Ingénierie dirigée par les modèles pour la conception et la mise en œuvre des réseaux de capteurs

Abdenour Kifouche

UNIVERSITÉ PARIS-EST

École doctorale : Mathématiques, Sciences et Technologies de l'Information et de la Communication

THÈSE DE DOCTORAT

pour obtenir le grade de docteur délivré par

Université Paris-Est

Spécialité doctorale "Électronique, Optronique et Système"

présentée et soutenue publiquement par

Abdenour KIFOUCHE

le 11 Septembre 2019

Ingénierie dirigée par les modèles pour la conception et la mise en œuvre des réseaux de capteurs

Directeur de thèse : **Geneviève BAUDOIN**
Co-encadrants de thèse : **Rédha HAMOUCHE, Rémy KOCIK**

Jury

Mme. Geneviève Baudoin,	Professeure	Directeur de thèse
M. Rédha Hamouche,	Maître de conférences	Co-Encadrant
M. Rémy Kocik,	Maître de conférences	Co-Encadrant
Mme. Atika Rivenq-Menhaj,	Professeure	Rapporteur
M. Hassine Moungla,	Maître de conférences	Rapporteur
M. Abderrezak Rachedi,	Professeur	Examinateur
M. Pascal Poizat,	Professeur	Examinateur
M. Bachir Djafri,	Maître de conférences	Examinateur

THESE

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes

6LoWPAN IPv6 over Low power Wireless Personal Area Network. 34, 35

ADC Analog-to-Digital Converter. 86, 87, 115

API Application Programming Interface. 111

ARIB Association of Radio Industries and Businesses. 33

ATL ATLAS Transformation Language. 53, 119

BIM Building Information Modeling. 133

BLE Bluetooth Low Energy. 108

CIM Computation Independant Model. 50

COV Composé Organique Volatil. 108

CSMA Carrier Sensing Multiple Access. 33

DAC Digital Analog Converter. 87

DSML Domain Specific Modelling Language. 48, 51

EMF Eclipse Modeling Framework. 51, 52

EMP Eclipse Modeling Project. 51

ETSI European Telecommunications Standards Institute. 33

FCC Federal Communications Commission. 33

GME Generic Modeling Environment. 51

GMF Graphical Modeling Framework. 52, 114, 132

GNSS Global Navigation Satellite System. 27

IBM International Business Machines. 51

IDM Ingénierie Dirigée par les Modèles. 5, 9, 19, 45–47, 49–51, 55–57, 59, 60, 68, 82, 100

IEEE Institute of Electrical and Electronics Engineers. 36

IoT Internet of Things. 83, 111, 126

IP Internet Protocol. 35

IPv4 Internet Protocol version 4. 35

IPv6 Internet Protocol version 6. 35

ISM Bande radio Industrielle, Scientifique et Médicale. 33, 37

LoRa Long Range. 105, 108, 114, 121–124, 130

LoRaWAN Long Range Wide Area Network. 37

LOS Line-Of-Sight. 38

LPWAN Low Power Wide Area Network. 36, 37

LR-WAN Low Rate Wide Area Network. 35

M2M Model-to-Model. 50, 71

M2T Model-to-Text. 50, 71

MAC Medium Access Control. 31, 33–36, 46, 72, 82, 99, 124

MDA Model-Driven Architecture. 50, 51

MODESENE MOdel Driven Environment for SEnsor NEtwork. 19, 105, 113, 114, 119, 121, 124, 129, 132

OCL Object Constraint Language. 69

OMG Object Management Group. 47, 49–51

OSI Open Standards Interconnection. 31, 35

PIM Plateform Indépendant Model. 50, 51, 56

PSM Platform Specific Model. 50, 51, 56

QVT Query View Transformation. 51

RdC Réseau de Capteurs. 17–19, 35, 38, 42, 46, 47, 59, 60, 63, 66, 68, 69, 71, 73, 78, 79, 82–84, 86, 89, 92, 102, 105, 108, 111, 114, 118, 120, 126, 131–133

RF Radio frequency. 87

RSA IBM Rational Software Architect. 51

SQL Structured Query Language. 83

TCP Transmission Control Protocol. 35

TDMA Time Division Multiple Access. 33

TOSSIM TinyOS SIMulator. 47

UDP User Datagram Protocol. 35

UIT Union Internationale des Télécommunications. 94

UML Unified Modeling Language. 51, 56, 58, 69

WSN Wireless Sensor Network. 28

XMI XML Metadata Interchange. 51, 69

XML eXtensible Markup Language. 51, 119

ZigBee Zig-zag Bees. 6, 105, 108, 114, 115, 120–124, 130

Remerciements

Cette thèse a été effectuée au sein du laboratoire ESYCOM de l'Université Paris-Est, à l'ESIEE-Paris, sous la direction de Geneviève Baudoin et co-encadrée par Rédha Hamouche et Rémy Kocik.

Tout d'abord, je tiens à remercier sincèrement Geneviève Baudoin, Rédha Hamouche et Rémy Kocik pour leurs soutiens, leur patience et leur confiance. Pendant toutes ces années, ils m'ont aidé à améliorer mes qualités de chercheur. J'apprécie de tout cœur le temps qu'ils ont investi pour faire de cette thèse un succès.

Je voudrais exprimer ma gratitude à Mme Atika Rivenq et à M. Hassine Moungla pour l'intérêt qu'ils portent à mon travail et l'honneur qu'ils m'ont fait en acceptant d'en être les rapporteurs.

Je tiens à remercier les membres du jury, M. Abderrezak Rachedi, M. Pascal Poizat et M. Bachir Djafri, d'avoir fait l'effort de lire et de réviser cette thèse, d'avoir fait le déplacement pour assister à la soutenance et de m'avoir donné un retour utile.

Je ne les remercierai jamais assez tous les quatre : Nadine, Rémy, Abdel et Rédha, pour la gentillesse infinie, le sens de l'humour dont ils ont fait preuve tout au long de ces annèes de thèse.

Je profite de cette opportunité pour remercier mes amis et collègues, y compris Derej, Alex, Varun, Alain, Adel, Imad, Miyassa, Kimon, Pablo, Pape, Houssein, Chouaib, siqi, Chérif, Zerihun pour l'amitié et le professionnalisme de toutes ces années. En particulier, j'exprime mes remerciements spéciaux à mes chers amis Sofian C, Sofian N, Abdessalam, Yacine, Riadh, Bilal, Oualid, Ilyas, Walid, Kamel, Yaften pour les plus beaux moments que nous avons partagé. L'amitié est importante, elle rend la vie encore plus belle.

Surtout, je tiens à remercier mes parents pour leur dévouement et leur sacrifice et pour m'avoir donné cette chance. J'aimerais aussi remercier ma famille pour leur amour et leur soutien inconditionnels.

Introduction générale

« *Learning is the only thing the mind never exhausts, never fears, and never regrets.* »

Leonardo da Vinci

Motivations

La conception et la mise en œuvre d'un Réseau de Capteurs (RdC) implique différentes étapes, notamment le dimensionnement du réseau, le développement de logiciels embarqués, la réalisation de matériels, des analyses et des simulations, le déploiement physique, l'exploitation des données et la maintenance du réseau. L'ensemble de ces étapes constitue le cycle de vie d'un RdC. la mise en oeuvre et l'exploitation des réseaux de capteurs nécessitent donc la maîtrise de différents domaines tels que l'électronique, l'informatique, les radiocommunications et les réseaux. Elle implique aussi différents acteurs de divers métiers. Par exemple, l'exploitation des données peut être réalisée par des acteurs de domaines non techniques, comme des médecins dans le domaine de la santé.

