contextuelles présentées dans le paragraphe 3.3.2. Le remplacement des données par des métadonnées contextuelles associées n'est pas toujours possible. Certaines applications visent à obtenir la valeur la plus précise de la grandeur observée et n'entrent donc pas dans le cadre d'utilisation des métadonnées contextuelles. Cependant, il n'est pas nécessaire d'utiliser les métadonnées sur tous les attributs de la relation. Considérons une application de surveillance d'incendie qui se base sur des relevés de températures, la localisation des acquisitions est l'élément le plus important. Il est donc possible de renvoyer un intervalle plutôt qu'une température précise.

La solution proposée s'appuie sur les propriétés ou les caractéristiques des données collectées. Une ou plusieurs propriétés sont définies par l'utilisateur. A chaque acquisition, les données brutes vont être étudiées vis-à-vis de ces propriétés et le résultat sera stocké dans une structure de type tableau de bits (voir Figure 3.33). Celui-ci répertorie l'identifiant de la page stockant la donnée et indique pour chaque propriété si les données acquises la respectent (« 1 ») ou non (« 0 »).

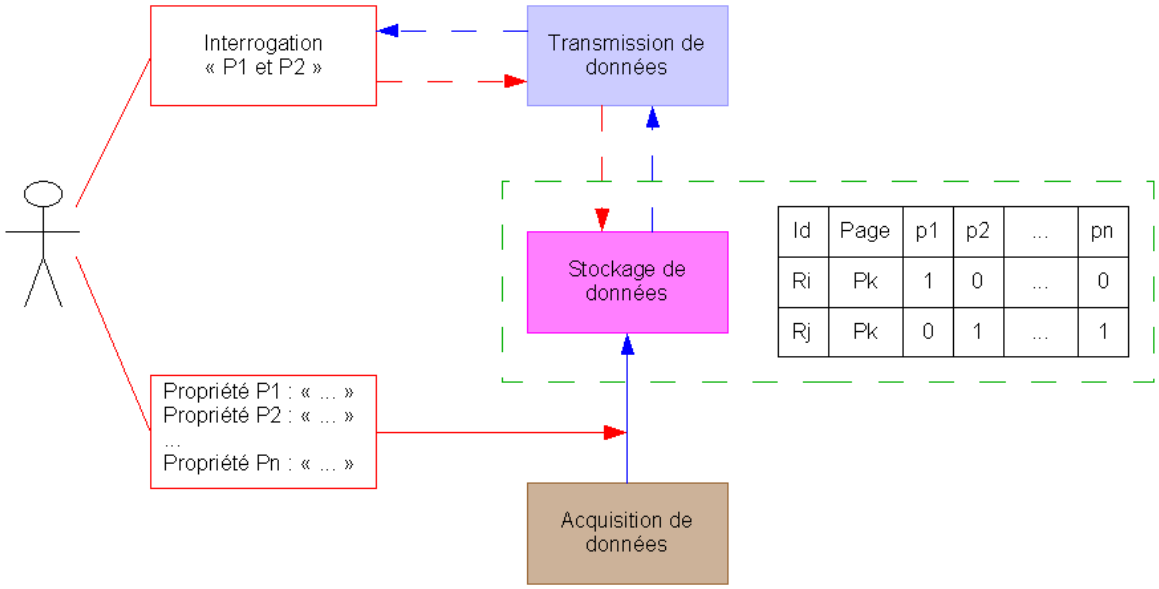

Figure 3.33 – Gestion des propriétés des grandeurs collectées

Naturellement seules les données respectant une ou plusieurs propriétés seront présentes dans le tableau d'indexation. La définition des propriétés est primordiale. Chaque propriété doit d'une part avoir une réalité par rapport à l'application. D'autre part, suivant les cas, deux ou plusieurs propriétés pourront être regroupées. Et à l'inverse, une propriété pourra être découpée en plusieurs. Le nombre de propriétés présentes est important. Plus celui-ci est grand, plus l'indexation apporte d'informations sur les données et permet une recherche plus rapide. Cependant, la taille du tableau d'indexation a tendance à croître avec l'augmentation du nombre de propriétés, les chances pour une donnée d'y figurer étant plus importante. Par exemple, pour une application d'acquisition de température, les propriétés suivantes peuvent être énoncées :

- « P1 : temp > S » ;
- « P2 : zone = A ou C ».

Les opérateurs « et », « ou » et « ! » ont été définis pour pouvoir combiner les propriétés. La formule « P1 et P2 » renvoie les enregistrements dont la température est supérieure à S et obtenus dans la zone A ou C. En terme d'application, son intérêt est de surveiller plus étroitement la zone A et C où les températures ont tendance à dépasser un seuil limite. Dans cet exemple, la propriété P2 aurait pu être découpée en « P2' : zone = A » et « P2'' : zone = C ». Mais, dans ce cas, la taille du tableau d'indexation augmente d'un bit multiplié par le nombre de données répertoriées. Ce tableau subit potentiellement des modifications à chaque acquisition de données. Par conséquent, il est stocké en mémoire RAM et sauvegardé suivant une fréquence donnée dans une page de type « CHECKPOINT ».

Une perspective envisagée à cette approche est la conception d'un système intelligent capable de générer de nouvelles propriétés par lui-même en se basant sur les interrogations successives de l'utilisateur final et des données brutes collectées.

Enfin, dans quelques applications, l'utilisateur final récupère des indications sur l'objet observé à une fréquence donnée. Et, de manière aléatoire, il va faire appel à une même requête pour avoir des précisions. Connaissant cette requête avant le déploiement, le système peut être programmé pour établir le résultat de celle-ci au fur et à mesure de nouvelles acquisitions et être prêt au moment de l'interrogation.

3.5 Les autres fonctionnalités du système LiveFile

En l'associant avec d'autres composants logiciels, le système LiveFile offre d'autres fonctionnalités. La première est la sauvegarde de données sur plusieurs capteurs. Celle-ci a été introduite dans le paragraphe 3.2.2.3 portant sur l'utilisation de la primitive In() pour accéder aux données. La seconde concerne de potentielles applications au niveau de la gestion de la communication au sein du RCSF. Cette partie présente donc ces différentes fonctionnalités en indiquant quel type de composants logiciels sont nécessaires pour permettre leurs utilisations.

3.5.1 La sauvegarde de données d'un capteur

Le stockage des données dans la mémoire Flash évite de perdre l'ensemble des données collectées après une panne du système. Les données ainsi sauvegardées peuvent être restaurées après récupération du capteur et résolution de la panne. Les deux grandes causes de pannes rencontrées dans un capteur sans fil sont soit un blocage au niveau du système d'exploitation soit un problème d'alimentation en énergie. Des mécanismes de type « chien de garde » limitent l'impact du premier type de panne. Au niveau du second type de panne, de nombreux travaux sont menés pour d'une part augmenter le rendement des batteries utilisées et d'autre part diminuer l'énergie consommée par le capteur. A ce titre, le développement des technologies de communication IEEE 802.15.4 ZigBee ont réduit, de manière significative, l'énergie consommée par rapport au Wi-Fi IEEE 802.11b.

Même si les données ne sont pas perdues, elles restent inaccessibles tant que le capteur en panne n'a pas été récupéré. Or, dans certains cas, les capteurs sont utilisés car l'accessibilité au lieu de déploiement est compliquée voire impossible autrement. Par conséquent, le système LiveFile a été muni d'un mécanisme pour permettre la sauvegarde des données d'un capteur sur plusieurs autres. Les informations de type « CHECKPOINT » étant transmises, le procédé employé se situe entre la duplication et la réplication des données. Supposons que le capteur tombe en panne, les pages qui auraient été libérées après le dernier « CHECKPOINT » ne seraient d'une part pas prises en compte. D'autre part, la sauvegarde de tout type de données sur un autre capteur se produit si elle est clairement explicitée dans l'argument « key » de la primitive Out(). Seules les données les plus importantes seront a priori sauvegardées plusieurs fois. Par conséquent, il est difficile de parler de réplication sachant que la copie n'est pas totalement fidèle à l'original.

Une sauvegarde par bloc de pages est également envisageable et serait peu coûteuse en ressources si la liste des pages à transmettre est obtenue sans nécessiter de lourds traitements. La sauvegarde de données sur un autre capteur serait utile pour disposer d'une copie des données stockées dans un buffer de type SRAM et en attente d'une programmation en mémoire Flash. Ce mode de fonctionnement revient à privilégier la mémoire Flash au détriment de l'énergie consommée pour la transmission des données.

Le système LiveFile ne possède pas de mécanisme évolué pour partager les données d'un capteur entre plusieurs autres. L'étape de duplication est réalisée par l'utilisation de la primitive Out(). En revanche, la récupération de données est plus complexe et pose différentes problématiques qui n'ont pas encore été abordées. Les principales sont le choix de la structure de données utilisée pour savoir sur quel capteur sont présentes les différentes informations disséminées. L'emplacement de stockage de cette structure est également important. La première solution est de transmettre celle-ci à la station centrale de collecte de données. Actuellement, si un capteur en choisit plusieurs autres pour sauvegarder ses données, la méthode définie dans le système LiveFile impose de stocker l'intégralité des données sur chacun d'entre eux pour en faciliter la récupération par la suite.

La zone où les données d'un autre capteur sont stockées doit être clairement délimitée. Deux solutions sont possibles pour réaliser cette opération. La première est de réserver au niveau de chaque capteur, avant leur déploiement, une zone de mémoire Flash dédiée à cette fonction. La problématique à considérer dans ce cas est le correct dimensionnement de cette zone. La deuxième solution est de réserver une ou plusieurs pages de mémoire Flash où sont indiqués les emplacements des données issues d'un autre capteur. Elle ne pénalise donc pas le capteur qui héberge les données d'un autre mais complexifie grandement le processus de sauvegarde. Dans la version actuelle du système LiveFile, ces problématiques ont été contournées en commençant le stockage des données d'un autre capteur à partir de la fin de la mémoire Flash et en conservant le numéro de la page programmée en dernier (voir Figure 3.34).

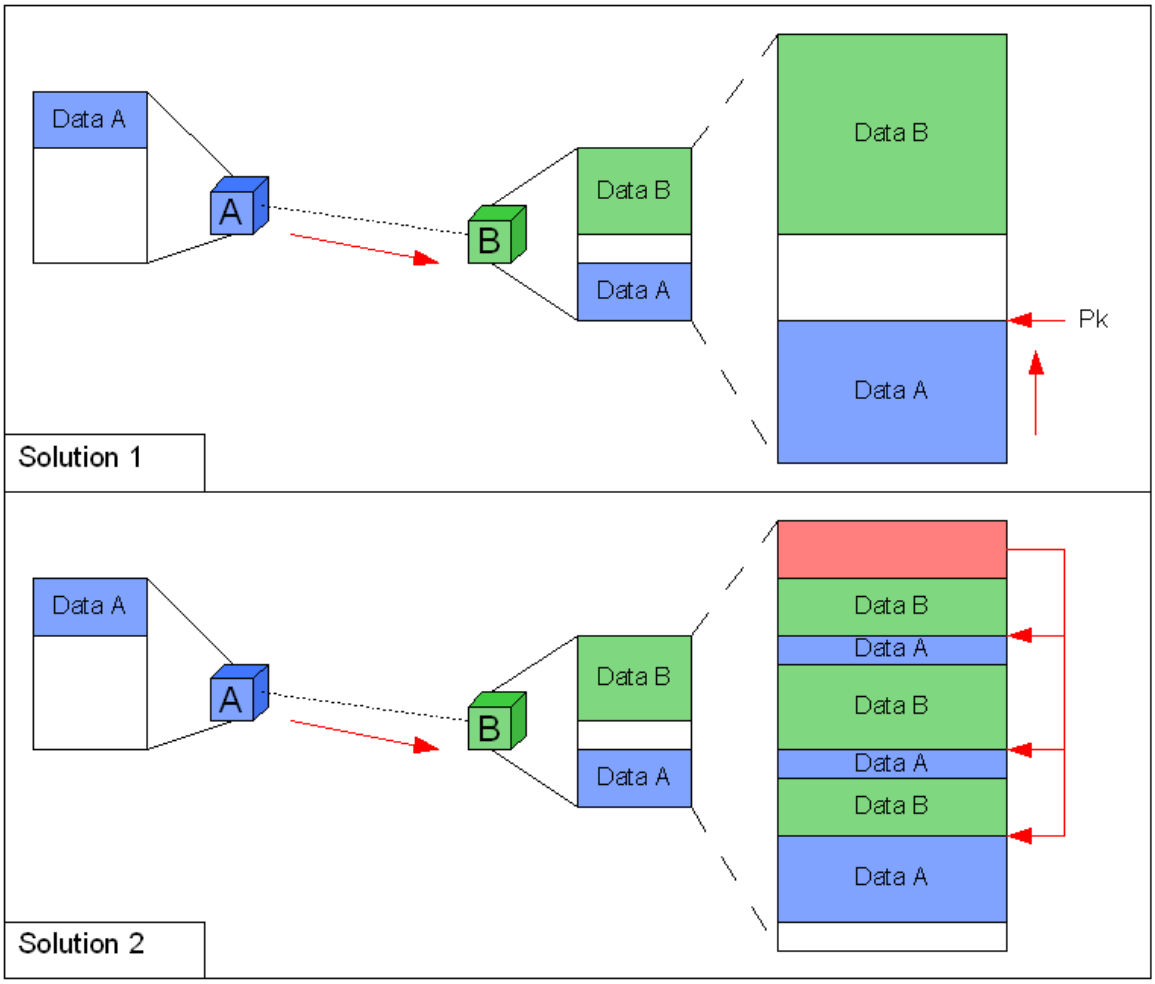

Figure 3.34 – Sauvegarde des données sur un autre capteur

Ce fonctionnement a été validé entre deux capteurs mais pas au niveau d'un RCSF. Pour effectuer une évaluation complète, le système LiveFile doit être associé à un protocole de routage. Au niveau des réseaux sans fil Ad Hoc, différents protocoles ont été élaborés. Ils sont répartis principalement dans les 3 grandes familles suivantes :

- les protocoles proactifs ;
- les protocoles réactifs ;
- les protocoles hybrides.

Le principe de base des protocoles proactifs est que chaque nœud du réseau a, dans sa table de routage, un chemin vers l'ensemble des nœuds du réseau. Ce mode de fonctionnement implique des échanges constants entre les différents nœuds pour prendre en compte les potentiels changements de topologie du réseau. Le principal avantage de ce type de protocole est que chaque nœud dispose immédiatement d'un chemin vers n'importe quel autre nœud du réseau. Les protocoles DSDV [Perkins 1994] et OLSR [Clausen 2003] appartiennent à cette famille.

Dans un protocole réactif, les chemins sont construits au fur et à mesure des communications entre les nœuds. Au départ, la table de routage d'un nœud est vide. Quand un nœud va vouloir communiquer avec un autre, la première étape est la recherche d'un chemin valide par l'envoi de messages à travers le réseau. Si un tel chemin est découvert, la communication pourra débuter. Par rapport à un protocole proactif, le nombre de messages pour établir les routes est moindre. Des exemples de protocoles réactifs sont AODV [Perkins 2003] et DSR [Johnson 2007].

Les protocoles de routage hybrides combinent les aspects proactif et réactif de différentes manières [Abolhasan 2004]. Par exemple, dans [Ramasubramanian 2003] et [Haas 1997], une zone au fonctionnement proactif est définie autour de chaque nœud. Au-delà de cette zone, un protocole réactif est utilisé. D'autres protocoles comme FZRP [Yang 2007] utilisent des méthodes différentes à base de constructions d'arbres de routage.

Par rapport à leurs caractéristiques, les protocoles de routage proactifs voire hybrides semblent être les mieux adaptés pour réaliser la sauvegarde de données sur plusieurs nœuds ou capteurs. La table de routage pourra être ainsi utilisée pour trouver un capteur dans lequel les données pourront être stockées.

Pour certaines applications, un intérêt existe pour qu'un capteur n'effectue pas une copie de ses données dans un nœud voisin. Si l'on reprend l'exemple d'une application de surveillance d'incendie de forêt, les capteurs doivent être très proches les uns des autres pour obtenir un quadrillage assez fin de la zone observée et permettre une localisation la plus précise possible. Lorsqu'un incendie se déclare, suivant la vitesse d'intervention des pompiers, un capteur et le voisin sur lequel il a choisi de dupliquer ses données pourraient être tous les deux endommagés et entraineraient la perte des données sur le départ du feu.

3.5.2 La gestion de la communication

Le système LiveFile peut également intervenir au niveau de la communication. Dans la fonctionnalité précédente, les liens qui peuvent exister entre le système LiveFile et le protocole de routage utilisé ont été introduits. Dans cette section, les rôles que peut jouer le système LiveFile en ce qui concerne la gestion de la communication vont être présentés.

3.5.2.1 La gestion des données de routage

L'une des premières utilisations pressentie pour le système LiveFile était la gestion des données liées au routage. Au-delà du stockage non volatile de la table de routage pour cause de redémarrage du système, d'autres applications existent.