Tout au long du cycle de vie d'un RdC, plusieurs outils/applications de domaines spécifiques sont utilisés (conception du matériel, codage du logiciel, spécification et simulation des réseaux et protocoles de communication, simulation de l'environnement et évaluation de son impact sur le RdC). Ces outils interviennent à différents niveaux d'abstraction et utilisent des langages et des sémantiques différentes. Ils sont généralement indépendants et rarement connectés les uns aux autres.

Dans cette démarche, il est nécessaire de décrire le RdC manuellement dans chacun de ces outils en utilisant les langages qui leur sont propres. Lorsque le nombre de nœuds du RdC augmente, le temps passé pour faire ces descriptions devient important. Une modification du réseau doit le plus souvent être répercutée dans chaque outil. Il est alors difficile d'assurer la cohérence entre les différentes descriptions. Ce cloisonnement conduit bien souvent à des erreurs de conception, des mauvaises interprétations et éventuellement à des dysfonctionnements.

Ces difficultés mettent en évidence le besoin d'une approche plus centralisée des différents aspects du RdC. Cette approche vise à faciliter et fluidifier la collaboration entre

les acteurs impliqués à chaque étape du cycle de vie, éviter les erreurs de conception et réduire les coûts de développement.

Objectifs

L'objectif principal de cette thèse est de proposer un environnement de conception et de mise en œuvre de réseaux de capteurs. Plus précisément, il s'agit de :

— Définir une méthodologie pour la conception et la mise en œuvre des RdC couvrant l'ensemble des étapes du cycle de vie ;

— Développer des modèles génériques permettant de décrire différentes architectures matérielles, logicielles et réseaux. Ces modèles devront supporter l'hétérogénéité présente dans les réseaux de capteurs ;

— Offrir un environnement permettant l'analyse/simulation du RdC tout au long du cycle de développement. Ceci permettra d'estimer le plus tôt possible les performances d'un RdC ;

— Accélérer l'étape du développement logiciel en proposant de générer automatiquement le code source embarqué sur chaque nœud du RdC ;

— Intégrer dans un même environnement les phases d'exploitation et de maintenance d'un RdC pour faciliter l'exploitation de données.

Contributions

Pour atteindre ces objectifs, nous proposons :

✓ Une méthodologie basée sur l'ingénierie dirigée par les modèles, IDM ou MDE (Model-Driven Engineering) [4]. Elle vise à couvrir toutes les phases du cycle de vie d'un RdC en formalisant à l'aide d'un métamodèle toutes les caractéristiques de celui-ci ;

✓ Une approche de conception multi-facettes permettant de faciliter la conception des RdC. Cette dernière inclut des spécifications de la structure du réseau, l'architecture logicielle et matérielle des nœuds, le flux de données et l'environnement physique de déploiement ;

✓ Une analyse/simulation des performances d'un RdC. Les analyses peuvent être internes ou via des outils externes moyennant des passerelles vers des simulateurs. Ceci permet d'avoir une meilleure optimisation dans les choix de conception (architecture, topologie du réseau, technologie des communications, etc.) avant le déploiement physique du RdC ;

✓ Différentes transformations de modèles qui seront utilisées par exemple pour obtenir automatiquement du code source embarqué sur chaque nœud à partir d'une

description de haut niveau des architectures matérielles et logicielles. Cette fonction accélère le processus de développement logiciel et permet d'éviter les erreurs de programmation ;

✓ Un Framework d'aide à la conception mettant en œuvre notre méthodologie. Ce Framework, nommé MODESENE, offre un environnement de description de modèles basé sur plusieurs facettes (réseau, matérielle, logicielle, flux de données et environnement physique). Une passerelle entre ce Framework et le simulateur Omnet++ [5] a été développée pour estimer les performances du RdC. A partir des modèles décrits, le Framework assure la génération automatique des codes sources pour les nœuds capteurs. Une deuxième passerelle entre le Framework et l'outil Node-RED [6] a été également développée. Elle permet de fournir automatiquement des interfaces graphiques pour la phase d'exploitation de données et de maintenance ;

✓ Un cas d'étude pour la mesure climatique dans le bâtiment a été défini. Ce système, déployé dans notre école ESIEE-Paris, constitue un RdC d'expérimentation pour valider la méthodologie et le Framework proposés.

Structure de la thèse

Ce manuscrit est organisé en cinq chapitres. Le premier et le deuxième chapitres traitent respectivement des généralités sur les réseaux de capteurs et les approches d'ingénierie dirigées par les modèles. Les chapitres 3, 4 et 5 présentent nos contributions. Dans le troisième chapitre, une vue d'ensemble de notre méthodologie est présentée en restant à un niveau de détail relativement élevé pour des raisons de clarté. Les modèles développés pour cette méthodologie sont détaillés dans le quatrième chapitre. Une expérimentation à travers un cas d'étude est mise en évidence dans le dernier chapitre.

Le contenu de chaque chapitre est brièvement présenté comme suit :

- **Chapitre 1 :** dans ce chapitre, nous introduisons le contexte général de notre travail en définissant les différents concepts liés aux réseaux de capteurs et en présentant leurs applications. Nous décrivons ensuite la structure d'un RdC et nous donnons ses propriétés. Enfin, nous identifions les défis liés à la conception des réseaux de capteurs ;
- **Chapitre 2 :** dans ce chapitre, nous présentons l'état de l'art concernant les approches de modélisation des réseaux de capteurs. Nous nous focalisons sur les approches d'ingénierie dirigées par les modèles (IDM). Nous proposons des critères de comparaison pour réaliser une étude des approches existantes pour la conception des réseaux de capteurs à base de l'IDM. Nous synthétisons ensuite leurs avantages et leurs limites ;

- **Chapitre 3 :** dans ce chapitre, nous présentons un aperçu global de notre méthodologie de conception de réseaux de capteurs. Nous commençons par décrire ses objectifs, puis nous identifions les facettes permettant de modéliser les différents aspects d'un RdC. Ensuite, nous décrivons les transformations reliant les facettes entre elles ;
- **Chapitre 4 :** dans ce chapitre, nous présentons les modèles de notre méthodologie. Nous commençons par décrire les modèles de chaque facette. Nous soulignons aussi les analyses et les simulations à base de ces modèles. Nous présentons ensuite les modèles de la génération automatique du code source ;
- **Chapitre 5 :** ce dernier chapitre présente la mise en œuvre et l'expérimentation de notre méthodologie dans le Framework proposé. Dans un premier temps, nous décrivons l'étude de cas portant sur la mesure climatique dans le bâtiment intelligent. Ensuite, nous détaillons la conception multi-facette de ce système à travers ce Framework . Enfin, nous présentons l'utilisation des fonctionnalités du Framework pour le développement du cas d'étude.

La conclusion générale résume les problématiques ciblées, les contributions obtenues et les perspectives de cette thèse.

Liste des publications

Cette thèse a donné lieu à des publications dans des revues et conférences internationales et nationales :

Publications en revues internationales :

— Kifouche, A., Hamouche, R., Kocik, R., Rachedi, A. and Baudoin, G., "Model driven Framework to enhance sensor network design cycle". Transactions on Emerging Telecommunications Technologies, p.e3560, 2019.

— Kifouche, A., Hamouche, R., Kocik, R. and Baudoin, G. (2019) 'Sensor network design for smart building', Int. J. Digital Enterprise Technology, Vol. 1, No. 4, pp.368–387.

Communications en conférences internationales :

— Kifouche, A., Baudoin, G., Hamouche, R. and Kocik, R., "Generic sensor network for building monitoring : design, issues, and methodology". In 2017 IEEE Conference on Wireless Sensors (ICWiSe), pp. 1-6, Miri, Malaisie. November 2017.

— Kifouche, A., Hamouche, R., Kocik, R., Rachedi, A. and Baudoin, G., 2019, April. Model-Driven Framework to Speed up Design and Exploitation of Sensor Networks. In IEEE Wireless Communications and Networking Conference (WCNC), Marrakech, Maroc, Avril 2019.

— Kifouche, A., Hamouche, R., Kocik, R., Rachedi, A. and Baudoin, G., 2019, Methodology for the design and implementation of sensor networks : application to smart buildings, In International Workshop on Information Technology and Communication (IWITC'19) IoT and Smart City, Villeneuve d'Ascq, France, July 2019.

Communication en conférence nationale :

— Kifouche, A., Baudoin, G., Hamouche, R. and Kocik, R., Model-Driven Framework for Design and Exploitation of Sensor Networks. In Journée Systèmes Embarqués et Objets Communicants (SEOC2019). CNAM Paris France, 2019.

Chapitre 1

Généralités sur les réseaux de capteurs

« *You can engineer systems to ensure that equipment is operating correctly, but the problem is that it is difficult to keep the sensor clean in such a dusty environment.* »

Nima Nabavi

Sommaire

1.1 Introduction

Selon les résultats de récentes études [7, 8], il y aura 50 milliards de dispositifs connectés à Internet en 2020 (ce qui équivaut à sept fois la population mondiale). Ces dispositifs ne sont pas seulement des smartphones et des tablettes, mais aussi des appareils qui peuvent effectuer diverses fonctions, comme des fonctions d'acquisitions de données (capteurs) ainsi que des opérations sur un environnement (actionneurs). Dans ce contexte, nous retrouvons les concepts des réseaux de capteurs.

Le présent chapitre a pour objectif de présenter les notions de base des réseaux de capteurs. Pour ce faire, nous définissons les réseaux de capteurs et décrivons leurs applications dans la section 1.2. Nous donnons ensuite dans la section 1.3 un aperçu sur la structure d'un réseau de capteurs. Dans la section 1.4, nous décrivons les propriétés des réseaux de capteurs. Nous examinons dans la section 1.5 les défis de la conception des réseaux de capteurs. Enfin, nous résumons les principales idées de ce chapitre dans une conclusion.

1.2 Réseaux de capteurs : définitions et applications

Les réseaux de capteurs sont constitués d'un ensemble de nœuds connectés. Ces nœuds sont en mesure de collecter des données à partir de capteurs distribués et de transmettre des informations utiles à un utilisateur final ou à une application externe [9].

Le réseau de capteurs peut être utilisé dans de nombreux domaines d'applications dont les domaines militaire, environnemental, sanitaire, domestique ou commercial [10]. Plusieurs exemples sont déjà développés [11–13]. Selon l'objectif du déploiement [14, 15], on peut distinguer deux catégories d'applications : les applications de surveillance, et les applications de localisation et suivi (Figure 1.1) [16–18]. Ces deux catégories d'applications sont présentées ci-après.

1.2.1 Applications de surveillance

Les applications de surveillance comprennent la surveillance de l'environnement intérieur et extérieur, la surveillance de la santé, la surveillance sismique et structurale, etc. Elles sont caractérisées par des collectes de données périodiques, continues, événementielles, ou à la demande.

— Les applications de collectes périodiques recueillent des données sur un phénomène (s) à intervalles réguliers, par exemple, les applications qui recueillent la température dans des bâtiments ;

— Les nœuds capteurs peuvent recueillir des données en continu, par exemple, les applications de vidéosurveillance permettent de surveiller une zone en continu ;

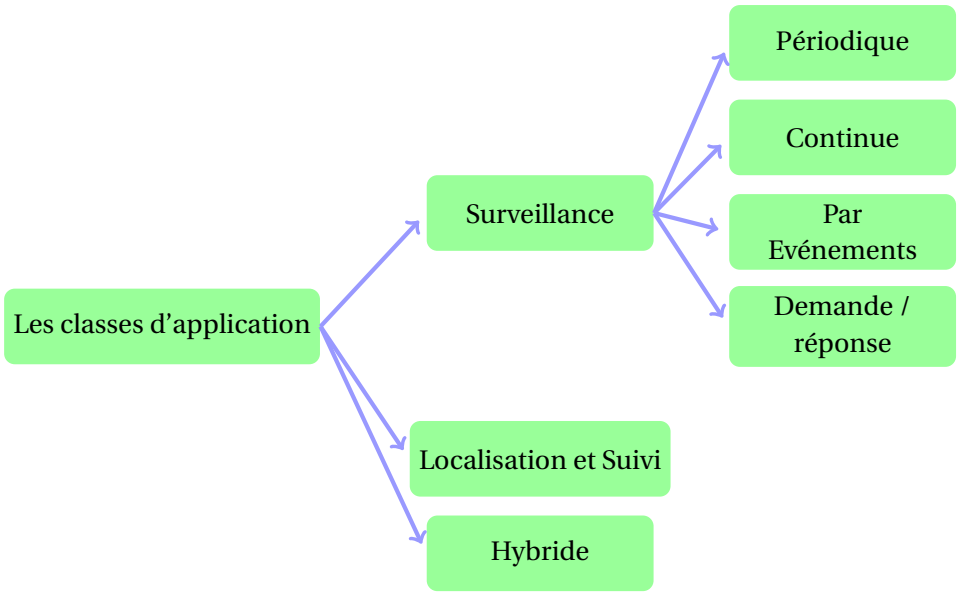


FIGURE 1.1 – Types d'application dans les réseaux de capteurs.

— Certains réseaux peuvent transmettre des données uniquement lorsque des événements d'intérêt se produisent, par exemple, les applications qui détectent la présence d'objets dans une zone surveillée ;

— Un autre type d'application de collecte des données uniquement sur demande [19], par exemple, les applications qui ont une contrainte énergétique élevée. Dans ce cas les communications sont initiées par l'utilisateur pour la collecte de données.

Parmi les applications réelles développées, on peut citer la surveillance volcanique. Un réseau composé de 16 nœuds capteurs a été déployé sur le volcan Reventador dans le nord de l'Équateur [20]. Chaque capteur est un dispositif T-mote sky [21] équipé d'une antenne omnidirectionnelle externe, d'un sismomètre et d'un microphone. Chaque nœud capteur est alimenté par piles. Les nœuds capteurs sont placés à environ 300 mètres les uns des autres. Les nœuds relaient les données via un routage à sauts multiples vers un nœud passerelle. Le nœud passerelle est relié à un module radio longue distance qui transmet les données collectées à la station de base. Pendant le fonctionnement en réseau, chaque nœud capteur échantillonne des données sismo-acoustiques à 100 Hz. Les données sont stockées dans la mémoire locale. Lorsqu'un événement intéressant se produit, le nœud envoie un message à la station de base. Lorsque la collecte des données est terminée, les nœuds reprennent l'échantillonnage et le stockage local des données des capteurs.

1.2.2 Applications de localisation et suivi

Les applications de localisation et de suivi comprennent le suivi d'objets tels que les animaux, les humains et les véhicules. De nos jours, un grand nombre de techniques et

de technologies ont été développées pour les systèmes de localisation. Ces systèmes permettent de répondre de manière plus spécifique à différents besoins par exemple dans les domaines de la sécurité et de la télésurveillance.

La localisation et le suivi des capteurs peuvent être effectués en s'appuyant sur différentes technologies : systèmes globaux Global Navigation Satellite System (GNSS) de navigation par satellites (en outdoor), systèmes de localisation à infrastructures terrestres (utilisant différentes technologies radio), centrales inertielles. La localisation et le suivi peuvent dans certaines applications utiliser une approche collaborative entre nœuds à localiser.