La première est née à partir du protocole de routage OLAR (Obstacle Location-Aided Routing) [De Sousa 2005]. Ce dernier a été élaboré à partir du protocole LAR version 2 [Ko 2000]. Pour être plus complet, les familles de protocoles suivantes peuvent être ajoutées à celle présentées dans la section précédente :

- les protocoles géographiques ;
- les protocoles multicast ;
- les protocoles geocast.

Le protocole OLAR fait partie des protocoles géographiques. Ces protocoles partent du principe que les capteurs sont munis de dispositifs de localisation. Avec la démocratisation des systèmes GPS, cette hypothèse est de plus en plus valide. A partir de ces informations sur la position des nœuds, différentes méthodes ont été élaborées pour réduire la quantité de messages transmis pour établir une communication [Stojmenovic 2002].

Le protocole OLAR intègre la prise en compte des obstacles à la communication pour améliorer la qualité des chemins créés. Pour envisager une implémentation en réel de ce protocole, différents verrous doivent être levés :

- la détection et l'évaluation des obstacles ;
- le stockage des informations sur les obstacles.

Le système LiveFile apporte en partie une solution au deuxième verrou. Il autorise le stockage des informations sur les obstacles avant même le déploiement à l'aide de la mémoire Flash. Cependant, ce verrou ne se résume pas uniquement à l'aspect gestion des données au niveau du capteur. Il implique également la définition du format de stockage des informations sur les obstacles.

Le protocole CIVIC (Communication Inter Véhicules Intelligente et Coopérative) a été, au départ, élaboré pour répondre aux besoins de la communication entre véhicules civiles ou agricoles [Chanet 2007]. Il subit actuellement des modifications pour pouvoir être utilisé dans d'autres types d'applications [Hou 2007a]. L'intégration de la nouvelle version de CIVIC avec le système LiveFile au sein du système d'exploitation LIMOS aboutira en un système complet adaptatif dédié au RCSF.

3.5.2.2 La Qualité de Service

Dans la communication sans fil, la Qualité de Service (ou QdS) correspond à un niveau d'exigence lié aux besoins de l'application. Ces niveaux sont établis à partir de différents critères ou grandeurs propres aux réseaux tels que la bande passante disponible, le délai de communication, le nombre de messages perdus. Le contexte des RCSF constitués de nœuds mobiles disposant de ressources limitées complexifie la définition et le respect de la QdS.

Dans le cadre de RCSF, de nouvelles techniques d'estimation de la bande passante disponible sont à l'étude. L'objectif visé est d'obtenir une méthode non invasive avec une estimation la meilleure possible. L'estimateur NIMBE en est un exemple [Chanet 2006] [Chanet 2007]. Son principe de fonctionnement s'appuie sur le relevé et l'interprétation des temps d'aller-retour ou RTT (Round-Trip Time). L'estimateur NIMBE est associé à un automate de décision qui régule le trafic suivant la valeur de bande passante disponible relevée. En supposant la topologie du réseau étudié fixe, les valeurs des RTT de l'instant courant pourraient être comparées à d'autres très proches présentes dans un historique. Le comportement du réseau pourrait être ainsi mieux évalué par rapport à ses réactions passées. L'historique des RTT avec leur évolution à partir d'une situation donnée pourrait être stocké dans le système LiveFile. Des intervalles de RTT plus ou moins fins pourraient être définis à l'aide des métadonnées contextuelles :

- « RTTint1 » : 0ms < RTT < 1ms ;
- « RTTint2 » : 1ms < RTT < 2ms ;
- « RTTint3 » : 2ms < RTT < 3ms, etc.

La recherche d'une situation rencontrée précédemment serait ainsi accélérée. Chaque intervalle « RTTint » devra comporter un nombre de valeurs en rapport avec la puissance de

calcul du capteur, le processus de décision devant opérer rapidement. L'avantage de cette méthode est d'améliorer l'estimation sans générer plus de trafic.

Le multi-support consiste à utiliser différentes technologies sans fil complémentaires pour répondre à la QoS requise. Les protocoles de communication Wi-Fi et ZigBee peuvent être associés pour bénéficier respectivement du débit de l'un et de la portée de l'autre. Le choix du protocole utilisé s'établit à partir des relevés provenant d'un estimateur de bande passante. A l'aide du système LiveFile, la décision de conserver ou de changer le protocole actuel peut se baser sur les valeurs acquises sur plusieurs cycles de mesure plutôt qu'un seul voire deux.

3.6 Synthèse sur le système de fichiers LiveFile

Même si le système LiveFile peut être utilisé avec un autre système d'exploitation, son élaboration s'appuie sur 2 objectifs :

- l'intégration dans le système d'exploitation LIMOS ;
- le développement d'un système reconfigurable et adaptable à différents types d'applications.

Pour atteindre le premier objectif, le concept LINDA présent dans le système LIMOS a été étendu. Les primitives Out() et In() rendent transparente l'utilisation du système LiveFile vis-à-vis du noyau LIMOS.

En ce qui concerne le second objectif, de nouvelles fonctionnalités sont venues et viennent enrichir le système LiveFile pour répondre aux besoins mis en évidence au fur et à mesure des applications supportées. Deux formats différents sont définis pour stocker les données collectées :

- les enregistrements ;
- les fichiers.

Les enregistrements sont des structures de données adaptées aux données collectées. Chaque enregistrement contient un ensemble de renseignements correspondant aux besoins de l'utilisateur. Les fichiers sont plus proches de flux de données sans organisation particulière regroupés au sein d'une même entité.

Le système LiveFile tend ainsi à devenir un intergiciel pour la gestion des ressources mémoire. De plus, le module de gestion de données doit être complété pour en faire un SGBD embarqué. Au-delà des opérateurs généralement présents dans un SGBD, le système LiveFile pourra être muni de commandes pour activer des processus ou des événements. A la différence du système TinyDB, l'ordonnancement et la réalisation des tâches générées seront de la responsabilité du noyau LIMOS.

Le chapitre suivant présente différentes applications et quelques cas d'utilisation du système complet comprenant le capteur sans fil LiveNode, le noyau temps réel LIMOS et le microsystème de fichiers interrogatif LiveFile.

Chapitre 4

La plateforme LiveNode

Le noyau LIMOS et le système de fichiers LiveFile ont été élaborés pour pouvoir répondre aux besoins d'un large panel d'applications de RCSF. Le noyau LIMOS est d'une part un système d'exploitation offrant trois configurations différentes : multitâches, basé sur les événements et hybride, ce dernier combinant les deux premiers. D'autre part, le format des données gérées par le système LiveFile est adapté à l'application. Les fichiers et les enregistrements sont les deux structures de données disponibles. Les fichiers sont plutôt réservés au stockage de données de manière temporaire suite à une indisponibilité momentanée de la communication. Les enregistrements servent de conteneurs pour le stockage et l'interrogation des données issues des capteurs de grandeur physique. L'implémentation du système LiveFile ainsi que les modifications effectuées sur le noyau LIMOS pour permettre son intégration sont décrites en détails dans ce chapitre.

Ces deux composants logiciels s'appuient sur un capteur sans fil nommé LiveNode (LIMOS Versatile Embedded Node). Ce capteur a été conçu à partir de travaux de recherche dans le domaine de la télémédecine et de la communication inter-véhicule (CIV). Trois différents types de capteurs dont le capteur LiveNode sont présentés au début de ce chapitre. Ils représentent différentes générations de capteurs sans fil. Le capteur LiveNode est un capteur « évolué » dont la principale particularité est son architecture matérielle multi-composant.

Le capteur LiveNode associé aux systèmes LIMOS et LiveFile constituent un dispositif complet dédié aux applications de RCSF. Le fonctionnement de ce dispositif est illustré dans le cadre de différents projets réalisés et dans un cas d'utilisation ayant pour objet l'acquisition et la gestion de données environnementales.

4.1 Les différentes versions de capteurs sans fil réalisées

Au sein de l'équipe SMIR (Systèmes Multi-sensoriels Intégrés Intelligents Répartis) du laboratoire LIMOS (Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes) UMR CNRS 6158, différents projets de recherche dans des domaines d'applications divers et variés ont permis le développement de dispositifs électroniques proches des capteurs sans fil. Le projet STAR (Système Télé-Assistant Réparti) a, par exemple, donné lieu à la création d'un capteur biomédical dédié à la détection et à l'analyse des signaux électrocardiogrammes [de Vaulx 2003] [Zhou 2004b]. D'autres développements répondant à la problématique de la communication inter-véhicule ont donné naissance au capteur E-CIVIC [Chanet 2007]. La réalisation de ces deux capteurs a permis à l'équipe d'acquérir un niveau d'expertise permettant l'élaboration d'un capteur sans fil multi-composant nommé LiveNode. Les principales caractéristiques et spécificités de ces trois types de capteurs sont présentées dans ce qui suit.

4.1.1 Les capteurs biomédicaux

Différents capteurs biomédicaux ont été développés dans le cadre du projet de télémédecine STAR (Système Télé-Assistant Réparti) introduit dans le chapitre 1 [de Vaulx 2003] (voir Figure 4.1).

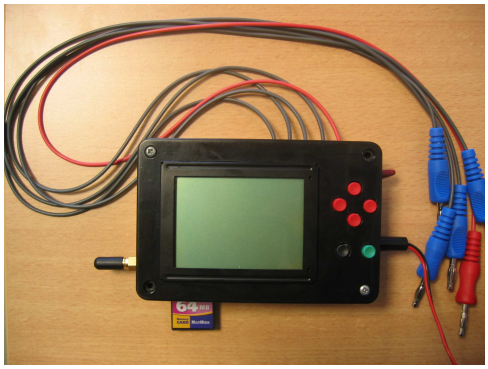

Figure 4.1 – Capteur biomédical du projet STAR

Chacun l'ont été en respectant les recommandations publiées par l'AHA (American Heart Association). Chaque capteur permettait de récupérer les signaux électrocardiogrammes sur quatre dérivations différentes. La dernière version s'appuie sur une architecture matérielle basée sur un microcontrôleur MSP430 et équipée d'un module de communication IEEE 802.15 Bluetooth. Les traitements réalisés par ces capteurs biomédicaux ont également évolué. Les premiers étaient uniquement en charge de la récupération et du conditionnement des signaux électrocardiogrammes auprès de la personne sous surveillance. L'interprétation des anomalies cardiaques détectées est une fonctionnalité qui a été ajoutée par la suite. Cette information est fournie à titre indicatif et ne remplace aucunement le diagnostic d'un cardiologue.

Ce projet a également donné lieu au développement de deux systèmes d'exploitation embarqués : DREAM [de Vaulx 2003] et SDREAM [Zhou 2004b].

Sur ce capteur STAR, devait s'appuyer une plateforme dédiée à la détection de problèmes d'arythmies cardiaques à distance et en temps réel [Zhou 2004a] [Zhou 2004b] (voir Figure 4.2). Ce dispositif est divisé en deux grandes parties :

- Les équipements fournis à la personne sous surveillance ;
- Un centre hospitalier de traitement et d'intervention.

Un ordinateur personnel est installé au domicile de la personne sous surveillance. Cet ordinateur est équipé d'un module de communication Bluetooth pour pouvoir recevoir les informations collectées sur la personne par le capteur biomédical. Ces informations sont ensuite transmises de l'ordinateur vers un centre hospitalier, par l'intermédiaire d'une ligne téléphonique classique, pour être interprété par un cardiologue. Ainsi, suivant le problème détecté, ce dernier peut ou non faire intervenir rapidement des secours au domicile de la personne sous surveillance. Pour limiter au maximum les diagnostics erronés suite à une mauvaise manipulation du capteur biomédical de la part de la personne sous surveillance, l'ordinateur est muni d'une caméra utilisée, si nécessaire, par le centre hospitalier pour lever tout doute. La principale source d'erreurs ou de relevés anormaux est le décollement d'une électrode.

La possibilité d'équiper le capteur biomédical STAR avec un module de communication GSM (Global System for Mobile communications) / GPRS (General Packet

Radio Service) et d'un système de localisation GPS (Global Positioning System) a été envisagée pour donner encore plus de liberté à la personne sous surveillance.

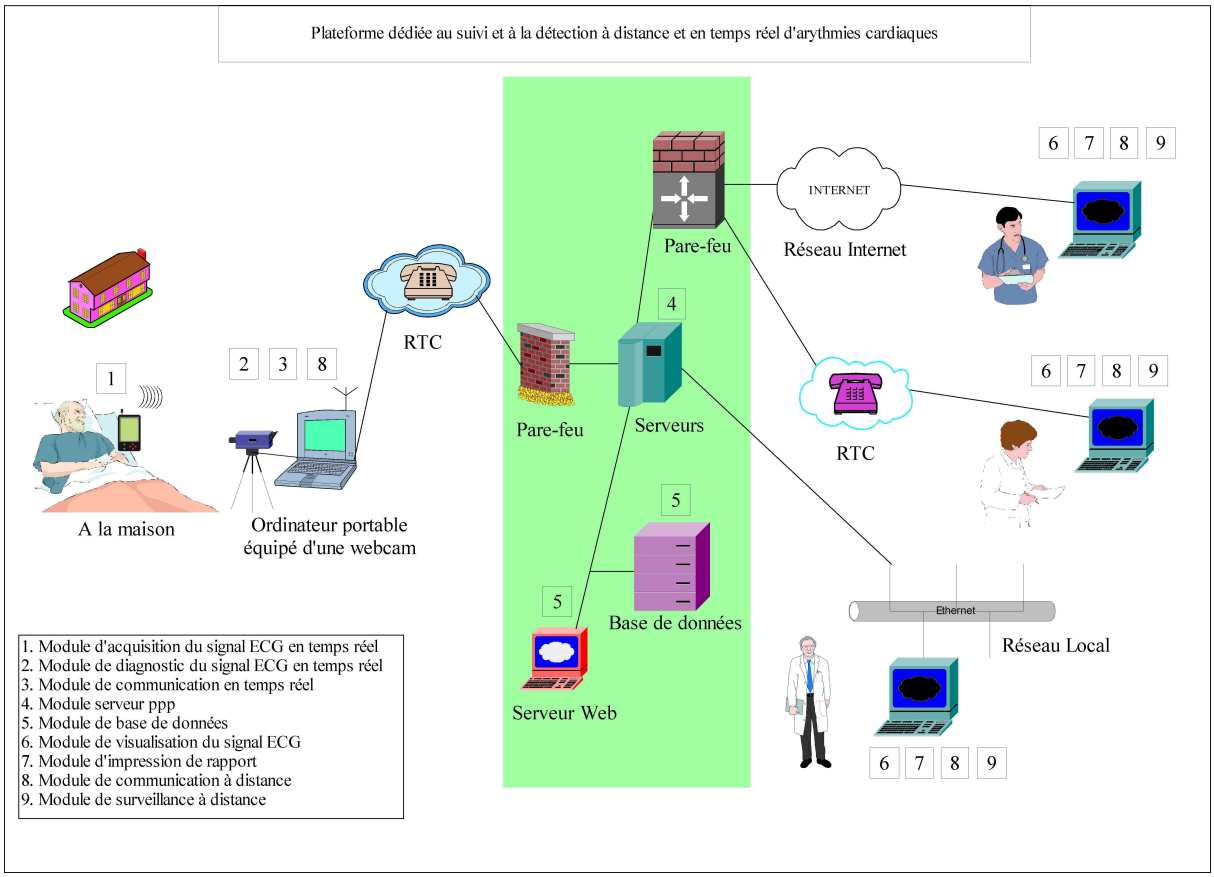

Figure 4.2 – Plateforme STAR [de Vaulx 2003]

4.1.2 Le capteur E-CIVIC

Différents travaux ont été menés, au sein de la fédération de recherche TIMS (Technologies de l'Information, de la Mobilité et de la Sûreté) du pôle clermontois, sur la thématique des véhicules intelligents et la communication inter-véhicule. A ce titre, une première version du capteur sans fil E-CIVIC (Embedded-Communication Inter Vehicle Intelligent and Cooperative), possédant l'architecture matérielle décrite ci-dessus, a été développé :

- 3 unités de calcul : 2 microprocesseurs MSP et 1 microprocesseur CP3000 ;
- 3 modules de communication : 1 module IEEE 802.11b Wi-Fi, 1 module IEEE 802.15 Bluetooth et 1 module GSM/GPRS ;
- 1 système de localisation par GPS ;
- Différentes interfaces séries tel que des ports RS232 et USB ;
- 1 interface bus CAN (Controller Area Network).

Le microcontrôleur CP3000 est l'unité de calcul principal. Elle fait appel aux unités secondaires que sont les microprocesseurs MSP pour des traitements spécifiques. La présence de modules de communication Wi-Fi et Bluetooth permet l'utilisation de la technique du multi-support.

Le capteur E-CIVIC est également muni d'applications logicielles adaptées aux contraintes de la communication inter véhicule comme, par exemple, le protocole de routage CIVIC (Communication inter Vehicle Intelligent and Cooperative), introduit dans le paragraphe 3.5.2.1 du chapitre 3 [Zhou 2006d]. Ce protocole intègre différentes fonctionnalités comme des techniques d'estimation de la bande passante disponible et des méthodes pour prendre en compte les spécificités du domaine d'application comme, par exemple, la vitesse des véhicules. Enfin, le fonctionnement général du capteur E-CIVIC est à la charge du micronoyau temps-réel tolérant aux pannes DREAM (voir Figure 4.3).