Parmi les applications réelles de localisation, on peut citer ZebraNet [22]. Le système ZebraNet est un réseau de capteurs sans fil mobile utilisé pour suivre les migrations des animaux en particulier les zèbres. ZebraNet est composé de nœuds de capteurs insérés dans le collier du zèbre. Le nœud se compose d'un micro-contrôleur, d'une mémoire, d'un module RF et d'une unité GPS. Les relevés de position sont effectués à l'aide du GPS et envoyés à la station de base. Le but est d'enregistrer avec précision les positions de chaque zèbre et de les utiliser pour l'analyse.

(a) Bâtiment intelligent (b) Ville intelligente (c) Réseau électrique intelligent

(d) Santé intelligente (e) Maison intelligente (f) Production Intelligente

FIGURE 1.2 – Applications typiques des réseaux de capteurs.

Il faut noter qu'il existe aussi des applications hybrides qui regroupent les deux catégories de classes pour réaliser un système complet, mais ces applications nécessitent plus de ressources de traitement. La Figure 1.2 représente quelques exemples de domaines d'applications potentiels des réseaux de capteurs et qui ont un grand potentiel aussi bien aujourd'hui que dans le futur.

1.3 Structure d'un réseau de capteurs

Un réseau de capteurs est composé de plusieurs nœuds qui utilisent des médias de communication pour véhiculer l'information vers des points de collecte [10, 23].

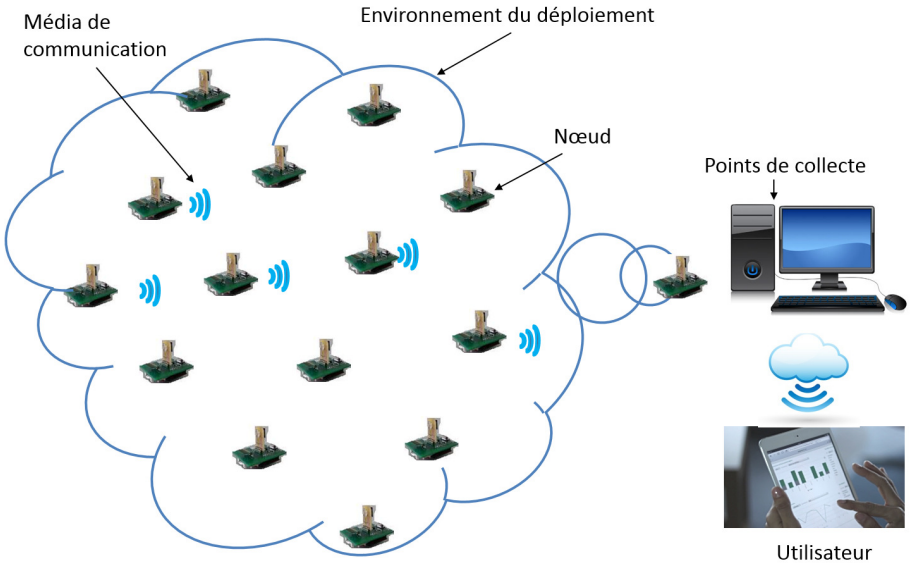


FIGURE 1.3 – Architecture de réseau de capteurs.

La Figure 1.3 illustre l'architecture d'un réseau de capteurs. Les différents éléments constituant cette architecture sont détaillés ci-après :

— **Environnement du déploiement** : il définit la zone dans laquelle les nœuds seront déployés, ce qui peut se traduire par l'environnement physique du déploiement.

— **Nœud** : les nœuds ont pour fonction principale de recueillir les données et de les acheminer vers les stations de traitement. Les nœuds sont souvent alimentés par batterie, ce qui nécessite d'optimiser la consommation d'énergie afin d'augmenter la durée de vie.

— **Médias de communication** : les nœuds peuvent utiliser différents médias de communication pour transmettre et recevoir des données dont des liaisons filaires et des liaisons radio. Les communications sans fil sont plus en plus répandues dans les réseaux de capteurs modernes. Dans les réseaux de capteurs sans fil (*Wireless Sensor Network (WSN)*), les nœuds utilisent des technologies de communication différentes comme ZigBee, LoRa, Sigfox. Chaque technologie a des caractéristiques spécifiques en termes de débit, de portée, de consommation énergétique, etc.

— **Point de collecte** : les nœuds communiquent des données vers des points de collecte, appelés également centre de traitement ou station de base. Les points de collecte ont, en général, plus de ressources de calcul que les nœuds capteurs. Ils ont moins de contraintes sur le plan énergétique. Ils agissent comme une interface entre l'utilisateur et le réseau.

— **Utilisateur** : l'utilisateur exploite les données acquises et peut configurer les paramètres du réseau comme changer la fréquence des acquisitions.

1.3.1 Architecture d'un nœud

Un nœud est une architecture matérielle et logicielle qui peut effectuer plusieurs fonctionnalités [10]. La Figure 1.4 décrit l'architecture matérielle d'un nœud. Elle se compose généralement des éléments suivants :

— Un capteur qui réalise l'acquisition des données.

— Une unité de traitement pour l'analyse et le traitement des mesures acquises par les capteurs.

— Une unité de communication pour la réception et la transmission des données vers d'autres nœuds.

— Une mémoire pour le stockage temporel ou permanent.

— Un actionneur pour agir sur l'environnement.

— Une source d'énergie pour alimenter toutes ces unités.

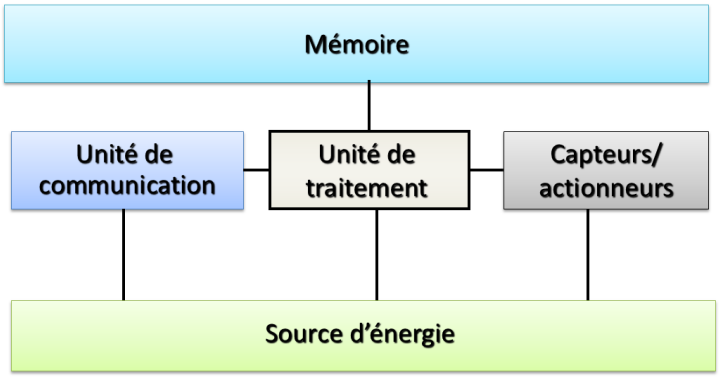


FIGURE 1.4 – Structure matérielle d'un nœud.

L'architecture logicielle implémente des fonctions comme l'acquisition de données, la communication avec un autre nœud, la sécurité pour protéger les informations échangées sur le réseau, la localisation dans une application de suivi, etc.

Selon la combinaison de l'architecture matérielle et logicielle de chaque nœud, cinq types de nœuds peuvent être identifiés :

— **Nœud capteur :** il comprend des unités d'acquisition de données comme la température et l'humidité. Ces données sont par la suite envoyées vers d'autres nœuds.

— **Nœud actionneur :** il comprend des actionneurs pour agir sur l'environnement.

— **Nœud routeur :** pour le routage de données entre les nœuds capteurs. Certains nœuds peuvent jouer le rôle de capteur et routeur en même temps.

— **Nœud passerelle :** ce nœud est équipé de deux unités de communication. Il s'agit d'un nœud chargé de connecter et de transmettre des données vers d'autres réseaux de communication par le biais d'autres technologies.

— **Nœud puits (Sink) :** il s'agit d'un nœud qui a pour tâche spécifique de recevoir, de traiter et de stocker les données provenant des autres nœuds capteurs.