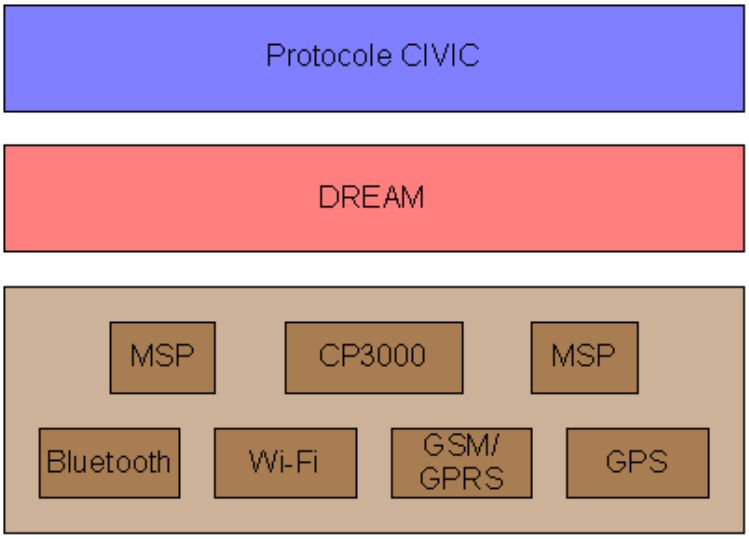

Figure 4.3 – Architecture du capteur sans fil E-CIVIC

4.1.3 Le capteur LiveNode

Bien qu'adaptée aux contraintes rencontrées, l'architecture matérielle du capteur sans fil E-CIVIC comporte un certain nombre d'inconvénients. Le principal est la complexité d'utilisation de cette architecture que ce soit pour ajouter de nouveaux composants matériels ou pour implémenter de nouvelles applications logicielles. Ce capteur sans fil a donc laissé place à un capteur E-CIVIC de deuxième génération [Chanet 2007] rebaptisé LiveNode (LIMOS Versatile Embedded Node) [Hou 2007b].

Le capteur sans fil LiveNode intègre d'importantes évolutions tant sur le plan matériel que logiciel. Au niveau matériel, tout d'abord, l'approche multi-composant a été introduite et privilégiée. Ainsi, une unité matérielle de base a été définie et conçue autour du microcontrôleur AT91SAM7S256 de la famille des ARM7TDMI. Une des particularités de ce microcontrôleur est d'avoir une fréquence de fonctionnement que l'on peut faire varier de 500Hz à 55MHz. Les autres caractéristiques principales de ce microcontrôleur sont :

- 64Ko de mémoire SRAM et 256Ko de mémoire Flash ;
- 1 convertisseur Analogique/Numérique 8 voies 10 bits ;
- 2 ports série RS232 ;
- 1 port USB ;
- 1 interface SPI ;
- 1 bus I²C.

Les différentes interfaces dont dispose ce microcontrôleur permettent de connecter un large panel de dispositifs. Dans l'unité matérielle de base, une des interfaces série est réservée

à un module de communication de type soit IEEE 802.11b Wi-Fi, soit IEEE 802.15.4 ZigBee. Un système de localisation par GPS ainsi qu'un module GSM peuvent également être ajoutés grâce aux ports série restants.

Cette unité logicielle de base représente la configuration la plus simple du capteur LiveNode (voir Figure 4.4). Plusieurs unités de ce type peuvent être combinées pour répondre aux contraintes d'un large panel d'applications. Par exemple, deux unités disposant d'un module de communication différent peuvent être associées pour permettre l'utilisation de la technique du multi-support.

Figure 4.4 – Capteur sans fil LiveNode

La gestion du capteur sans fil LiveFile a été confiée à un nouveau système d'exploitation : le noyau hybride LIMOS. Ces deux éléments ont été développés, en parallèle, quasiment en même temps. Le protocole CIVIC a été porté sur ce nouveau noyau tout en ayant subi des améliorations comme, par exemple, la présence de techniques d'estimation de la bande passante disponible plus performantes.

4.2 Implémentation de la gestion de données

Le système LiveFile a été développé à partir de besoins observés au niveau de la gestion et de l'interrogation des données collectées ou gérées dans une application de RCSF. Les différents concepts et mécanismes introduits par le système LiveFile ont été présentés dans le chapitre 3. L'implémentation de ces différents éléments à travers des fonctions et leur intégration dans le noyau LIMOS est l'objet de la présente partie. La plateforme LiveNode peut être présentée suivant différents angles d'étude. Celui de la gestion de données au sein du capteur LiveNode a été rétenu.

La première section s'attache à la description des caractéristiques de la mémoire Flash interne du microcontrôleur RISC 32 bits AT91SAM7S256 (ARM7TDMI) d'Atmel Corporation [ARM7 2007] utilisé dans le capteur LiveNode. Elle fournit le contexte dans lequel les différents éléments de programmation qui seront présentés ont été développés. Puis, les primitives associées respectivement à la gestion de la mémoire et à celle des données sont présentées. Au fur et à mesure des développements, la nécessité d'une interface permettant l'interrogation par un utilisateur des données stockées dans un capteur a été mise en évidence. La dernière section donne les premières indications sur le fonctionnement d'une interface dédiée à la gestion de données au sein de la plateforme LiveNode.

4.2.1. Mémoire physique et son adressage

Le système LiveFile a été implémenté dans le microcontrôleur AT91SAM7S256 équipant le capteur sans fil LiveNode [ARM7 2007]. Ce microcontrôleur est développé par la société Atmel et fait partie de la famille des microcontrôleurs RISC ARM7TDMI à laquelle appartient également le microcontrôleur LPC2106 de la société Philips. Les caractéristiques et le fonctionnement de la mémoire disponible au niveau du microcontrôleur AT91SAM7S256 ont d'abord été étudiés.

La mémoire Flash de ce microcontrôleur est découpée en 1024 pages de 256 octets pour un total de 256Ko. La taille de la mémoire SRAM n'est, quant à elle, que de 64Ko. L'adressage de la mémoire Flash se fait par blocs de 32 bits ou 4 octets (voir Figure 4.5).

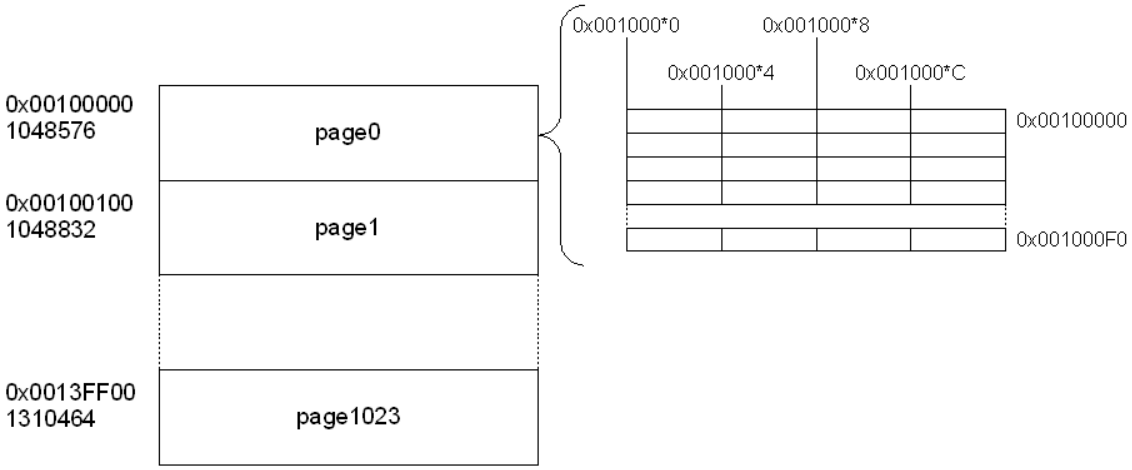

Figure 4.5 – Adressage de la mémoire Flash dans le microcontrôleur AT91SAM7S256

Dans le microcontrôleur AT91SAM7S256, la première page « p_0 » a pour adresse, en hexadécimal, 0x00100000 et la dernière 0x0013FFF0. La formule suivante permet de passer d'un numéro de page « p_i » à l'adresse de mémoire Flash associée :

$$adr(p_i)_{hexa} = adr\ (p_0)_{hexa} + (taille\_page * i)_{hexa}$$

Ainsi, différents formats de données peuvent être utilisés pour stocker l'identifiant d'une page :

- l'adresse physique de la page ;
- le numéro de la page par rapport à l'adresse de début de la mémoire Flash.

Chacune de ces solutions a des avantages et des inconvénients. La première solution permet de s'affranchir de l'organisation de la mémoire utilisée. La seconde utilise moins de mémoire pour le stockage de l'identifiant. Au niveau programmation, l'utilisation de caractères (« char ») à la place d'entiers (« int ») permet de réduire le nombre d'octets nécessaire au stockage d'une adresse. Suivant le type de données, une adresse peut être codée sur 2, 3 ou 4 octets. Cependant, l'utilisation d'adresses mémoire stockées sous forme de caractères requiert des conversions avant toutes opérations d'écriture ou de lecture.

Comme pour l'adressage, toute programmation ou écriture doit s'effectuer avec au minimum 32 bits. Les écritures par blocs de 8 ou 16 bits provoquent des erreurs. En revanche, la lecture par blocs de 8 bits est autorisée.

De part ses caractéristiques de langage haut niveau muni de primitives de programmation bas niveau, le langage C est utilisé dans l'implémentation de nombreux noyaux temps réel embarqués. Il en est ainsi pour le noyau LIMOS. Le système LiveFile a donc été implémenté essentiellement dans ce langage de programmation pour d'une part faciliter le portage sur différents noyaux et microcontrôleurs et d'autre part obtenir une empreinte mémoire réduite.

En cas de changement d'architecture matérielle, les principales modifications à effectuer concernent la spécification des primitives d'accès à la mémoire Flash avec l'indication de la taille des pages à gérer. Un autre élément à prendre en compte est la localisation des différents composants logiciels présents au sein de la mémoire Flash, le principal étant le système d'exploitation. Certains programmes sont parfois répartis à différents emplacements de la mémoire Flash. L'« ARM Boot Loader » en est un exemple et est présent à l'adresse 0x0013E700 quand il est installé. L'utilisation du système LiveFile requiert donc une étude au préalable de l'espace mémoire réellement disponible. Des zones de « sécurité » sont à définir pour éviter les risques d'écriture dans des emplacements réservés et non protégés.

4.2.2 Fonctions pour la gestion de la mémoire Flash

La première fonctionnalité attribuée au système LiveFile est la gestion de l'écriture et de la lecture de la mémoire Flash [De Sousa 2007a] [De Sousa 2007b]. Ces deux opérations font appel à des primitives bas niveau propres à la mémoire Flash utilisée qui, dans notre cas, est celle embarquée dans le microcontrôleur AT91SAM7S256.

L'accès à la mémoire Flash en lecture est obtenu en se positionnant à l'adresse mémoire où sont stockées les données recherchées (voir Figure 4.6). Les données peuvent être lues directement ou copiées dans un buffer de lecture. L'emploi d'un tel buffer permet la modification des données lues et, par la suite, leur recopie. Généralement, dans un souci de préservation de la mémoire Flash, les données provenant d'une page et qui auraient été modifiées seront recopiées dans une page différente. Dans l'exemple fourni, la constante « FLASH_PAGE_SIZE » indique la taille d'une page de mémoire Flash c'est-à-dire 256 octets pour le microcontrôleur AT91SAM7S256.

L'écriture ou programmation est une opération plus complexe qui passe par la manipulation d'un certain nombre de registres (Figure 4.7). Avant toute écriture, les différents

verrous de protection doivent être levés. La constante « FLASH_START_ADDRESS » est fixée à « 0x00100000 ».

```
unsigned int flash_read(unsigned int address, void *data, unsigned int length)
{
  unsigned int *paddress, *padata;

  paddress = (unsigned int *)address;
  pdata = (unsigned int *)data;

  /* lecture page par page */
  if (length <= FLASH_PAGE_SIZE) {
      while (length > 0) {
      /* Recopie des données de la mémoire vers un buffer */
      *pdata++ = *paddress++;
        length -=4;
      }
  }
  else
    return 1;

  return 0;
}
```

Figure 4.6 – Lecture de la mémoire Flash

```
unsigned int flash_write(unsigned int address, void *data, unsigned int length)
{
  …

  /* Récupération de l'identifiant d'une page à partir de son adresse  */
  page = (address - (unsigned int)FLASH_START_ADDRESS) / FLASH_PAGE_SIZE;

  paddress = (unsigned int *)address;
  pdata = (unsigned int *)data;

  /* Déverrouillage des protections de la page */
  AT91C_BASE_MC->MC_FCR = AT91C_MC_CORRECT_KEY |
                    AT91C_MC_FCMD_CLR_GP_NVM | (AT91C_MC_PAGEN & (page << 8)) ;

/* Ecriture des données */
  if (length <= FLASH_PAGE_SIZE) {
      while (length > 0) {
        *paddress++ = *pdata++;
        length -=4;
      }
  }
  else
    return 1;

  AT91C_BASE_MC->MC_FCR = AT91C_MC_CORRECT_KEY |
                    AT91C_MC_FCMD_START_PROG | (AT91C_MC_PAGEN & (page << 8) ) ;

  while(!(AT91C_BASE_MC->MC_FSR & AT91C_MC_FRDY));

  return 0;
}
```

Figure 4.7 – Ecriture dans la mémoire Flash

Comme indiqué dans le chapitre 3, suivant les cas, un ou plusieurs buffers de pré-programmation (ou buffer cache d'écriture) sont utilisés pour insérer les données dans la mémoire Flash. La taille de ces buffers est choisie pour contenir une quantité de données en accord avec les caractéristiques de la mémoire Flash au niveau de la programmation. Dans le cas du microcontrôleur AT91SAM7S256, la taille de ces buffers peut contenir une quantité de données multiples de 4 octets. Si les données sont plus petites que le buffer, des éléments dits de bourrage (« padding ») sont ajoutés. Ce bourrage est en fait réalisé à l'initialisation du buffer. Le caractère de bourrage utilisé doit être choisi en fonction de l'application. Il doit naturellement être différent des valeurs qui peuvent être potentiellement collectées. Pour des grandeurs physiques variant de 0 à 50, le caractère « @ » (valeur ASCII de 64) peut, par exemple, être utilisé.

Avant d'effectuer des opérations sur la mémoire Flash, une étape d'initialisation est nécessaire (voir Figure 4.8).

```
void flash_init()
{
  AT91C_BASE_MC->MC_FMR = (((MCK / 666666) << 16) & AT91C_MC_FMCN) |
                          AT91C_MC_FWS_1FWS;
}
```

Figure 4.8 – Initialisation de la mémoire Flash

Ces opérations d'écriture et de lecture de la mémoire Flash sont intégrées respectivement dans la primitive Out() et dans la primitive In() liées au concept LINDA étendu. Le système LiveFile gère deux formats de données : les enregistrements de données et les flux de données ou fichiers. La sauvegarde d'informations système est obtenue à l'aide de points de sauvegarde appelés « checkpoint ». Une fonction est dédiée à l'insertion de ces différents types de données. Celles-ci sont appelées à partir de la primitive Out() (voir Figure 4.9, Figure 4.10 et Figure 4.11).

L'insertion d'un fichier diffère de celle d'un enregistrement par le fait qu'elle peut requérir plusieurs pages de mémoire Flash. A l'aide de l'argument « size » qui fournit la taille du fichier à insérer, le nombre de pages de mémoire Flash nécessaires va être recherché. Après l'écriture des données dans la mémoire Flash, les pages utilisées pour stocker le fichier seront répertoriés pour pouvoir le récupérer par la suite. Seules les pages totalement remplies sont programmées. Les données restantes sont stockées dans un buffer de pré-programmation. Cependant, la page devant stocker ces données est connue et fait partie de la liste de celles utilisées pour stocker le fichier.

L'attribut « data_owner » est facultatif. Son utilisation est nécessaire si la fonctionnalité de sauvegarde des données sur plusieurs capteurs est activée. Pour éviter la multiplication des buffers de pré-programmation, les données issues d'un autre capteur seront directement stockées dans la mémoire Flash même si leur taille n'atteint pas le seuil d'écriture. Dans le cadre des points de sauvegarde, les données sont également écrites directement dans la mémoire Flash.