Plateformes matérielles existantes

La conception d'un réseau de capteurs dépend forcement des nœuds capteurs souvent appelés mote (des plateformes faisant office de nœuds capteurs). Les motes les plus courants sont standardisés pour une utilisation dans les applications classiques [24]. La Figure 1.5 présente quelques exemples de ces motes comme le Mica2, TelosB, Iris et waspmote. Une étude comparative des différentes plateformes de capteurs est faite dans [25]. Les plateformes ont des caractéristiques différentes en termes de :

— Vitesse d'exécution,

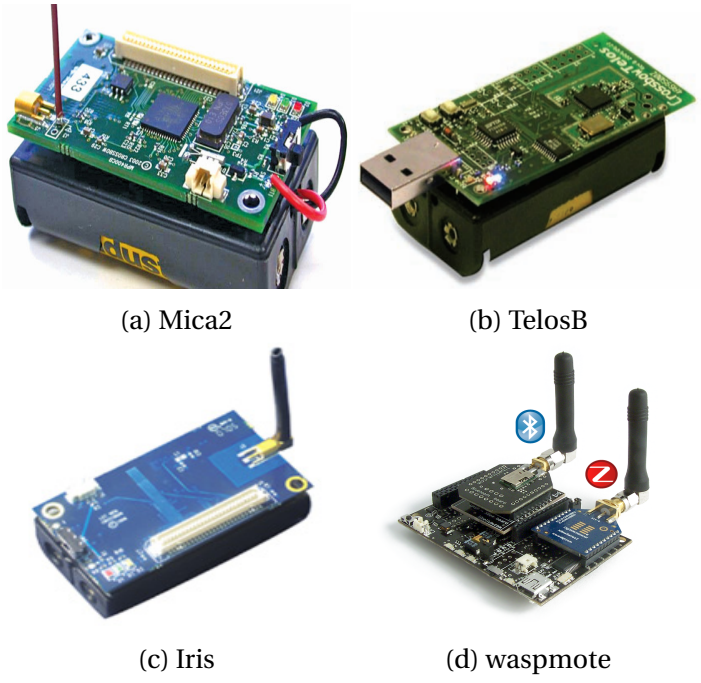
— Architecture de l'unité de traitement,

— Capacité mémoire (de programme et de données),

— Capteurs présents sur le mote,

— Nombre d'entrées et de sorties,

— Dimensions du mote.



(a) Mica2 (b) TelosB

(c) Iris (d) waspmote

FIGURE 1.5 – Exemples de plateformes matérielles existantes pour les réseaux de capteurs.

Systèmes d'exploitation des réseaux de capteurs

En raison de la diversité des applications des réseaux de capteurs et des plateformes matérielles de capteurs disponibles, des systèmes d'exploitation ont été conçus pour aider les utilisateurs à développer des applications complexes (par exemple, TinyOS, Contiki et MANTIS) [26]. Le service de base d'un système d'exploitation consiste à masquer les détails de bas niveau des nœuds capteurs en fournissant une interface de haut niveau facile à utiliser [27].

Certains systèmes d'exploitation sont encore en cours de développement et d'amélioration alors que d'autres ne sont plus maintenus. Nous avons constaté que la plupart des systèmes d'exploitation comme TinyOS supportent les motes de la série MICA conçue par UC Berkeley [28]. TinyOS a été étendu à de nombreuses plateformes matérielles de capteurs, bénéficiant ainsi d'une large diffusion. Presque tous les systèmes d'exploitation sont basés sur la programmation C, sauf TinyOS qui fournit son propre langage de programmation, nesC (une extension du langage C conçue pour incarner les concepts d'exécution de TinyOS [29]).

1.3.2 Protocoles de communication

Les protocoles de communication sont structurés en plusieurs couches. Chacune implémente des tâches réseau distinctes et fournit des services à sa couche supérieure. En procédant ainsi, les détails et la complexité de la mise en œuvre de la couche est masquée par rapport aux autres couches.

Le modèle d'Open Standards Interconnection (OSI) [30] propose une structure en couches de base pour les protocoles de communication dans les réseaux informatiques généraux. Dans ce modèle, un réseau est divisé en 7 couches.

La pile de protocoles de réseaux de capteurs proposée dans [1] est un modèle OSI simplifié qui convient mieux aux réseaux de capteurs. La Figure 1.6 représente cette pile qui est constituée de cinq couches horizontales et trois plans verticaux. Les couches verticales sont : application, transport, réseau, MAC et physique. Les plans de gestion de la consommation énergétique, de la mobilité et de la gestion des tâches sont utilisés par les nœuds pour réduire leur consommation énergétique globale, gérer leurs déplacements et coordonner leurs tâches. La mise en œuvre et l'optimisation de ces services peut se faire au niveau de toutes les couches.

La couche d'application fournit une interface entre les utilisateurs et les protocoles. Elle reçoit les données utilisateur et utilise les protocoles de couches inférieures pour établir des connexions. La couche transport fournit des services de communication de bout en bout, tels que la prise en charge des flux de données et le contrôle des flux orientés connexion.

Les données recueillies par les nœuds capteurs d'un réseau de capteurs sont transmises à la station de base où les données sont collectées et traitées. Dans les réseaux de

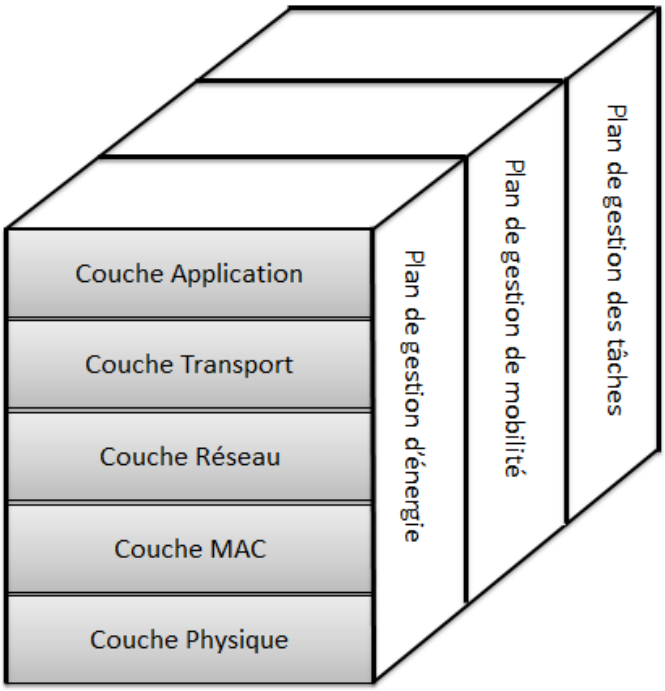


FIGURE 1.6 – Pile de protocoles des réseaux de capteurs [1].

petite taille où la station de base et les nœuds capteurs sont proches, les nœuds peuvent communiquer directement avec la station de base, on parle alors de communication à saut unique (*single-hop communication*). Pour couvrir une grande zone, il faut déployer un grand nombre de nœuds et ce scénario nécessite une communication à sauts multiples (*multi-hop communication*) car la plupart des nœuds capteurs sont si éloignés qu'ils ne peuvent communiquer directement avec la station de base. La communication à saut unique est aussi appelée communication directe et la communication à sauts multiples est appelée communication indirecte [31].

Dans les communications à sauts multiples, les nœuds capteurs non seulement détectent et délivrent leurs mesures, mais servent également de relais aux autres nœuds vers la station de base. Le processus de recherche d'un chemin approprié d'un nœud source à un nœud destination est appelé routage et c'est la fonction principale de la couche réseau.

Le protocole de routage peut être défini comme un processus de sélection du chemin approprié pour que les données transitent de la source à la destination. Le processus est confronté à plusieurs difficultés lors du choix de l'itinéraire, qui dépend du type de réseau, des caractéristiques du canal et des indicateurs de performance [31].

Le routage des données est lié à la topologie du réseau qui peut être de trois types, à savoir étoile, arbre, et maillée (Figure 1.7). En fonction de la topologie du réseau et des métriques à optimiser (durée de vie, latence, etc.), de nombreux protocoles de routage sont développés tels que *Flooding* [32] et *LEACH* [33]. Un aperçu des protocoles de routage pour les réseaux de capteurs est présenté dans [34].