Comme indiqué précédemment, pour économiser de la mémoire, plusieurs caractères de type « char » d'une taille d'un octet peuvent être utilisés à la place d'un entier de type « int » d'une taille de quatre octets. L'adresse d'une page au format « 0x00100000 » peut être ainsi stockée sur trois caractères plutôt qu'un entier de type long. Le stockage de l'adresse d'une page plutôt que son numéro permet de se détacher de l'organisation propre au microcontrôleur AT91SAM7S256 avec ces 1024 pages de mémoire Flash.

```
int record_insert(unsigned char *data, unsigned int size)
{
  …
  /* Initialisation du pointeur d'écriture du buffer */
  if (pbuffcour->indice == 0)
     pbuffcour->indice = FLASH_PAGE_HEADER;

  /* Test du seuil d'écriture */
  if ((pbuffcour->indice + size) <= WRITE_THRESHOLD) {
    /* Insertion des données au buffer de pré-programmation */
    …
  }
  else {
    /* Recherche d'une page libre */
    if (savepage_search(&page_address) == -1)
      return -1;

    page = (unsigned char *)page_address;

    /* Modification du nombre d'écritures de la page */
    nbiw = get_nbwrite(page);
    nbiw++;
  …

    /* Récupération de l'espace de la page libre après l'insertion des données */
    if (pbuffcour->indice < FLASH_PAGE_SIZE)
      tab_free_block[ind_free_block++] = FLASH_PAGE_SIZE – pbuffcour->indice ;

    while(flash_write(page_address,(void*)pbuffcour->buff, pbuffcour->indice));

    /* Copie des données non insérées dans le buffer d'écriture */
    …
  }

    return 0;
}
```

Figure 4.9 – Description de la fonction « record_insert() »

La fonction « savepage_search() » est en charge de deux opérations :

- la recherche de pages libres ;
- la préservation des pages trop utilisées.

Pour la recherche d'une page libre, un appel à la fonction « freepage_search() » est réalisé.

```
int file_insert(unsigned char* data, unsigned int size) {
        Estimation du nombre de pages nécessaires au stockage du fichier
                Réservation des pages nécessaires au stockage

        Stockage des données dans la mémoire Flash
                Ecriture des pages pleines
                Liste des pages utilisées pour le stockage du fichier

        Stockage dans le buffer de pré-programmation des données restantes
                Conservation de la dernière page en attente d'être programmée}
```

Figure 4.10 – Fonctionnement de la fonction « file_insert() »

```
void Out(unsigned char *Key, unsigned char *data, unsigned int size)
{
  …
  struct Tuple *tupleptr;

  if (Key[0] == 'f')
  {
    /* Décomposition de la clé */
    explode_key(Key, data_type, data_owner, ip_address);

    /* Insertion d'un enregistrement */
    if (strcmp(data_type,"RECORD") == 0)
      record_insert(data, size, data_owner, ip_address);

    /* Insertion d'un fichier */
    if (strcmp(data_type,"FILE") == 0)
      file_insert(data, size, data_owner, ip_address);

    /* Insertion d'un point de sauvegarde */
    if (strcmp(data_type,"CHECKPOINT") == 0)
      checkpoint_insert();
  }
  else
  {
    /* Récupération du tuple recherché */
    cle = atoi((const char *)Key);
    tupleptr = tab_tuple[cle];

    while (i < size) {
    /* Test au niveau de la file de message */
    if (tupleptr->tuple_wptr >= (unsigned char *) tupleptr ->tuple_endadr)
      tupleptr ->tuple_wptr = (unsigned char *) tupleptr ->tuple_staadr;

    /* Insertion des données dans le tuple */
    *tupleptr ->tuple_wptr++ = *data++;
    i++;
    }

    tupleptr->tuple_msgnum++;

    /* Changement de l'état du tuple */
    if (tupleptr->tuple_state == TUPLE_NOUSED )
      tupleptr->tuple_state   = TUPLE_USED ;
  }
}
```

Gestion de la mémoire

Gestion des tuples

Figure 4.11 – Description de la primitive Out()

Une page contenant un point de sauvegarde est divisée en plusieurs parties dont le nombre dépend de la configuration du système LiveFile. Ces parties sont relatives aux types d'information suivants :

- la liste des pages libres ;
- la liste des emplacements libres ;
- la liste des pages préservées ;
- la liste des pages utilisées par un fichier ;
- la liste des informations liées à la gestion des propriétés.

D'autres types d'informations pourront être rajoutés suivant l'évolution des fonctionnalités du système LiveFile. La quantité d'informations système stockée dans la mémoire Flash doit être proportionnellement faible par rapport à la quantité de données collectées. L'importance des différentes informations système et donc, de la nécessité de les sauvegarder ou non dans la mémoire Flash est établie en étudiant les caractéristiques de l'application supportée.

Au niveau de la lecture des données, l'argument « Key » est de nouveau mis à contribution (voir Figure 4.12).

```
void In(unsigned char* Key, unsigned char* data, unsigned int* size) {
…
 if ((Key[0] == 'f') || (Key[0] == 'r'))
 {
  explode_key(Key, command, data_owner, ip_address);

  /* lecture de données */
  if (Key[0] == 'f')
   {
   /* Données d'une page */
   if (command[0] == 'P')
    datapage_search(data, size, command);

   /* Données d'un enregistrement */
   if (command[0] == 'R')
    recorddata_search(data, size, command);

   /* Données d'un fichier */
   if (command[0] == 'F')
    filedata_search(data, size, command);
  }

  /* interrogation des données */
  if (Key[0] == 'r')
   query_process(data, size, command);

 }
 else {
  cle = atoi((const char*)Key);
  tupleptr = &tab_tuple[cle];

  /* Récupération des données du tuple */
  while (*tuple->tuple_rptr++ == EOM) {
   if (tupleptr->tuple_rptr >= (unsigned char *)tupleptr->tuple_endadr)
    tupleptr->tuple_rptr = (unsigned char *)tupleptr->tuple_staadr;

   *data++ = *tupleptr->tuple_rptr;
   *size++;
  }

  tupleptr->tuple_msgnum--;

  /* Changement de l'état du tuple */
  if (tupleptr->tuple_msgnum == 0)
   tupleptr->tuple_state = TUPLE_NOUSED;
 }
}
```

Gestion de la mémoire

Interrogation des données

Gestion des tuples

Figure 4.12 – Description de la fonction In()

Le format de l'argument « Key » tel qu'il est défini dans le système LiveFile permet un accès aux données d'une page, d'un enregistrement ou d'un fichier. La primitive In() est, en fait divisée en trois parties :

- accès à la mémoire Flash ;
- interrogation des données ;
- gestion des tuples associés aux événements ou processus.

L'utilité de pouvoir récupérer le $j^{ième}$ enregistrement d'une page « i » est à évaluer en fonction de l'application supportée. En règle générale, certains traitements associés à une fonctionnalité jugée non indispensable pour un type d'applications donné, peuvent être retirés pour réduire la taille du système LiveFile. Des fonctionnalités restent à ajouter aux différentes procédures de recherche présentées afin de prendre en compte les paramètres « data_owner » et « ip_address ». Il serait ainsi possible de récupérer à partir du capteur A, les données du capteur C (« data_owner ») qui pourrait être stockées au sein du capteur B (« ip_address »).

4.2.3 Fonctions pour l'interrogation de données

Pour accéder aux données suivant leurs caractéristiques plutôt que suivant leur format de stockage, le système LiveFile a été muni d'un module d'interrogation s'appuyant sur le langage SQL. Selon le type de requête, leur prise en compte et les traitements associés sont réalisées par les primitives In() ou Out(). Cette dernière a été de nouveau modifiée pour permettre la réalisation de la clause « INSERT INTO » (voir Figure 4.13).

```
void Out(unsigned char *Key, unsigned char *data, unsigned int size)
{
…
if (Key[0] == 'r')
     query_insert(Key, data, &size);
…
}
```

Figure 4.13 – Interrogations de données à l'aide de la primitive Out()

L'argument « Key » est de nouveau mis à contribution avec, dans ce cas, l'emploi du caractère « r » à la place du caractère « f ». La clause « INSERT INTO » permet uniquement d'insérer des données localement dont le propriétaire est le capteur sur lequel la requête est exécutée.

La clause « DELETE FROM », pourrait être implémentée dans l'une ou l'autre primitive. Dans la primitive Out(), elle correspondrait à l'ajout d'un élément vide à la place d'un enregistrement ou d'un fichier existant. Puisque la suppression d'un enregistrement peut impliquer la recherche de la page où il est stocké, cette clause fait partie de celles à la charge de la primitive In(). Parfois, il arrive de demander la suppression du $k^{ième}$ enregistrement sans avoir l'information sur la page utilisée pour le stocker. Dans tous les cas, cette clause n'entraîne pas de programmation de la mémoire Flash. Elle indique simplement dans les structures prévues à cet effet la libération d'un emplacement ou d'une page. La primitive In() représente donc un point d'entrée pour la quasi-totalité des clauses SQL disponibles sous le système LiveFile.

La présence de ces deux mots clés SQL fournit à l'utilisateur une interface standard et répandue pour l'insertion et la suppression d'enregistrements dans la mémoire Flash. A celles-ci, vient s'ajouter le mot clé « SELECT » qui associé à la clause « WHERE » autorise la recherche d'enregistrements répondant à des critères de sélection donnés.

L'interprétation des requêtes employant la clause « SELECT » est réalisée en différentes étapes au sein de cette fonction :

- Traitements des éventuelles conditions ;
- Recherche des enregistrements ou tuples répondant aux conditions souhaitées ;
- Tri des enregistrements ;
- Extraction des attributs ;
- Opérations d'agrégation sur les attributs.

Les requêtes reçues sont décomposées et chacune de leurs conditions présentes sont stockées dans une structure adaptée (voir Figure 4.14).

```
typedef struct condition{
        unsigned char att[MAX_SIZE_ATT];
        unsigned int pos_att;
        unsigned char op[3];
        unsigned char value;
        unsigned char lien[4];
} cond_;
```

Figure 4.14 – Stockage des conditions d'une requête

L'attribut « op » stocke un des opérateurs mathématiques suivants : « = », « < », « <= », « >= », « > » et « <> ». L'attribut lien contient les opérateurs logiques « AND » et « OR » permettant de relier plusieurs conditions. L'élément « E » a été rajouté pour indiquer qu'il s'agit de la dernière condition de la requête.

La fonction « query_process() » fait appel à différentes fonctions pour établir le résultat d'une requête (Figure 4.15).

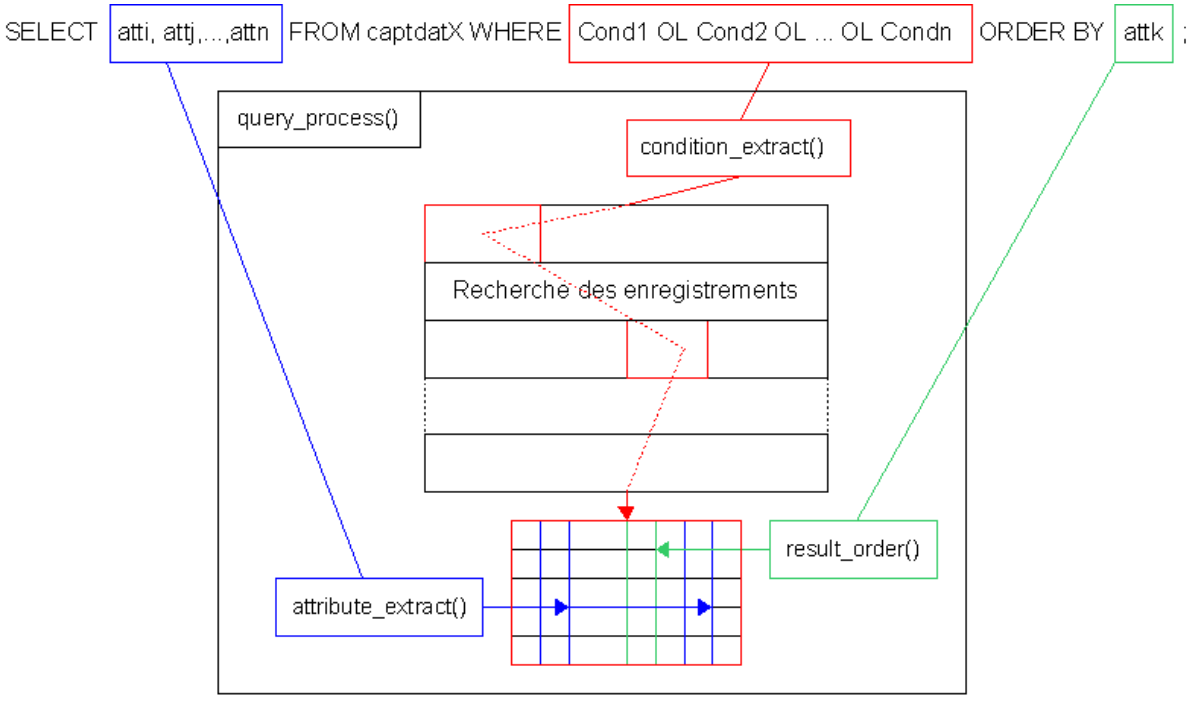

Figure 4.15 – Fonctionnement de l'interrogation de données

La première fonction appelée est « condition_extract() » dont le rôle est d'extraire et de traiter les conditions de sélection d'une requête. L'ensemble des conditions associées à une requête sont stockées dans un tableau prévu à cet effet « tab_condition ». La taille de ce tableau c'est-à-dire le nombre maximal de conditions de sélection possibles dans une requête est déterminé suivant l'application. S'agissant d'interrogations de données présentes dans un système embarqué, le nombre de conditions ne devrait pas dépasser 4.

L'étape suivante consiste à rechercher les enregistrements répondant à ces conditions de sélection. Les enregistrements obtenus sont ensuite triés, si nécessaire, suivant un attribut donné. Par défaut, le tri s'effectue par ordre croissant. La fonction « attribute_extract() » sélectionne les attributs demandés. Le résultat d'une requête est enfin stocké sous différents formats :

- tableau d'enregistrements ;
- chaînes de caractères au format prédéfini.

La seconde solution permet de réduire l'espace mémoire pour stocker les données sélectionnées mais peut nécessiter, en cas d'éléments manquants, la présence d'un caractère séparateur.

4.2.4 Fonctions avancées de gestion de données

Les différents traitements associés aux métadonnées contextuelles et aux propriétés interviennent au sein de certaines des fonctions qui ont été décrites jusqu'à présent dans ce chapitre. Cette section donne des indications sur la mise en place de ces fonctions de gestion des données avancées.

4.2.4.1 Utilisation des métadonnées contextuelles

Dans le chapitre précédent, les métadonnées ont été présentées comme une méthode pour réduire la quantité de données stockées et/ou transmises. Elles permettent donc d'économiser de la mémoire et de l'énergie. Suivant le dictionnaire d'interprétation utilisé, les métadonnées fournissent un premier niveau de sécurisation des données.

Les traitements relatifs à l'utilisation des métadonnées interviennent au niveau de différents composants du dispositif et essentiellement au sein du microsystème de fichiers LiveFile. Quand les métadonnées sont appliquées au niveau du stockage, les tests et conversions sont réalisés avant l'insertion dans le buffer de pré-programmation. En ce qui concerne la transmission, les données issues d'une page ou provenant du résultat d'une requête SQL sont transformées avant d'être fournies au module de communication.

Le dictionnaire d'interprétation est une structure stockant, pour chaque attribut de l'enregistrement, la valeur de la métadonnée utilisée. Si l'on reprend l'exemple de la section 3.3.2.1 du chapitre 3, la valeur « 0 » indique que la donnée issue du capteur de température ne subit pas de changement, la valeur « 1 » qu'elle est arrondie et la valeur « 2 » implique une recherche de l'intervalle dans laquelle la mesure de la température est présente.

Les informations présentes dans le dictionnaire d'interprétation sont ensuite utilisées par la fonction « datatometada() » pour effectuer les traitements requis (voir Figure 4.16). Cette fonction gère également l'utilisation potentielle de plusieurs dictionnaires d'interprétation. Les données stockées sous forme d'enregistrement sont converties en chaînes de caractères pour pouvoir être insérées directement dans la mémoire Flash. Cette conversion est réalisée par la fonction « datatochar() ». Les traitements relatifs à une métadonnée contextuelle peuvent s'appliquer à plusieurs attributs. Le calcul d'une moyenne ou d'un

arrondi peut s'appliquer autant à une valeur de température qu'à celle d'une pression. En revanche, certaines opérations sont propres à un type de données. Les conversions d'une date en nombre de minutes ou de secondes écoulées depuis le début de l'année en est un exemple.