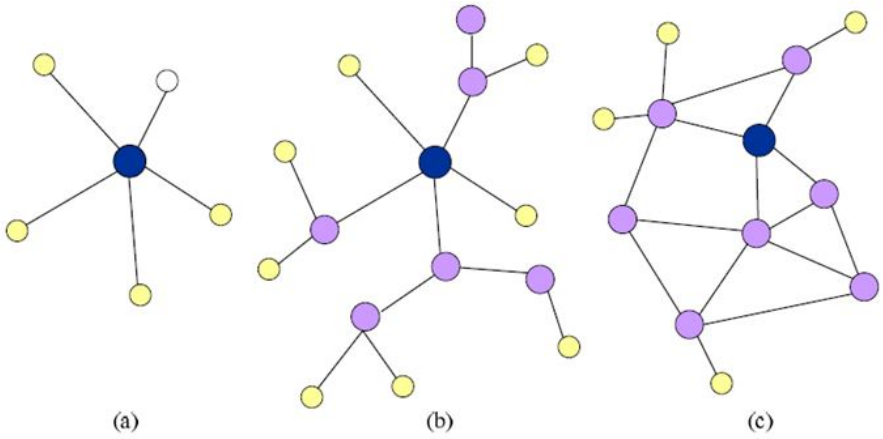


FIGURE 1.7 – Topologies de base des réseaux de capteurs : (a) étoile, (b) arbre, (c) maillée.

La couche MAC définit les procédures d'accès au canal afin d'éviter les pertes de données et les collisions de paquets. Elle assure la fiabilité de la transmission entre deux nœuds en utilisant certains mécanismes tels que les retransmissions et les acquittements (ack). Elle vise aussi à réduire la consommation d'énergie.

Les protocoles MAC peuvent être de type synchrone ou asynchrone. Dans les protocoles synchrones, les transmissions des nœuds sont organisées de manière ordonnée pour éviter les collisions (comme TDMA). Ainsi, l'énergie perdue à écouter le réseau se trouve réduite. Le protocole S-MAC est un exemple de protocole MAC synchrone conçu explicitement pour les réseaux de capteurs [35]. Les protocoles asynchrones tentent de réduire les probabilités de collision en utilisant des mécanismes d'accès aléatoire, comme le CSMA. Le protocole B-MAC est un exemple de protocole MAC asynchrone basé sur CSMA pour les réseaux de capteurs [36].

La couche physique définit les spécifications électriques et physiques des liaisons. Elle effectue le codage, la transmission, la réception et le décodage des données. Elle fournit des services de détection de porteuse et de détection de collision à la couche MAC.

Parmi les propriétés de cette couche figure la bande de fréquences de fonctionnement. Dans les réseaux de capteurs, de nombreuses technologies utilisent des bandes de fréquences dites *libres* telles que les bandes radio industrielles, scientifiques et médicales (ISM). Les bandes ISM varient selon les régions et elles sont soumises à des contraintes concernant les seuils de puissance d'émission à ne pas dépasser ainsi qu'un rapport cyclique (*duty-cycle*) qui doit se situer typiquement entre 0.1% et 10% suivant la fréquence et la région. La Figure 1.8 illustre les fréquences associées à trois zones, à savoir les USA/Canada, l'Europe et le Japon [2]. Ces fréquences sont définies et contrôlées par les agences FCC, ETSI, ARIB, respectivement. La bande 2.4 Ghz est autorisée partout dans le monde, elle est partagée par de nombreux protocoles (Wifi, ZigBee, Bluetooth, etc.). La bande en dessous du Giga (915MHz aux US et 868MHz en Europe) est un peu moins encombrée.

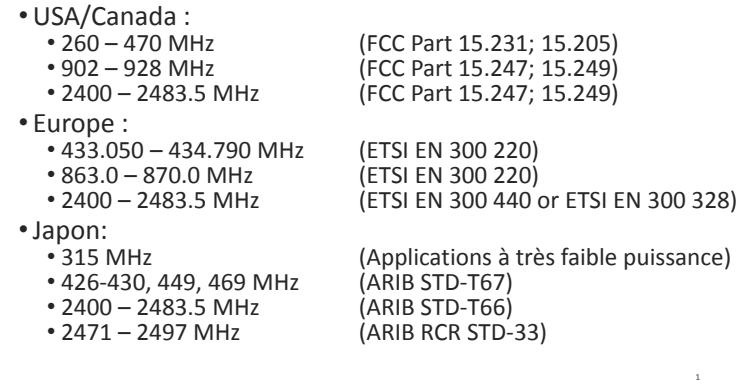


FIGURE 1.8 – Attribution du spectre des fréquences : Bandes de fréquences sans licence ISM/SRD (Short Range Devices) [2].

Pour soutenir le développement d'applications de réseau de capteurs et structurer l'ensemble des protocoles, des standards sont définis tels que le 802.15.4 et 6LoWPAN.

Standard IEEE 802.15.4

Le groupe IEEE 802.15 définit des standards qui permettent la connexion entre des systèmes pas trop distants et à faible puissance de transmission [37]. Dans ce contexte, le groupe IEEE propose le standard IEEE 802.15.4. Ce standard a été développé en 2003 et corrigé en 2006, spécifie les deux couches physique et MAC pour les réseaux faible débit.

Pour la couche physique, le standard opère sur les 3 bandes de fréquence : 868 MHz (Europe) avec 20 Kbps sur un canal de fréquence, 915 MHz (Amérique) avec 40 Kbps sur 10 canaux fréquentiels, 2450 MHz (monde entier) avec 250 Kbps sur 16 canaux fréquentiels. Il utilise deux techniques de modulations : *BPSK Binary Phase-Shift Keying* (8068,915 MHz) et *OQPSK Offset Quadrature Phase Shift Keying* (2450MHz).

Deux types de dispositifs différents peuvent se joindre à un réseau IEEE 802.15.4 [38] :

— FFD (Full Function Device), un FFD est un dispositif capable de servir de coordinateur PAN (Personal Area Network).

— RFD (Reduced Function Device), un RFD est un dispositif qui ne peut pas servir de coordinateur PAN. Il est destiné à des applications simples de détection et d'envoi ; il n'a pas besoin d'envoyer de grandes quantités de données et peut être associé à un seul FFD.

Ce standard définie deux modes pour la couche MAC :

— Mode non coordonné (*non beacon enabled mode*) : le dispositif communique des données dès que le canal est inoccupé.

— Mode coordonné (*beacon enabled mode*) : dans ce mode, le coordinateur transmet périodiquement des trames balises pour synchroniser les transmissions.

IEEE 802.15.4 fonctionne selon deux topologies : la topologie en étoile ou la topologie peer-to-peer. Il peut supporter 65000 dispositifs communicants. Une description détaillée du standard IEEE 802.15.4 se trouve dans [38]. Basé sur ce standard, des technologies de communication sont développées tels que zigbee et MiWi.

6LoWPAN

Avec la croissance actuelle d'Internet et la volonté de connecter plus d'objets [39–41], le nombre d'adresses IPv4 disponibles ne permet plus de répondre à ce besoin. Dans cette optique, IPv6 propose de multiplier la taille des adresses par 4, passant ainsi de 32 bits (IPv4) à 128 bits. Il en découle une augmentation significative du nombre d'objets connectés.

IPv6 over Low power Wireless Personal Area Network (6LoWPAN) propose une adaptation du protocole IPv6 au monde des réseaux LR-WAN [42, 43], plus précisément sur les couches physiques et liaisons définies par le standard IEEE 802.15.4.

En effet, la taille maximale de la trame 802.15.4 est de 127 octets avec 46 octets de configuration (l'en-têtes MAC, la sécurisation de la couche de liaison, la séquence de contrôle des trames). Il ne reste que 81 octets disponibles à la couche réseau.