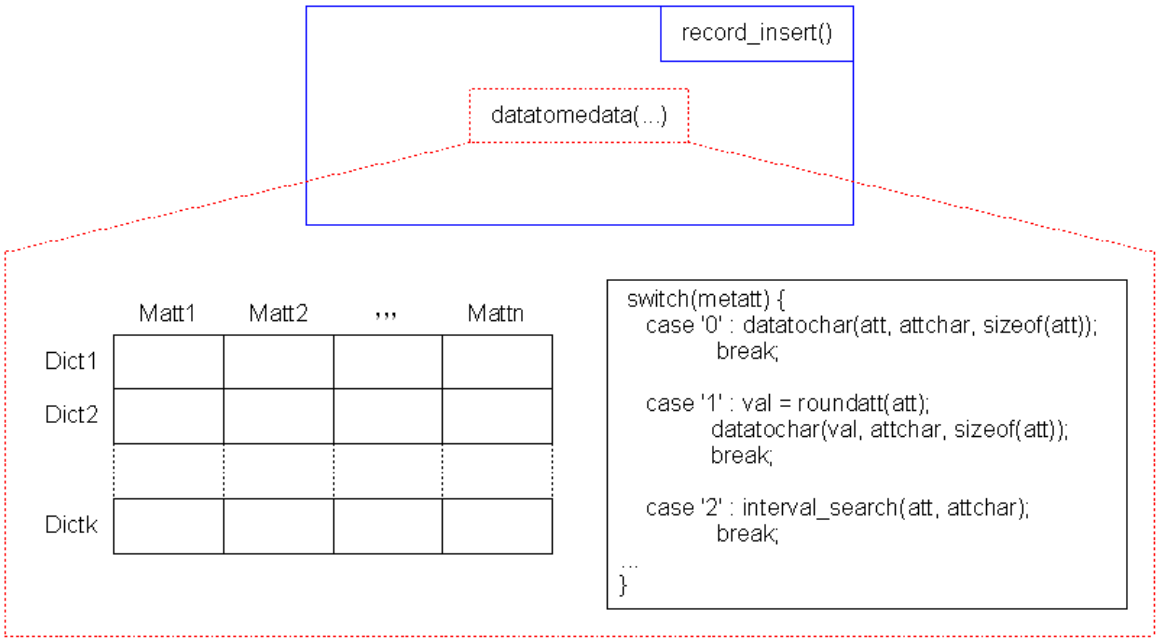

Figure 4.16 – Fonctionnement des métadonnées dans le stockage des données

Les performances du système sont liées aux traitements engendrés par les métadonnées contextuelles. Ceux-ci doivent bénéficier d'une implémentation optimisée à la fois en temps d'exécution et en taille de code. Pour les traitements lourds, une réflexion doit être menée pour déterminer l'emplacement le plus adapté à les héberger soit sur le capteur sans fil, soit au niveau de la station de collecte des données.

4.2.4.2 Utilisation des propriétés

Les propriétés présentes dans les données collectées sont établies avant leur stockage. Suivant la politique de réservation des pages choisies, l'utilisation d'un buffer de pré-programmation peut complexifier la gestion des propriétés. La méthode la plus simple consiste à réserver une page ou un emplacement mémoire avant de recevoir des données. L'emplacement de stockage est ainsi connu à l'avance et est indiqué directement dans la structure dédiée aux propriétés. L'autre méthode s'appuie sur une allocation de page ou d'emplacement seulement après avoir rempli le buffer de pré-programmation. Dans ce cas, un stockage intermédiaire des propriétés relatives aux enregistrements présents dans ce buffer est nécessaire pour pouvoir rajouter, par la suite, l'information sur l'emplacement de stockage finalement utilisé. Suivant le taux et la vitesse de renouvellement des données, l'une ou l'autre méthode est préconisée.

Comme pour les métadonnées contextuelles, les propriétés d'un enregistrement sont établies avant son stockage dans la mémoire Flash à partir d'un ou d'un ensemble de ses attributs. Si le rôle des métadonnées contextuelles est de prétraiter les données avant le stockage ou la transmission, celui des propriétés est d'accélérer l'interrogation. Ainsi, le moteur d'interrogation a été modifié pour pouvoir supporter des requêtes à base de propriétés (voir Figure 4.17).

Les principales fonctions impliquées dans la gestion des propriétés sont « prop_extract() » et « prop_select() ». La première permet de déterminer les caractéristiques ou propriétés des enregistrements et est appelée depuis la fonction « record_insert() ». La seconde fonction intervient dans la fonction « query_process() » et sert à interroger les enregistrements stockés en mémoire Flash suivant une ou plusieurs propriétés données.

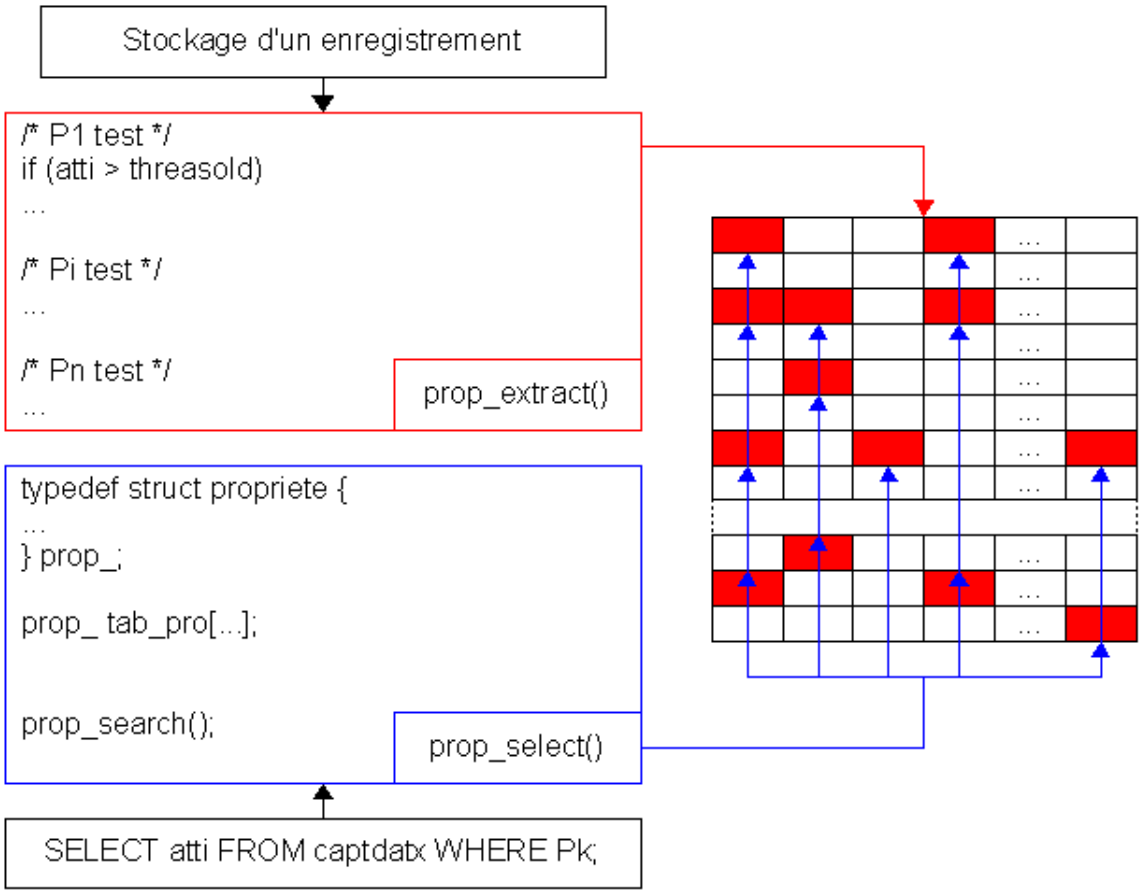

Figure 4.17 – Fonctionnement des propriétés

Différentes méthodes sont utilisables pour stocker les renseignements sur les propriétés :

- Utilisation de champ de bits ;
- Découpage en bits d'un caractère ;
- Structure à base de caractères.

Comme dans le cas des adresses de mémoire Flash, chaque mode de stockage des propriétés a ses avantages et ses inconvénients. Par exemple, les solutions qui minimisent la taille mémoire impliquent des traitements de pré et post-stockage (voir Figure 4.18).

```
typedef struct propriete{
  unsigned int page : 22;
  unsigned int idrec : 8;
  unsigned int p1 : 1;
  unsigned int p2 : 1;
} prop_;
```

```
typedef struct propriete2{
  unsigned int page;
  unsigned char idrec;
  unsigned char p1;
  unsigned char p2;
} prop2_;
```

Figure 4.18 – Différentes solutions pour le stockage des propriétés

L'espace mémoire disponible et envisagé pour stocker les informations propres aux propriétés conditionne leur emploi. En effet, cette technique d'accès plus rapide aux données est obtenu au détriment d'une consommation plus importante des ressources en puissance de calcul et de mémoires à la fois RAM et Flash. Le tableau contenant les informations sur les propriétés est stocké en mémoire vive avant une éventuelle programmation en mémoire Flash dans le cadre d'une procédure de sauvegarde du système (« checkpoint »).

4.2.5 Nécessité d'une interface dédiée à la gestion de données

Une interface d'accès aux données et mise à disposition de l'utilisateur est envisageable pour un nombre conséquent d'applications de RCSF. Par exemple, une application d'acquisition de données comprend une partie de collecte d'informations à l'aide de capteurs de grandeur physique mais également la transmission de celles-ci à l'utilisateur. L'interface serait hébergée au sein d'une des stations centrales de collecte. Dans le cadre d'une application de substitution d'une infrastructure de réseau fixe, la quantité de données stockées en mémoire par un capteur sans fil ainsi que la durée de ce stockage peut servir d'indicateurs sur l'état de fonctionnement du réseau. En effet, dans ce type d'application, le stockage en mémoire Flash n'intervient que lorsque le capteur sans fil est dans l'incapacité de relayer les informations reçues.

En ce qui concerne la gestion des données au sein de la plateforme LiveNode, l'intérêt de disposer d'une interface utilisateur est multiple. Tout d'abord, certaines commandes pour accéder aux données pourraient être exécutées directement par l'utilisateur final. Le formatage et le mode d'affichage des données pourraient également être spécifiés par celui-ci.

L'activation à distance des modules de gestion de données fait partie des applications envisagées. L'utilisateur pourrait intervenir de deux manières différentes. La première consiste à explicitement activer ou désactiver un module comme le moteur d'interrogation. La difficulté majeure réside dans la prise en compte du changement de configuration par le système d'exploitation. La seconde s'appuie sur le fonctionnement des modules utilisés. Par exemple, la définition d'un ensemble de propriétés par l'utilisateur impliquerait tacitement l'activation du module dédié à leur utilisation. Ce mécanisme ne pourra être possible qu'avec l'existence d'une interface permettant de définir ces propriétés.

Dans un premier temps, l'ensemble des composants logiciels formant les différents modules de gestion des données sont déjà présents au sein des capteurs sans fil. Dans un second temps, avec l'évolution de la plateforme, les modules manquants au sein d'un capteur pourraient être téléchargés à partir d'un autre capteur ou d'une station centrale de collecte. Ce mode de fonctionnement à base de mises-à-jour dynamiques des composants logiciels d'un capteur est beaucoup complexe et demande encore un certain nombre de développements.

L'interprétation d'une requête SQL par le moteur d'interrogation est une étape impliquant un certain nombre de traitements qui comprennent essentiellement l'extraction des conditions de sélection et des attributs. Pour réduire la quantité de ressources consommées par cette fonctionnalité, des prétraitements sur les requêtes peuvent être envisagés avant leur transmission au sein du RCSF. Ainsi, l'interface pourrait, en plus d'offrir un point d'entrée au niveau du RCSF pour les interrogations soumises par l'utilisateur, convertir les requêtes classiques ou faisant appel à la notion de propriétés. Les exemples suivants permettent d'illustrer ces conversions :

- « SELECT temp FROM captdat1 ; » => « captdat1, temp ; » ;
- « SELECT P1 FROM capdat1 ; » => « captdat1, P1 ; » ;

Cette conversion effectuée par l'interface en amont économiserait d'une part de l'énergie en diminuant la quantité de bits transmis. D'autre part, en prenant l'hypothèse valide que la station centrale où seraient effectués les prétraitements disposerait d'une capacité de calcul plus importante que les capteurs, le temps d'exécution des requêtes serait également réduit.

Les métadonnées contextuelles peuvent également être utilisées. La requête suivante INSERT INTO captdat1(temp, localisation, date) VALUES (12, 'A', 10) ; correspond à l'insertion de la température « 12 » collectée dans la zone « A » à la date du « 10 Janvier ». Après sa transformation au niveau de l'interface, elle devient :

- Conversion classique : « captdat1, temp=12, localisation=A, date=10 ; » ;
- Conversion avec utilisation des métadonnées contextuelles : « +01001000001010 ; » ;

Les clauses « INSERT » et « DELETE » pourront être associées respectivement aux opérateurs « + » et « - ». La syntaxe utilisée doit prendre en compte les différents types de requêtes et être une vraie source d'économie de ressources.

La conception et le développement d'une telle interface a été initiée et fait partie des travaux de recherche en cours. L'application hébergeant cette interface pourrait recevoir des informations sur le routage provenant du RCSF et ainsi prendre des décisions sur la politique d'exécution d'une requête donnée. La distribution au sein du RCSF des traitements liés à la gestion des données sont une problématique qui reste encore à étudier.

4.3 Cas d'utilisation

La plateforme LiveNode a été et continue d'être évaluée dans le cadre de différents projets. Le capteur sans fil LiveNode et le noyau temps réel LIMOS ont été testés dans le cadre du projet MobiPlus [Zhou 2006a]. Cette première application a montré l'intérêt de disposer d'un module dédié à la gestion des données. Des fonctionnalités importantes ont été ajoutées au niveau de la plateforme LiveNode grâce à l'intégration du microsystème de fichiers LiveFile. Actuellement, des travaux de recherche sont menés pour compléter cette plateforme afin d'en faire un dispositif adapté à l'acquisition de données environnementales [De Sousa 2008].

4.3.1 L'application MobiPlus

L'application MobiPlus ou « Système Autonome d'Aide aux Personnes à Mobilité Réduite » s'intègre dans un projet de recherche à l'initiative du Syndicat Mixte des Transports en Commun de l'agglomération clermontoise (SMTC). Son objectif est l'amélioration de l'accessibilité aux transports en commun des personnes à mobilité réduite.

4.3.1.1 Présentation de l'application

Seul le mode de transport par bus est visé par le projet. Contrairement à la plupart des métros ou des tramways, rares sont les bus dont le plancher arrive à la même hauteur que le trottoir. Les personnes en fauteuil roulant, utilisant des béquilles et malvoyantes ont ainsi des difficultés pour accéder au bus.

Pour remédier à cela, une solution a été proposée dans le cadre du projet MobiPlus. Celle-ci s'appuie sur différentes technologies de communication sans fil et est divisée en deux parties :

- le sous-système « quai » ;
- le sous-système « bus ».

Le sous-système « quai » est constitué d'une borne dans laquelle est installé un capteur sans fil LiveNode relié à un lecteur RFID (voir Figure 4.19). Chaque personne est munie d'un badge ou d'un tag RFID contenant uniquement le type d'handicap dont elle souffre. Lors de son passage près d'une borne située à un arrêt de bus, le lecteur RFID présent détecte la présence du tag et envoie des informations au capteur sans fil LiveNode (voir Figure 4.20).

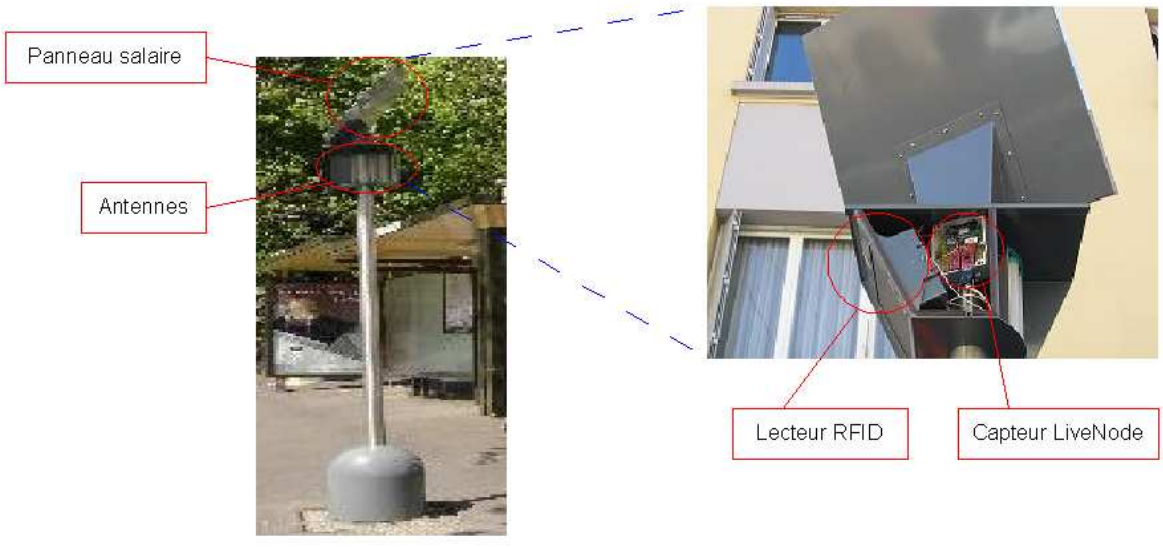

Figure 4.19 – Sous-système « quai » de l'application MobiPlus

Ce dernier va ensuite transmettre, grâce aux deux antennes de la borne, ces informations en direction d'un bus en approche. Chaque antenne est dédiée à un protocole de communication différent. Dans cette application, le capteur sans fil LiveNode est en configuration multi-support Wi-Fi et ZigBee. Un panneau solaire complète ce dispositif pour rendre la borne autonome en énergie et faciliter son intégration dans le mobilier urbain.

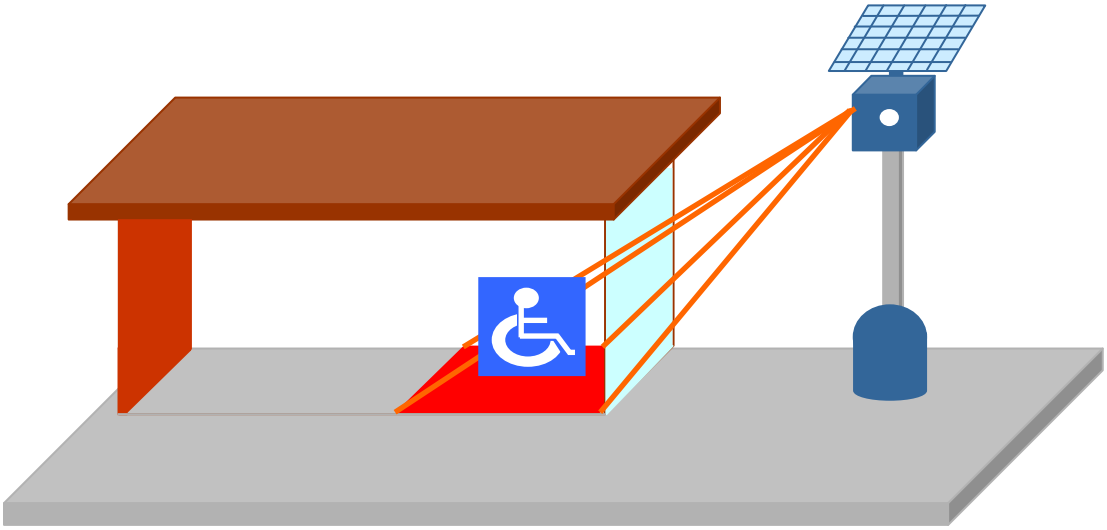

Figure 4.20 – Fonctionnement du sous-système « quai »

Quand un bus arrive à proximité de l'arrêt, ses antennes reçoivent les informations et les transmettent au capteur sans fil LiveNode du sous-système « bus ». Celui-ci va activer un annonceur sonore qui va avertir le conducteur du bus de la présence d'une personne souffrant d'un handicap. Suivant le type d'handicap annoncé, celui-ci va amorcer la manœuvre adéquate comme placer la porte du milieu du bus en face de la borne pour permettre le déploiement automatique d'une palette facilitant son accès (voir Figure 4.21).

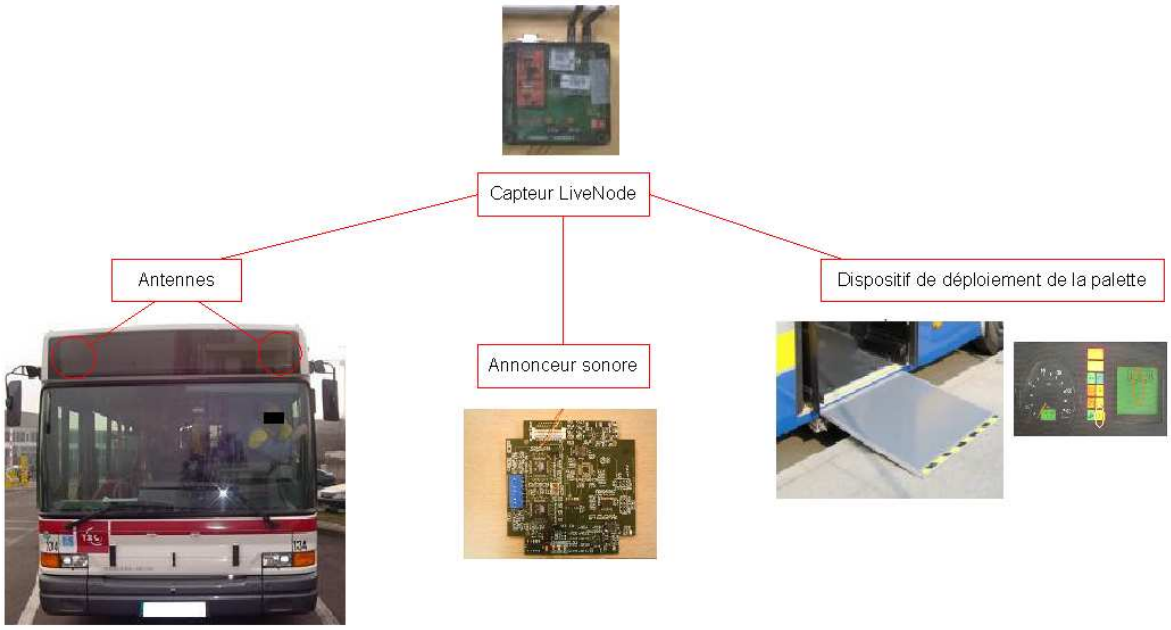

Figure 4.21 – Sous-système « bus » de l'application MobiPlus

4.3.1.2 La plateforme dédiée à l'application MobiPlus

L'application MobiPlus s'appuie sur un capteur sans fil LiveNode en configuration multi-support combinant les protocoles de communication Wi-Fi et ZigBee. Deux unités matérielles de base sont donc combinées pour obtenir cette fonctionnalité.

Quatre événements principaux sont définis dans le noyau LIMOS (voir Figure 4.22) en charge de la gestion des ressources du capteur sans fil LiveNode. Le premier événement est associé à la réception de données de la part du lecteur RFID par l'intermédiaire d'un port série RS232. Le second événement est dédié à la gestion de la communication et peut intégrer des fonctionnalités liées au routage. Les deux derniers événements sont propres à chaque protocole de communication utilisé.

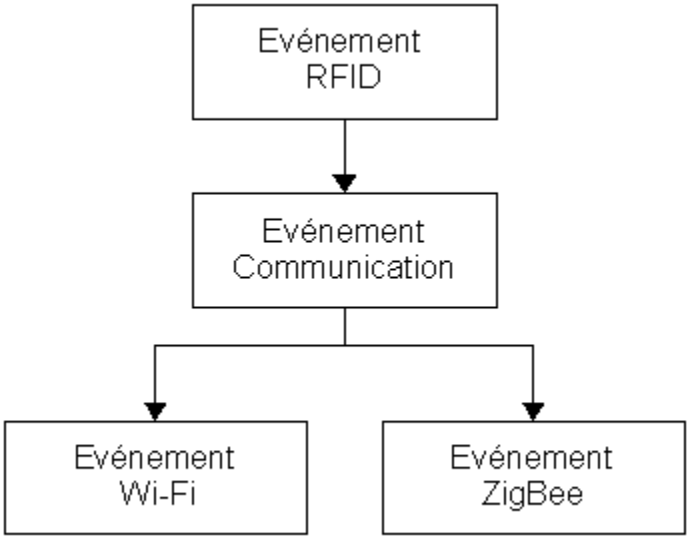

Figure 4.22 – Configuration du noyau LIMOS dans l'application MobiPlus

L'intérêt d'un système évolué pour la gestion des données apparaît dans ce type d'application pour stocker les différentes informations propres à chaque arrêt comme leur localisation géographique. D'autres données relatives à l'exploitation de l'application MobiPlus comme le taux de fréquentation peuvent également être stockée. Aucune autre information, sauf celle du type d'handicap dont souffre la personne, n'étant présente au niveau du tag RFID, l'anonymat des utilisateurs reste ainsi préservé.

4.3.2 Evaluation des performances

Un autre domaine d'application de l'ensemble de ce dispositif est l'acquisition de données environnementales. La plateforme composée du capteur LiveNode, du noyau LIMOS et du microsystème de fichiers LiveFile peut être utilisée comme outil de supervision de grandeurs environnementales. L'objet de cette partie est d'évaluer les performances, au niveau des temps moyens d'exécution, des différentes fonctions intervenant dans la gestion d'informations associées à des données environnementales.

4.3.2.1 Evaluation des fonctions pour la gestion des données

Pour illustrer ces propos, un type de données « data » est définie et correspond à une grandeur physique telle qu'une température, une pression, un taux d'humidité ou d'ensoleillement, ou un niveau de batterie (voir Figure 4.23). Les attributs associés à ce type

de données varient suivant les fonctionnalités présentes au sein du capteur sans fil comme c'est le cas pour la date d'acquisition. Cette dernière peut être obtenue soit à l'aide d'un système GPS, soit en utilisant le temporisateur interne du capteur sans fil. Cette deuxième solution peut impliquer un protocole de synchronisation plus ou moins complexe pour remettre à jour l'horloge après une phase de sommeil.

```
typedef struct data
{
  unsigned char id;
  unsigned char val;
  unsigned char jour;
  unsigned char mois;
  unsigned char annee;
  unsigned char heure;
  unsigned char min;
  unsigned char crc;
} data_;
```

Figure 4.23 – Exemple d'enregistrement

Les différentes fonctions de gestion des données du système sont évaluées sur cette structure de données nécessitant 8 octets pour son stockage. Le critère d'évaluation utilisé est le temps moyen d'exécution. Un espace mémoire d'une taille de 64Ko est réservé pour le stockage de ces données.

Les premières évaluations portent sur le fonctionnement standard de l'insertion de données (voir Table 4.1).

Catégories	Fonctions	Rôle	10 éléments	100 éléments	200 éléments
Insertion de données	flash_write()	Ecriture dans la mémoire Flash	0µs 28,836ms	86,531ms	173,063ms
	record_insert()	Insertion d'enregistrements	15,52µs 28,943ms	86,680ms	173.360ms
	Out()	Insertion de données aux formats différents	16,01µs 28.944ms	86,694ms	173,388ms

Table 4.1 – Evaluation des fonctions standards pour l'insertion de données

Pour l'insertion de 10 éléments, deux résultats sont présentés. Le premier fait référence à un comportement classique avec un seuil d'écriture fixé à la taille d'une page de mémoire Flash c'est-à-dire à 256 octets. Dans ce cas, aucun appel à la fonction « flash_write() » n'a lieu. En revanche, de nouvelles données sont ajoutées au buffer de programmation. Ce fonctionnement est celui considéré pour l'insertion de 100 et 200 éléments. Le second correspond à une écriture forcée dans la mémoire Flash et peut être obtenu en modifiant l'indice courant du buffer de programmation. Ce procédé est, par exemple, utilisé en cas de redémarrage programmé du système d'exploitation.

Les résultats de ces évaluations prennent en compte le délai à respecter entre plusieurs écritures successives. La durée de ce délai est d'environ 28 ms.

Les fonctions dédiées à l'accès aux données ont été évaluées suivant deux protocoles de test différents. Dans le premier, les enregistrements sont stockés de manière contiguë dans des pages dont l'adresse est connue (voir Table 4.2).

Catégories	Fonctions	Rôle	10 éléments	100 éléments	200 éléments
Accès aux données	flash_read()	Lecture de la mémoire Flash	10,46µs	41,84µs	73,22µs
	recorddata_search()	Recherche d'enregistrements	12,02µs	63µs	94,98µs
	In()	Accès aux différentes données stockées	12,14µs	64,12µs	95,1µs

Table 4.2 – Evaluation des fonctions standards pour l'accès aux données

Le second protocole vise à tester ces fonctions dans le pire des cas. Celles-ci sont appelées autant de fois qu'il y a d'éléments à accéder et, la mémoire est parcourue dans son intégralité à chaque fois.

Catégories	Fonctions	Rôle	10 éléments	100 éléments	200 éléments
Accès aux données	flash_read()	Lecture de la mémoire Flash	26,777ms	267,662ms	535,644ms
	recorddata_search()	Recherche d'enregistrements	27,391ms	273,101ms	546,234ms
	In()	Accès aux différentes données stockées	27,393ms	273,155ms	546,267ms

Table 4.3 – Evaluation des fonctions standards pour l'accès aux données au pire des cas

La fonction « file_insert() » est évaluée sur des fichiers de différentes tailles (voir Table 4.4). Contrairement à l'évaluation précédente, l'intégralité des données du fichier sont réellement écrites en mémoire Flash. Cela correspond à un fonctionnement où l'on libère le buffer de programmation après l'insertion d'un fichier. La taille d'une page de mémoire Flash (256 octets pour le microcontrôleur AT91SAM7S256) est un élément à prendre en considération dans l'interprétation des résultats obtenus.

Taille du fichier (en octets)	64	128	256	512	1024
File_insert()	28,957ms	28,959ms	28,965ms	57,809ms	115,497ms

Table 4.4 – Evaluation de la fonction « file_insert() »

Quand le fichier doit être stocké sur plusieurs pages, du fait de sa taille ou de la fragmentation de la mémoire Flash, le temps d'exécution est plus important. A l'inverse, peu de différence existe entre un fichier de 64 octets et un autre de 128 octets.

La fonction « checkpoint_insert() » produit une photographie instantanée du système au niveau de la gestion des données pour permettre sa restauration en cas de panne. Le temps

moyen d'exécution de cette fonction est de 28,868 ms en considérant une quantité d'informations sauvegardées inférieure à la taille d'une page.

4.3.2.2 Evaluation des fonctions pour l'interrogation

L'interrogation de données s'appuie sur la fonction « query_process() » qui est en charge de l'interprétation des requêtes soit directement, soit en faisant appel à d'autres fonctions. Par exemple, une requête contenant une condition de sélection identifiée par la présence de la clause « Where » demandera l'utilisation de la fonction « condition_extract() ».

Les temps d'exécution sont liés à la quantité de données stockées et à l'espace dédié à leur stockage. Pour cette évaluation, les résultats sont fournis pour des enregistrements soit stockés dans un bloc de pages connues voire contiguës, soit répartis aléatoirement, à partir de la première page, dans l'ensemble de l'espace mémoire de 64Ko. Les résultats donnent une indication sur les performances pouvant être obtenues en cas de connaissance de l'emplacement des enregistrements. Le nombre d'enregistrements insérés étant connu par le système, la recherche est arrêtée quand ils ont tous été trouvés.

La fonction « query_process() » est ainsi évaluée à la fois suivants le nombre de tuples présents et le type de requêtes considéré (voir Table 4.5).

Type de requêtes	Requêtes	10 éléments	100 éléments	200 éléments
Projections	Select id From data ;	27,96µs 178,25µs	79,01µs 1,623ms	95,34µs 2,913ms
	Select id, val From data Order By val ;	43,94µs 271,06µs	1,467ms 4,219ms	5,433ms 13,530ms
Sélections	Select id,val From data Where (val > 10) ;	34,86µs 194,35µs	85,90µs 1,662ms	101,94µs 2,974ms
	Select id,val From data Where (val > 10) AND (val < 20) ;	37,26µs 196,39µs	89,30µs 1,675ms	105,34µs 3,036ms
	Select id,val From data Where (mois = 1) AND (heure = 12) AND (min = 0) ;	41,46µs 217,14µs	94,55µs 1,691ms	109,52µs 3,109ms

Table 4.5 – Evaluation de la fonction « query_process() »

Ces résultats montrent que le tri des données est une opération coûteuse qu'il vaut mieux, si possible, effectuer au niveau de la station centrale de collecte.

Sur un même jeu de données généré, la différence de performances entre les requêtes de sélection avec une, deux ou trois conditions est causée par la durée de l'étape initiale d'interprétation de la requête et par le ou les tests supplémentaires à effectuer sur les données.

4.3.2.3 Evaluation des fonctions de gestion avancée

La détermination des propriétés d'une donnée et/ou sa conversion suivant les métadonnées contextuelles utilisées s'effectuent au sein de la fonction « record_insert() ». Ces opérations sont à la charge respectivement des fonctions « prop_extract() » et « datatometadata() » (voir Table 4.6). Le temps d'exécution de chacune de ces fonctions est calculé sur un enregistrement. 2 propriétés sont définies pour « prop_extract() » et un attribut est converti pour « datatometadata() ».

Operations	Temps moyen d'exécution
Extraction de propriétés	0,1µs
Conversion par métadonnées contextuelles	0,3µs

Table 4.6 – Performances des fonctions « prop_extract() » et « datatometadata() »

Les propriétés ont été élaborées pour accélérer l'accès aux données stockées. Une comparaison doit donc été effectuée entre des sélections classiques de données et leurs équivalents utilisant les propriétés (voir Table 4.7). A ce titre, la propriété suivante a été définie : P1 : « val > 10 »

Requêtes	10 éléments	100 éléments	200 éléments
Select id,val From data Where (val > 10) ;	194,35µs	1,662ms	2,974ms
Select id,val from data Where P1 ;	115,78µs	0,891ms	1,748ms

Table 4.7 – Comparaison de sélections avec ou sans utilisation des propriétés

Dans cette évaluation, les données ne sont pas stockées dans des blocs de mémoire connues. Les performances affichées par l'utilisation des propriétés sont toutefois à pondérer par rapport au surplus de mémoire consommée pour leur stockage qui peut varier entre 512 et 1024 octets. Les enregistrements répondant à la propriété P1 pouvant être enregistrés dans des pages différentes, les temps affichés sont plus importants que si les données étaient stockées de manière contiguë dans un emplacement connu.

4.3.2.4 Synthèse sur l'évaluation des performances

Au sein des composants logiciels de la plateforme LiveNode que sont le noyau LIMOS et le système de fichiers LiveFile, les différentes fonctionnalités de gestion des données sont assurées par différents modules. Ceux-ci peuvent être ainsi ajoutés ou supprimés suivant les contraintes de l'application supportée et faire varier ainsi l'empreinte mémoire de l'ensemble de la partie logicielle (voir Table 4.8). Les principaux modules sont :

- le module de gestion de la mémoire Flash ;
- le module de gestion des formats de données ;
- le module d'interrogation des données ;
- le module de gestion des propriétés ;
- le module de gestion des métadonnées contextuelles.