Il faut tenir compte également d'en-têtes d'IPv6 (40 octets), des éventuels en-têtes d'UDP (8 octets) ou de TCP (20 octets). En conséquence, la taille des données utiles n'est pas significative (81-40-8=33 octets sur UDP et 81-40-20=21octets sur TCP). Sachant que les spécifications d'IPv6 imposent une longueur minimale pour le champ de données de 1 280 octets ; 802.15.4 ne permet pas donc de respecter ces spécifications.

Le protocole 6LoWPAN propose des mécanismes de fragmentation/recomposition, de compression d'en-tête et de configuration IP automatique pour permettre la transmission des trames IPv6 sur le 802.15.4 [44]. Ainsi, le dispositif émetteur divise un paquet IPv6 en plusieurs trames 802.15.4, et le dispositif récepteur réassemble toutes les trames reçues pour régénérer le paquet IPv6 original.

Dans ce protocole, une couche adaptation (6LoWPAN adaptation layer) est mise entre la couche liaison et la couche réseau du modèle OSI. Le schéma de routage de ce protocole peut être exécuté selon deux modes : mesh-under et route-over [45]. Mesh-under implémente le routage au niveau de la couche d'adaptation et la décision de routage est prise uniquement avec des fragments du paquet IPv6 [46, 47]. Route-over implémente le routage au niveau de la couche réseau et chaque périphérique intermédiaire reconstitue tous les fragments du paquet IPv6 pour prendre la décision de routage.

1.3.3 Technologies de communications

Il existe une grande diversité de technologies de communications utilisées dans les RdC. Ces technologies peuvent être classées en deux catégories : (I) les technologies de radiocommunication à courte portée telles les technologies ZigBee, Bluetooth, et (II) les

technologies de radio à longue portée comme LoRa et SigFox qui font aussi partie des réseaux Low Power Wide Area Network (LPWAN). Les réseaux LPWAN sont caractérisés par une longue portée de communication et une faible consommation énergétique [48]. Dans les sections suivantes, nous présentons de courtes descriptions de deux exemples de chaque technologie.

ZigBee

Zigbee est un ensemble de protocoles de communications de haut niveau créé par ZigBee alliance. Cette technologie est caractérisée par des transmissions radio à faible consommation, faible débit et de courte portée [49, 50].

ZigBee fournit les spécifications de la couche réseau et de la couche application et adopte la standard IEEE 802.15.4 comme base des couches MAC et physique. Il peut émettre sur les trois bandes de fréquences : 868 MHz, 915 MHz ou 2,4 GHz avec des débits allant de 20 à 250 kbps en fonction de la bande utilisée. Pour les bandes basses de 868 MHz et 915 MHz, la modulation est BPSK, tandis qu'à 2,4 GHz, il utilise la modulation OQPSK. La faible consommation d'énergie limite les distances de transmission à un rayon de 10-100 mètres en visibilité directe, en fonction de la puissance de sortie et des caractéristiques environnementales. Zigbee distingue trois types de périphériques : ZigBee-Coordinateur (ZC), ZigBee-Routeur (ZR), ZigBee End-Device (ZED). Il supporte la topologie en étoile, en arbre et maillée.

Les applications de ZigBee sont nombreuses et couvrent plusieurs domaines dont la domotique, les médias, l'éclairage, le chauffage, la climatisation, les loisirs et jouets, les systèmes d'alarmes, la détection d'incendie, les contrôles de puissance et la sécurité industrielle.

Bluetooth

Bluetooth est un protocole standardisé pour l'envoi et la réception de données via une liaison sans fil. La première version de Bluetooth [51] est parue en 1999. L'IEEE a standardisé Bluetooth selon la norme IEEE 802.15.1, mais ne maintient plus la norme [52].

Bluetooth est un protocole pour une courte distance (10 m) de communication [53]. Ce protocole fonctionne dans les fréquences de 2,4 GHz dans la même bande de fréquences que ZigBee. Dans la première version de Bluetooth, un groupe de dispositifs Bluetooth qui partagent un canal de fréquence commun est appelé un piconet. Chaque piconet dispose d'un dispositif maître et au maximum de 7 dispositifs esclaves. Dans un piconet, le canal est partagé à l'aide d'une division temporelle où le maître attribue des créneaux temporels aux nœuds esclaves [54]. Le maître peut envoyer des données aux esclaves et leur demander des données. Les esclaves sont autorisés à émettre et à recevoir uniquement vers leur maître. La première version de Bluetooth utilise la modulation

GFSK (Gaussian Frequency Shift Keying) avec un débit binaire de 1 Mbps.

Il convient de noter que la technologie Bluetooth est améliorée et offre actuellement d'autres technologies comme Bluetooth Low Energy appelée BLE et Bluetooth Mesh [55, 56].

Un grand nombre d'applications est développé en utilisant la technologie bluetooth. Parmi ces applications, nous pouvons mentionner les systèmes multimédias et le contrôle des maisons intelligentes.

LoRa

LoRa (Long Range) est une technologie de communications sans fil utilisée pour une longue portée de communication (LPWAN) [57]. La technologie LoRa est utilisée dans le réseau Long Range Wide Area Network (LoRaWAN). Il est à noter que les plateformes LoRa peuvent être utilisées avec ou sans le réseau LoRaWAN.

Les communications LoRa peuvent être effectuées sur une zone de couverture de 10 à 15 km dans les zones rurales et de 2 à 5 km dans les zones urbaines [3]. La topologie du réseau est de type étoile. Les communications entre les nœuds sont réparties sur les différents canaux fréquentiels dans la bande ISM (434, 868 et 902 MHz) avec des débits de données théoriques allant de 0,3 kbps à 37.5 kbps. La technologie LoRa possède dix modes de fonctionnement qui se traduisent par des débits différents. La sélection du mode dépend de la bande de fréquence, de la modulation et du facteur d'étalement [58].

De nombreuses applications de LoRa sont développées en domotique. Ces applications sont toutes connectées pour offrir des services à distance. Vue la sensibilité importante des récepteurs LoRa, des capteurs peuvent être enfouis dans le sol pour collecter des données telles que l'humidité. Ces informations peuvent par exemple être utilisées en agriculture. A l'échelle d'une ville, des voitures connectées sont aussi envisageables.

SigFox

Sigfox est une société française qui a inventé son propre protocole de radiocommunication. Il est dédié pour les réseaux de capteurs et l'Internet des objets. Selon le site officiel de SigFox (avril 2019), soixante pays et régions sont couverts par le réseau SigFox.

Sigfox est un réseau bas débit (100 bps), de faible coût et avec des années d'autonomie grâce à sa faible consommation. SIGFOX estime que chaque passerelle peut traiter jusqu'à un million d'objets connectés, avec une zone de couverture de 30 à 50 km dans les zones rurales et de 3 à 10 km dans les zones urbaines [3]. Le nombre et la taille des données à transmettre ou recevoir sont limités pour respecter les exigences de la bande ISM. La topologie du réseau Sigfox est de type étoile. Par rapport aux autres technologies, les produits Sigfox sont commercialisés avec un abonnement au réseau Sigfox [59].

De nombreuses applications peuvent être développées en utilisant le réseau SigFox. Par exemple, Smockeo est un détecteur de fumée connecté Sigfox [59].

Le tableau 1.1 présente une comparaison des exemples précédents. Il synthétise leurs caractéristiques en termes de portée de communication, de fréquence utilisée, de débit fourni et de topologie. Cette analyse nous permet de constater que plusieurs technologies sont proposées pour le déploiement des réseaux de capteurs et ces technologies présentent des caractéristiques différentes. Nous pouvons conclure que le choix d'une technologie dépend des exigences et des spécifications de l'application visée, telles que la taille du réseau (nombre de noeuds), l'environnement intérieur ou extérieur, la taille des données à communiquer, la faible consommation sur le plan énergétique.