Certains modules ont besoin de fonctionnalités présentes dans d'autres pour fonctionner correctement. En outre, le module de gestion de la mémoire Flash est un module de base sur lequel s'appuient les autres. Ces modules sont eux-mêmes composés des

différentes fonctions dont les performances au niveau temps moyen d'exécution viennent d'être fournies.

Modules	Principales Fonctions	Empreinte mémoire
Gestion de la mémoire Flash	flash_init() flash_write() flash_read()	16 octets
Gestion des formats de données	savepage_search() get_nbwrite() datapage_search() record_insert() recorddata_search() file_insert() filedata_search() checkpoint_insert()	924 octets
Interrogation des données	query_process() query_insert() condition_extract() attribute_extract() result_order()	360 octets
Gestion des propriétés	prop_extract() prop_select()	124 octets
Métadonnées contextuelles	datatometadata() datatochar()	60 octets

Table 4.8 – Empreinte mémoire des différents modules de gestion des données

Pour évaluer l'empreinte mémoire de chaque module, sont prises en compte les fonctions mais également les structures de données et autres constantes qui pourraient y être définies. De constantes modifications sont apportées aux différentes parties permettant la gestion de données au sein du capteur sans fil LiveNode. L'empreinte mémoire des différents éléments tend donc à diminuer aux grés des optimisations ou à augmenter de part l'ajout de nouvelles fonctionnalités.

4.4 Synthèse sur la plateforme LiveNode

La plateforme LiveNode au départ composée d'un capteur sans fil multi-composant LiveNode géré par un micronoyau temps-réel LIMOS a été agrémentée d'un système de fichiers LiveFile. La palette d'applications supportée a ainsi pu être élargie grâce d'une part à l'ajout de nouvelles fonctionnalités et d'autre part à la modularité des différents éléments constituant cette plateforme.

Le fonctionnement de cette plateforme a été testé durant des expérimentations se déroulant dans le cadre d'une application réelle. Un capteur sans fil multi-support, privilégiant la robustesse de la communication, obtenu par association de deux capteurs LiveNodes a été utilisé. Le pilotage de celui-ci était à la charge du système d'exploitation LIMOS dont la fonction principale était d'assurer la bonne connexion entre le capteur et les autres dispositifs formant l'application.

L'ajout de fonctionnalités pour la gestion des données ouvre de nouvelles perspectives au niveau de l'utilisation de la plateforme dans le domaine de l'acquisition de données environnementales. Dans ce type d'applications, les principales étapes sont l'acquisition en elle-même, le stockage et la mise à disposition des données. Dans la plateforme actuelle, différents formats pour la gestion et la manipulation des données sont disponibles soit sous forme brute, soit dans une structure de données adaptées. En outre, l'élaboration d'un moteur d'interrogation répond aux spécifications requises pour la restitution des données à l'utilisateur final.

Ces deux applications appartiennent à des catégories différentes. La première entre plus dans celles de substitution d'une infrastructure filaire. La seconde fait totalement partie des applications d'acquisition de données. Elles tendent donc à illustrer la polyvalence de la plateforme LiveNode.

Conclusion

L'intérêt grandissant pour les réseaux de capteurs sans fil (RCSF), au niveau recherche, s'explique par les nombreuses applications envisagées alliées aux récentes avancées dans des domaines comme celui des technologies de communication sans fil. Les RCSF sont à l'intersection de différentes thématiques de recherche et s'accompagnent donc d'un ensemble de problématiques. La plupart d'entre elles dérivent des réseaux sans fil Ad Hoc ou sont liées aux ressources limitées des capteurs sans fil. Ces problématiques ne peuvent pas être toutes abordées en même temps. Cette thèse est centrée sur celle de la gestion des données au sein d'un capteur sans fil.

Bilan

La principale contribution de ce mémoire est la conception et la réalisation d'un microsystème de fichiers dédiés aux RCSF. Ce système a été élaboré progressivement en s'attachant, en premier lieu, à la gestion de la mémoire. Des fonctions ont ainsi été développées pour permettre une manipulation intelligente de la mémoire non volatile de type Flash. Elles comprennent des fonctions de base pour accéder à la mémoire Flash en lecture ou en écriture ainsi que d'autres, plus évoluées, offrant différents dispositifs permettant une utilisation raisonnée visant, par exemple, à préserver celle-ci.

Sur le même principe, des fonctionnalités de gestion de données classiques et d'autres plus avancées ont ensuite été ajoutées. Le système LiveFile permet de manipuler les deux types de données que sont les enregistrements et les fichiers. Un enregistrement est utilisé pour stocker les informations relatives à une grandeur physique observée. Le stockage s'effectue sous la forme d'une structure de données dédiée. L'accès à ces enregistrements est possible soit directement soit par l'intermédiaire d'interrogations utilisant une syntaxe proche de celle du langage SQL.

Les fichiers correspondent à un flux de données qui sont en relation les unes avec les autres mais qui n'ont pas d'organisation particulière. Les informations système issues de la gestion de ces différents types de données sont stockées, d'abord en mémoire RAM, puis en Flash. Ce stockage en mémoire non volatile constitue un mécanisme de recouvrement après une panne de premier niveau.

Plusieurs systèmes existants portant sur la gestion de la mémoire ou des données ont été étudiés. La première originalité du système LiveFile par rapport à ceux-ci est de grouper ces deux fonctions au sein d'un même composant logiciel. La seconde concerne l'introduction de fonctionnalités de gestion avancée des données comprenant les propriétés et les métadonnées contextuelles. Les propriétés sont équivalentes à un mécanisme d'indexation pour la recherche d'informations. Les métadonnées contextuelles visent d'une part la réduction de la quantité de données soit stockées, soit transmises. D'autre part, elles peuvent également servir de mécanismes de base pour la sécurisation des données.

Ce système vient compléter une plate-forme évolutive dédiée au RCSF qui comprend un capteur sans fil multi-composant LiveNode, piloté par un noyau temps-réel LIMOS. Plusieurs capteurs sans fil LiveNode peuvent être ainsi combinés pour en former un plus complet. Par exemple, deux capteurs avec des modules de communication sans fil différents regroupés forment un dispositif matériel implémentant le concept de multi-support. Les principales particularités du noyau LIMOS sont d'une part son architecture hybride à la fois multitâche et basée sur les événements. D'autre part, la communication entre objets systèmes c'est-à-dire soit entre processus, soit entre événement et la gestion des périphériques sont assurées par une extension du concept de programmation parallèle LINDA. L'intégration du microsystème de fichiers LiveFile au sein du noyau LIMOS a été réalisée principalement en étendant ce concept et a ainsi permis d'obtenir un haut niveau d'abstraction pour la gestion des entrées/sorties.

La présence de ces trois éléments polyvalents permet d'élargir la palette d'applications supportées par la plate-forme LiveNode. Celle-ci a pu, dans un premier temps, être évaluée positivement durant un projet de recherche avec des applications réelles. L'apport du système LiveFile ouvre de nouvelles perspectives dans le cadre d'applications d'acquisition de données environnementales même si elles ne sont pas les seules. Les applications de substitution d'une infrastructure de réseau fixe peuvent bénéficier de la capacité à stocker des flux de données temporairement au sein de la mémoire Flash.

Perspectives

Ce mémoire synthétise les premiers résultats et travaux visant à apporter des solutions à la problématique de la gestion des données dans les RCSF. Les apports recensés sont divers mais ne constituent qu'une base de travail devant être complétée par un ensemble d'études et de développements. En outre, le microsystème de fichiers présenté est un premier prototype auquel de nombreuses améliorations doivent être apportées afin d'obtenir un dispositif encore moins consommateur de ressources.

En ce qui concerne les fonctionnalités, certaines relatives aux systèmes de gestion de base de données doivent être ajoutées. En outre, les capteurs sans fil disposent de plus en plus souvent d'informations sur leur localisation. L'ensemble des données disponibles au sein d'un capteur sans fil est ainsi propice à des consultations par l'intermédiaire de requêtes spatiotemporelles.

La problématique abordée étant assez complexe, les travaux réalisés se sont focalisés sur la gestion des données au sein d'un capteur sans fil. Par conséquent, la première suite à donner à ces travaux, dont il a été fait mention tout au long de ce mémoire, est de considérer la gestion de données au niveau de l'ensemble du RCSF. Cela comprend la proposition de méthodes optimales, tant au niveau temporel que de la consommation des ressources, dans la distribution et le traitement des interrogations. Ces méthodes devront être a priori fortement liées au protocole de routage utilisé comme dans le cas de la sauvegarde de données sur plusieurs capteurs.

L'introduction d'une interface dédiée à l'accès aux données et à la configuration à distance des composants logiciels de la plate-forme LiveNode, démontre la volonté d'établir un pont entre le RCSF et l'extérieur. La phase d'acquisition n'est qu'une première étape qui est généralement suivie par la mise à disposition des données pour l'utilisateur final. Cette fonctionnalité implique un certain nombre de nouveaux problèmes. Par exemple, l'emploi des stations centrales de collecte comme point d'entrée du RCSF entraîne un épuisement prématuré des capteurs sans fil situés à proximité.

Cette interface n'est qu'une première étape dans l'intégration du RCSF comme entité faisant partie d'un système d'information global. Actuellement, de nombreux travaux portent sur la conception de systèmes d'information construits à partir de différentes sources de données hétérogènes dont font partie les RCSF. Dans notre cas, l'objectif visé est l'utilisation, de manière transparente, des informations issues du RCSF à partir d'un système de gestion de bases de données classiques. Pour l'atteindre, la conception de méthodes et le développement d'outils de modélisation adaptés aux RCSF et dédiés à ce type d'applications seront probablement nécessaires.

Bibliographie

[Abolhasan 2004] M. Abolhasan, T. Wysocki, E. Dutkiewicz
A review of routing protocols for mobile ad hoc networks
Ad Hoc Network, vol. 2, no. 1, pp.1-22, January 2004.

[Abrach 2003] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker; J. Deng, R. Han
MANTIS: System Support for MultimodAl NeTworks of In-situ Sensors
2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA), San Diego, Californie, USA, pp. 50-59, 19 Septembre 2003.

[Adya 2002] A. Adya, J. Howell, M. Theimer, W. J. Bolosky, J. R. Douceur
Cooperative Task Management without Manual Stack Management
Usenix Annual Technical Conference, Monterey, California, USA, pp. 289-302, June 2002.

[Ahuja 1986] S. Ahuja, N. Carriero et D. Gelernter
LINDA and friends
IEEE Computer, vol. 19, no. 8, pp. 26-34, August 1986.

[Akyildiz 2002] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci
A Survey on Sensor Networks
IEEE Communications Magazine, vol. 40, issue 8, pp. 102-114, August 2002.

[Amamra 2004] A. Amamra, R. Aufrère, J.-P. Chanet, G. De Sousa, J.-J. Li, H.-Y. Zhou, K.-M. Hou
A New Adaptive Zone Filter to Estimate RTT in MANET
5th International Arab Conference on Information Technology, ACIT'2004, Constantine, Algeria, Décember 12-15, 2004.

[Amamra 2008] A. Amamra, K.-M. Hou
SLOT: A Fast and Accurate Technique to Estimate Available Bandwidth in Wireless IEEE 802.11
30th International Conference on Computer Modeling and Simulation, UKSIM 2008, Cambridge, UK, pp. 46-51, April 1-3, 2008.

[AMD 2001] AMD, Advanced Micro Devices
AMD Advanced Architecture Flash Memory Devices
Flash Memory Articles and Reviews, 2001.

[ARM7 2007] ARM7, Atmel Corporation
AT91 ARM Thumb-based microcontrollers
Technical Report 6175s, 2007.

[Atmega 2008] Atmega, Atmel Corporation
www.atmel.com/dyn/products/devices.asp?family_id=607, 2008.

[AVR 2008] AVR, Atmel Corporation
www.atmel.com/products/AVR/, 2008.

[Bhatti 2005] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, R. Han
MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms
ACMKluwer Mobile Networks & Applications (MONET) Journal, Special Issue on Wireless Sensor Networks, Vol. 10, Issue 4, August 2005.

[Biagioni 2002] E. Biagioni, K. Bridges
The Application of Remote Sensor Technology to Assist the Recovery of Rare and Endangered Species
International Journal of High Performance Computing Applications, vol. 16, no. 3, pp. 315-324, 2002.

[Bloch 2003] L. Bloch
Les systèmes d'exploitation des ordinateurs. Histoire, fonctionnement, enjeux
Vuibert, Février 2003.

[Bonnet 1999] C. Bonnet et I. Demeure
Introduction aux systèmes temps réel
Hermes Science Publications, 1999.

[Bonnet 2001] P. Bonnet, J. Gehrke, P. Seshadri
Towards Sensor Database Systems
2nd International Conference on Mobile Data Management, Hong Kong, January 2001.

[Chanet 2006] J.-P. Chanet, K.-M. Hou, A. Amamra, G. De Sousa
A Non Invasive MANET Bandwidth Estimator for Quality of Service
5th International Information and Telecommunication Technologies Symposium, Cuiaba, Mato Grosso, Brazil, December 6-8, 2006.

[Chanet 2007] J.-P. Chanet
Algorithme de routage coopératif à qualité de service pour des réseaux ad hoc agri-environnementaux
Thèse de doctorat, Université Blaise Pascal, Clermont II, 2007.

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

[Clausen 2003] T. Clausen, P. Jacquet
Optimized Link state Routing Protocol
RFC 3626, October 2003.

[Codd 1970] E. F. Codd
A Relational Model of Data for Large Shared Data Banks
Communications of ACM, vol. 13, no. 6, pp. 377-387, June 1970.

[Dabek 2002] F. Dabek, N. Zeldovich, F. Kaashoek, D. Mazières, R. Morris
Event-driven Programming for Robust Software
10th ACM SIGOPS European Workshop, Saint-Emilion, France, pp. 186-189, September 22-25, 2002.

[Dai 2004] H. Dai, M. Neufeld, R. Han
ELF: An Efficient Log-Structured Flash File System for Micro Sensor Nodes
2nd International Conference on Embedded Networked Sensor System (SenSys'04), Baltimore, Maryland, USA, pp. 176-187, November 3-5, 2004.

[de Vaulx 2003] C. de Vaulx
Etude et développement d'un micro noyau réparti, temps réel et tolérant aux fautes : DREAM
Thèse de doctorat, Université Blaise Pascal, Clermont II, 2003.

[De Sousa 2005] G. De Sousa, J.-P. Chanet, A. Amamra, J. Hao, M.-A. Kang, F. Pinet, K.-M. Hou
Protocole de communication sans fil dédié : Obstacle Location-Aided Routing
9ème Journée Scientifique de l'Ecole Doctorale Sciences Pour l'Ingénieur, Université Blaise Pascal, Clermont-Ferrand, France, 27 Juin 2005.

[De Sousa 2007a] G. De Sousa, H.-Y. Zhou, K.-M. Hou, C. de Vaulx, J.-P. Chanet
LiveFile: A Compact and Interrogative System for Data Collection
International Workshop on Wireless Sensor Networks, Marrakech, Maroc, pp. 79-84, June 4, 2007.

[De Sousa 2007b] G. De Sousa, H.-Y. Zhou, K.-M. Hou, C. de Vaulx, J.-P. Chanet
Adaptive System for Wireless Sensor Networks Applications
Journal of Harbin Institute of Technology (HIT), vol. 39, pp. 154-157, October 2007.

[De Sousa 2008] G. De Sousa, H.-Y. Zhou, K.-M. Hou, C. de Vaulx, J.-P. Chanet
Système pour l'acquisition et la gestion de données environnementales
Atelier thématique « Système d'Information et de Décision pour l'Environnement (SIDE) » associé au XXVI^{ème} Congrès INFORSID, pp. 77-84, 27 Mai 2008.

[Deutsch 1996a] P. Deutsch
GZIP file format specification version 4.3
RFC 1952, May 1996.

[Deutsch 1996b] P. Deutsch
DEFLATE Compressed Data Format Specification version 1.3
RFC 1951, May 1996.

[Diao 2007a] X. Diao, E. Lai, K.-M. Hou, H.-Y. Zhou
An Auto-clustering Algorithm for Wireless Sensor Network Management Protocol
International Workshop on Wireless Sensor Networks, Marrakech, Maroc, pp. 17-22, June 4, 2007.

[Diao 2007b] Y. Diao, D. Ganesan, G. Mathur, P. Shenoy
Rethinking Data Management for Storage-centric Sensor Networks
3rd Biennial Conference on Innovative Data Systems Research, Asilomar, California, USA, pp. 22-31, January 7-10, 2007.

[Duffy 2006] C. Duffy, U. Roedig, J. Herbert, C. J. Sreenan
A Performance Analysis of MANTIS and TinyOS
Technical Report CS-2006-27-11, University of Cork, Ireland, 2006.

[Dulman 2004] S. Dulman, T. Hofmeijer, P. Havinga
AmbientRT – Real Time, Data Centric System Software for Wireless Sensor Networks
21st sensor symposium on Sensors, Micromachines and Applied Systems (SMAS 2004), Kyoto, Japan, pp. 1-6, October 14-15, 2004.

[Dunkels 2004] A. Dunkels, B. Grönwall, T. Voigt
Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors
1st IEEE Workshop on Embedded Networked Sensors (IEEE EmNetS-I), Tampa, Florida, USA, November 2004.

[Dunkels 2006] A. Dunkels, O. Schmidt, T. Voigt, M. Ali
Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems
4th International Conference on Embedded Networked Sensor Systems (SenSys'06), Boulder, Colorado, USA, pp. 29-42, November 2006.

[Friend 2004] R. Friend
Transport Layer Security (TLS) Protocol Compression Using Lempel-Ziv-Stac (LZS)
RFC 3943, November 2004.

[Gal 2005] E. Gal, S. Toledo
A Transactional Flash File System for Microcontrollers
USENIX Annual Technical Conference, Anaheim, California, USA, pp. 89-104, April 10-15, 2005.

[Gelernter 1985] A. D. Gelernter
Generative communication in LINDA
ACM Transactions on Programming Languages and Systems, pp. 80-112, 1985.

[Gupta 2005] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, S. C. Shantz
Sizzle: A standards-based end-to-end security architecture for the embedded Internet
Pervasive and Mobile Computing 1, Elsevier, pp. 425-445, August 2005.

[Gutiérrez 2007] C. Gutiérrez, S. Servigne, R. Laurini
Towards Real-time Metadata for Network-based Geographic Databases
5th International Symposium, Spatial Data Quality 2007, Enschede, the Netherlands, June 13-15, 2007

[Haas 1997] Z. J. Haas
A New Routing Protocol for the Reconfigurable Wireless Networks
6th International Conference on Universal Personal Communications ICUP'97, San Diego, California, USA, pp. 562-566, October 1997.

[Hadim 2006] S. Hadim, N. Mohamed
Middleware Challenges and Approaches for Wireless Sensor Networks
IEEE Distributed Systems Online, IEEE Computer Society, vol. 7, no. 3, March 2006.

[Heinzelman 2002] W. B. Heinzelman, A. P. Chandrakasan, H. Balakrishnan
An Application-Specific Protocol Architecture for Wireless Microsensor Networks
IEEE Transactions on Wireless Communications, vol. 1, no. 4, pp. 660-670, October 2002.

[Hewson 2007] D. J. Hewson, J. Duchêne, F. Charpillet, J. Saboune, V. Michel-Pellegrino, H. Amoud, M. Doussot, J. Paysant, A. Boyer, J.-Y. Hogrel
The PARAChute project: remote monitoring of posture and gait for fall prevention
EURASIP Journal on Advances in Signal Processing, vol. 2007, issue 1, Article ID 27421, 15 pages, February 2007.

[Hill 2000] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister
System Architecture Directions for Networked Sensors
9th International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, Massachusetts, USA, vol. 35, no. 11, pp.93-104, November 2000.

[Hofmeijer 2005] T. J. Hofmeijer, S. O. Dulman, P. G. Jansen, P. J. M. Havinga
AmbientRT – real time system software support for data centric sensor networks
Technical Report TR-CTIT-05-02, Centre for Telematics and Information Technology, University of Twente, Holland, 2005.

[Hollenbeck 2004] S. Hollenbeck
Transport Layer Security Protocol Compression Methods
RFC 3749, May 2004.

[Hou 2007a] K.-M. Hou, J.-J. Li, H.-Y. Zhou, J.-P. Chanet, J. Hao, G. De Sousa, C. de Vaulx, A. Amamra, M. Kara, D. Bendali-Amor
Cooperative Inter-vehicle communication protocol dedicated to intelligent transport systems
Journal of Harbin Institute of Technology (HIT), vol. 39, pp. 145-152, October 2007.

[Hou 2007b] K.-M. Hou, G. De Sousa, J.P. Chanet, H.-Y. Zhou, M. Kara, A. Amamra, X. Diao, C. de Vaulx, J.-J. Li, A. Jacquot
LiveNode: LIMOS versatile embedded wireless sensor node
Journal of Harbin Institute of Technology (HIT), vol. 39, pp. 140-144, October 2007.

[Huffman 1952] D. A. Huffman
A Method for the Construction of Minimum-Redundancy Codes
Institute of Radio Engineers (IRE), vol. 40, no. 9, pp. 1098-1101, September 1952.

[Huynh 2000] T. N. Huynh, O. Mangisengi, A. M. Tjoa
Metadata for Object-Relational Data Warehouse
2nd International Workshop on Design and Management of Data Warehouses (DMDW'00), Stockholm, Sweden, June 5-6, 2000.

[Jacquot 2007] A. Jacquot, J.-P. Chanet, K.-M. Hou, H.-Y. Zhou
Un objet communicant intelligent pour des réseaux agri-environnementaux : LiveNode
5ème édition des journées STIC & Environnement, Lyon, 13-15 Novembre 2007.

[Jain 2003] M. Jain, C. Dovrolis
End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput
IEEE Transactions on Networking, vol. 11, no. 4, pp. 537-549, August 2003.

[Jansen 2003] P. G. Jansen, S. J. Mullender, P. J. M. Havinga et H. Scholten
Lightweight EDF Scheduling with Deadline Inheritance
Research report, University of Twente, Holland, May 2003.

[Johnson 2007] D. Johnson, Y. Hu, D. Maltz
The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4
RFC 4728, February 2007.

[Kara 2007] M. Kara, K.-M. Hou, J.-P. Chanet, H.-Y. Zhou, M.-A. Kang, F. Pinet
Low Cost Differential GPS receivers (LCD-GPS): A Local Cooperative Differential GPS solution
International Workshop on Wireless Sensor Networks, Marrakech, Maroc, pp. 29-34, June 4, 2007.

[Karl 2003] H. Karl, A. Willig
A short survey of wireless sensor networks
Technical Report TKN-03-018, Telecommunication Networks Group, Technische Universität Berlin, October 2003.

[Ko 2000] Y.-B. Ko, N. H. Vaidya
Location-Aided Routing (LAR) in mobile ad hoc networks
Wireless Networks, vol. 6, no. 4, pp. 307-321, July 2000.

[Lalooses 2007] F. Lalooses, H. Susanto, C. H. Chang
An Approach for Tracking Wildlife using Wireless Sensor Networks
International Workshop on Wireless Sensor Networks, Marrakech, Maroc, pp. 71-77, June 4, 2007.

[Lauer 1978] H. C. Lauer, R. M. Needham
On the duality of operating systems structures
2nd International Symposium on Operating Systems, October 1978.

[Le Borgne 2007] Y. Le Borgne, M. Van der Haegen, G. Bontempi
Localization for wireless sensor networks with CC2420 radios
International Workshop on Wireless Sensor Networks, Marrakech, Maroc, pp. 23-28, June 4, 2007.

[Levis 2004] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A Woo, E. Brewer, D. Culler
The Emergence of Networking Abstractions and Techniques in TinyOS
1st USENIX/ACM Symposium on Networked Systems Design and Implementation, (NSDI 2004), San Francisco, California, USA, pp. 1-14, March 29-31, 2004.

[Liu 1973] C. L. Liu, J. W. Layland
Scheduling algorithms for multiprogramming in a hard real-time environment
Journal of ACM, vol. 20, no. 1, pp. 46-61. 1973.

[Madden 2005] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong
TinyDB: an Acquisitional Query Processing System for Sensor Networks
ACM Transactions on Database Systems (TODS), vol. 30, no. 1, pp. 122-173, March 2005.

[Mainwaring 2002] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, J. Anderson
Wireless sensor network for habitat monitoring
1st ACM International Workshop on Wireless Sensor Networks and Applications, Atlanta, Georgia, USA, pp. 88-97, Septemeber 2002.

[Mathur 2006a] G. Mathur, P. Desnoyers, D. Ganesan, P. Shenoy
Ultra-Low Power Data Storage for Sensor Networks
5th International Conference on Information Processing in Sensor Networks, Nashville, Tennessee, USA, pp. 374-381, April 19-21, 2006.

[Mathur 2006b] G. Mathur, P. Desnoyers, D. Ganesan, P. Shenoy
Capsule: An Energy-Optimized Object Storage System for Memory-Constrained Sensor Devices
4th ACM Conference on Embedded Networked Sensor Systems (SenSys'06), Boulder, Colorado, USA, pp. 195-208, November 1-3, 2006.

[MSP 2008] Mixed-Signal Processors, Texas Instrument
www.ti.com/sc/docs/msp/tools/macromod.htm, 2008.

[Nekoogar 2004] F. Nekoogar, F. Dowla, A. Spiridon
Self Organization of Wireless Sensor Networks Using Ultra-Wideband Radios
IEEE Radio and Wireless Conference, Atlanta, Georgia, USA, pp. 451-454, September 19-22, 2004.

[Noël 2006] G. Noël
Indexation dans les bases de données capteurs temps reel
Thèse de doctorat, Institut National des Sciences Appliquées de Lyon, 2006.

[NS 2008] The Network Simulator ns-2
The ns Manual (formerly ns Notes and Documentation)
The VINT project, www.isi.edu/nsnam/ns/, August 12, 2008.

[Park 1997] Y.-H. Park
Etude en vue de la réalisation d'un noyau temps-réel multiprocesseur et l'environnement de développement intégré
Thèse de doctorat, Université de Technologie de Compiègne, 1997.

[Perkins 1994] C. E. Perkins, P. Bhagwat
Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers
ACM SIGCOMM Symposium on Communication, Architectures and Protocols, vol. 24, no. 4, pp. 234-244, London, UK, August 1994.

[Perkins 2003] C. E. Perkins, E. M. Belding-Royer, S. Das
Ad hoc On-Demand Distance Vector (AODV) Routing
RFC 3561, July 2003.

[Plesse 2005] T. Plesse, C. Adjih, P. Minet, A. Laouiti, A. Plakoo, M. Badel, P. Muhlethaler, P. Jacquet, J. Lecomte
OLSR Performance Measurement in a Military Mobile Ad-hoc Network
Special issue on data communication and topology control in ad-hoc networks, Elsevier, vol. 3, issue 5, pp. 575-588, September 2005.

[Pujolle 2007] G. Pujolle
Les Réseaux
Eyrolles, Août 2007.

[Puthenpurayil 2007] S. Puthenpurayil, R. Gu, S. S. Battacharyya
Energy-aware Data Compression for Wireless Sensor Networks
International Conference on Acoustics, Speech, and Signal Processing, vol. 2, Honolulu, Hawaii, USA, pp. 42-45, April 15-20, 2007.

[Ramasubramanian 2003] V. Ramasubramanian, Z. J. Hass, E. G. Sirer
SHARP: a Hybrid Adaptive Routing Protocol for Mobile Ad Hoc Networks
4th ACM International Symposium on Mobile Ad Hoc Networking & Computing, Annapolis, Maryland, USA, pp. 303-314, 2003.

[Römer 2002] K. Römer, O. Kasten, F. Mattern
Middleware Challenges for Wireless Sensor Networks
ACM SIGMOBILE Mobile Computing and Communications Review, vol. 6, issue 4, pp. 59-61, October 2002.

[Rosenblum 1991] M. Rosenblum, J. K. Ousterhout
The Design and Implementation of a Log-Structured File System
13th ACM Symposium on Operating Systems Principles, Asilomar, California, USA, pp. 1-15, October 13-16, 1991.

[Rowstron 1996] A. I. T. Rowstron
Bulk primitives in LINDA run-time systems
Thèse de doctorat, Université de York, Royaume-Uni, 1996.

[Sadler 2006] C. M. Sadler, M. Martonosi
Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks
4th ACM Conference on Embedded Networked Sensor Systems (SenSys), Boulder, Colorado, USA, pp. 265-278, October 31 – November 3, 2006.

[Sha 1990] L. Sha, R. Rajkumar, J. P. Lehoczky
Priority inheritance protocols: An approach to real-time synchronization
IEEE Transactions on Computers, vol. 39, no. 9, pp. 1175-1185, September 1990.

[Shannon 1948] C. E. Shannon
A Mathemarical Theory of Communication
The Bell System Technical Journal, vol. 27, pp. 379-423, 623-656, October 1948.

[Soulignac 2006] V. Soulignac, F. Barnabé, D. Rat, F. David
SIGEMO : un système d'information pour la gestion des épandages de matières organiques. Du cahier des charges à l'outil opérationnel
Ingénieries – EAT, vol. 47, pp. 37-42, 2006.

[Stojmenovic 2002] I. Stojmenovic
Position-based routing in ad hoc networks
IEEE Communications Magazine; vol.40, no. 7, pp. 128-134, July 2002.

[Stojmenovic 2005] I. Stojmenovic
Handbook of Sensor Networks: Algorithms and Architectures
Wiley Series on Parallel and Distributed computing, November 2005.

[Tanenbaum 2003] A. S. Tanenbaum
Réseaux
Pearson Education, Mai 2003.

[Tanenbaum 2006] A. S. Tanenbaum, A. S. Woodhull
Operating Systems Design and Implementation
Prentice Hall, Third Edition, January 2006.

[Toshiba 2006] Toshiba
NAND vs. NOR Flash Memory Technology Overview
Technical Report, 2006.

[von Behren 2003] R. von Behren, J. Condit, E. Brewer
Why Events Are A Bad Idea (for high-concurrency servers)
9th Workshop on Hot Topics in Operating Systems (HotOS IX), Lihue (Kauai), Hawaii, USA, pp. 19-24, May 18-21, 2003.

[Wanner 2005] L. F. Wanner, A. S. H. Junior, F. V. Polpeta, A. A. Fröhlich
Operating System Support for Handling Heterogeneity in Wireless Sensor Networks
10th IEEE Conference on Emerging Technologies and Factory Automation EFTA, vol. 2, September 19-22, 2005.

[Welch 1984] T. A. Welch
A Technique for High-Performance Data Compression
IEEE Computer Society, vol. 17, no. 6, June 1984.

[XBee 2007] XBee, IEEE 802.15.4 OEM RF Modules, MaxStream
XBee™/ XBee-Pro™ OEM RF Modules
Product manual, 2007.

[Yang 2007] C.-C. Yang, L.-P. Tseng
Fisheye zone routing protocol: a multi-level zone routing protocol for mobile ad hoc networks
Computer Communications, vol. 30, no. 2, pp. 261-268, January 2007.

[Yao 1996] E. Yao
H-LINDA : Un noyau temps réel multiprocesseur
Thèse de doctorat, Université de Technologie de Compiègne, 1996.

[Zeinalipour-Yazti 2005] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, W. A. Najjar
MicroHash : An Efficient Index Structure for Flash-Based Sensor Devices
4th USENIX Conference on File And Storage Technologies (FAST'05), San Francisco, California, USA, pp. 31-44, December 13-16, 2005.

[Zhao 2004] Y. Zhao, R. Shi, X. Yang
One Hop-DAD Based Address Autoconfiguration in MANET6
5th International Conference on Parallel and Distributed Computing, Applications and Technologies PDCAT, Singapore, pp. 674-680, December 8-10, 2004.

[Zhou 2004a] H.-Y. Zhou, K.-M. Hou, J. Ponsonnaille, L. Gineste, J. Coudon, G. De Sousa, C. de Vaulx, J.-J. Li, P. Chainais, R. Aufrère, A. Amamra, J.-P. Chanet
Remote Continuous Cardiac Arrhythmias Detection and Monitoring
Transformation of Healthcare with Information Technologies, Studies in Health Technology and Informatics, vol. 105, IOS Press, pp. 112-120, 2004.

[Zhou 2004b] H.-Y. Zhou
Réseau de capteurs sans fil dédié à la détection et au diagnostic d'arythmie cardiaque en temps réel, en continu et à distance
Thèse de doctorat, Université Blaise Pascal, Clermont II, 2004.

[Zhou 2006a] H.-Y. Zhou, G. De Sousa, J.-P. Chanet, K.-M. Hou, J.-J. Li, C. de Vaulx, M. Kara
An Intelligent Wireless Bus-Station System Dedicated to Disabled, Wheelchair and Blind Passengers
IET International Conference on Wireless, Mobile & Multimedia Networks, ICWMMN 2006, Hangzhou, China, November 6-9, 2006.

[Zhou 2006b] H.-Y. Zhou, K.-M. Hou, J.-P. Chanet, C. de Vaulx, G. De Sousa
LIMOS: a Tiny Real-Time Micro-Kernel for Wireless Objects
2nd IEEE International Conference on Wireless Communications, Networking and Mobile Computing WiCOM (WCNM) 2006, Wuhan, China, September 22-24, 2006.

[Zhou 2006c] H-Y. Zhou, K.-M. Hou, J.-P. Chanet, C. de Vaulx et G. De Sousa
A Novel Hybrid Operating System Dedicated to Wireless Sensor Network
IET International Conference on Wireless, Mobile & Multimedia Networks
ICWMMN 2006, Hangzhou, China, November 6-9, 2006.

[Zhou 2006d] H.-Y. Zhou
Implementation of CIVIC protocol: E-CIVIC
Rapport de post-doctorat, Cemagref, UR TSCF, UMR TETIS, Clermont-Ferrand, 2006.