TABLEAU 1.1 – Comparaison entre les technologies radio courtes et longues portées [3].

Technologie	**ZigBee**	**Bluetooth version 1.0**	**SigFox**	**LoRa**
Portée	10-100 m (LOS)	10 m (LOS)	30-50 km (rurale)	10-15 km (rurale)
fréquences (MHz)	868, 900, 2400	2400	868, 902	434, 868, 902
Débit brut maximal (kb/s)	20, 40, 250	1000	0.1	0.3–37.5
Topologie	étoile, en arbre, maillée	étoile	étoile	étoile
Nombre de Noeuds	65000	8 (piconet)	10 exp(6)	10 exp(4)

1.4 Propriétés d'un réseau de capteurs

Le réseau de capteurs se caractérise par un système distribué composé de plusieurs nœuds déployés dans un environnement physique. Ces nœuds communiquent entre eux et forment une topologie. Dans cette section, nous allons résumer quelques propriétés importantes des réseaux de capteurs.

1.4.1 Communications et flux de données

Dans certains cas, un faible débit est suffisant pour transmettre quelques octets qui portent des informations simples comme la température et la luminosité. A l'inverse, des applications, comme les réseaux de vidéosurveillance, nécessitent un débit de données important.

Dans un RdC, les nœuds de capteurs envoient les mesures à la station de base (communications montantes) alors que la station de base envoie les commandes aux nœuds de capteurs (communications descendantes). Il est à noter qu'il existe un déséquilibre entre les deux flux de communications ascendantes et descendantes en termes des quantités de données à communiquer.

1.4.2 Durée de vie

La durée de vie est considérée comme une caractéristique importante dans les réseaux de capteurs. Elle est liée à l'état des batteries et aussi au type d'application. Certaines applications ont une durée de vie courte, comme le cas des réseaux de capteurs

utilisés dans les catastrophes naturelles pour mieux gérer les équipes de sauvetage. Mais il existe aussi de nombreuses applications nécessitant des réseaux avec des durées de vie de plusieurs années comme les capteurs d'incendie. La durée de vie se délimite par le temps de déploiement du réseau et le temps de défaillance du premier nœud ou un certain pourcentage de nœuds. La défaillance d'un nœud est liée aux dysfonctionnements logiciels et matériels.

1.4.3 Déploiement et topologie

Le déploiement des nœuds dans un environnement physique peut se faire de deux manières selon la nature de l'application. La première concerne le déploiement aléatoire, par exemple, un déploiement rapide dans un champ de bataille. La deuxième est un déploiement organisé avec une distribution personnalisée des nœuds, par exemple, un déploiement de capteurs pour mesurer la température dans un bâtiment.

Après le déploiement du réseau, une topologie est établie. Elle permet de décrire les connexions entre les nœuds avec une configuration étoile, en arbre ou maillée. Cette topologie peut être statique ou dynamique suivant l'application visée.

1.4.4 Réseaux homogènes ou hétérogènes

Selon la composition des réseaux de capteurs, on trouve des réseaux homogènes et hétérogènes. Dans les réseaux homogènes, les nœuds ont la même architecture matérielle mais ils peuvent avoir des logiciels différents pour accomplir des tâches différentes. Par exemple, le logiciel des nœuds de capteurs n'est pas le même que celui de la station de base. Les réseaux hétérogènes se composent de nœuds avec des architectures matérielles différentes.

Une hétérogénéité peut être liée aussi aux technologies de communications comme nous l'avons souligné dans la section 1.3.3.

1.4.5 Mobilité

Dans un réseau, on peut avoir des nœuds statiques ou mobiles. Les nœuds statiques sont déployés à un endroit où ils restent statiques pendant toute la durée de vie du réseau. Dans un réseau mobile, les nœuds peuvent se déplacer dans un champ physique. Un exemple typique est celui des applications de suivi où les nœuds sont en mouvement [60, 61].

Par rapport aux nœuds statiques, il convient de noter que les nœuds mobiles nécessitent des algorithmes pour leur bon fonctionnement, en particulier le routage dynamique.

1.4.6 Sécurité, fiabilité, tolérance aux pannes

Les réseaux sans fil sont plus vulnérables aux attaques que les réseaux filaires. Il faut donc implémenter des algorithmes de sécurité pour la protection des données [62, 63].

Certains types d'applications traitent des données critiques comme les données médicales d'un patient. Assurer la réception de ces données est gérée par plusieurs mécanismes de communications comme la retransmission de l'information et les communications avec acquittement.

Certains nœuds peuvent échouer à transmettre des données pour différentes raisons comme le manque de puissance, des dommages physiques ou des interférences environnementales (ils fonctionnent souvent en bandes libres où la probabilité d'interférence est importante)[64]. L'échec de quelques nœuds de capteurs ne devrait pas affecter la tâche globale du réseau de capteurs. La tolérance aux pannes est la capacité de maintenir le fonctionnement du réseau de capteurs sans interruption malgré le dysfonctionnement de certains nœuds.

1.4.7 Cycle de vie

Le cycle de vie est défini comme une succession d'étapes qu'un produit ou un service parcourt dans le temps. Dans le domaine du développement logiciel, il s'agit d'un processus qui prend en compte la structure des étapes impliquées dans le développement d'une application. Ce processus est divisé en plusieurs phases telles que l'analyse des besoins, la conception, le codage, les tests, l'installation et la maintenance [65].

Dans le domaine des réseaux de capteurs, la conception et la mise en oeuvre des réseaux de capteurs implique différentes étapes : le dimensionnement du réseau, le développement de logiciels embarqués, la réalisation de matériels, des analyses et des simulations, le déploiement physique, l'exploitation des données, et la maintenance du réseau. Dans cette thèse, nous proposons de regrouper ces étapes dans un cycle de vie et qui peut être représenté par trois sous-cycles, à savoir le cycle de développement, le cycle de déploiement et le cycle d'exploitation du réseau (voir section 3.2).

1.5 Défis liés à la conception et la mise en œuvre d'un réseau de capteurs

La conception des réseaux de capteurs est influencée par divers facteurs qui peuvent se répartir selon les niveaux suivants : la consommation d'énergie, l'échelle du réseau, la maintenance, l'environnement physique de déploiement, les coûts de développement, et la complexité du développement. Ces facteurs sont importants parce qu'ils servent de lignes directrices pour concevoir et optimiser un réseau de capteurs.

1.5.1 Consommation d'énergie

Le nœud de capteur est un périphérique électronique, généralement équipé d'une source d'alimentation limitée [66]. Dans certains scénarios d'application, la reconstitution des ressources d'alimentation pourrait être impossible. Par conséquent, la durée de vie du nœud est directement dépendante de l'autonomie de la batterie.

La tâche principale d'un nœud de capteur est de détecter des événements, d'effectuer un traitement rapide des données locales, puis de transmettre ces données. La consommation électrique peut donc être divisée en trois domaines : la détection, le traitement des données et la communication. Le nœud dépense généralement le plus d'énergie pour la communication de données. Réduire la durée et le nombre de communications est un défi qui peut augmenter considérablement la durée de vie du capteur.

1.5.2 Réseau à grande échelle

Dans le cadre de l'étude d'un phénomène, le nombre de nœuds de capteurs déployés peut être de l'ordre de centaines ou de milliers. Dans un réseau à grande échelle, le dimensionnement du réseau est piloté par plusieurs choix comme le nombre de nœuds et les technologies de communication. Ces choix sont complexes par la dépendance des paramètres. Par exemple, augmenter le nombre de nœuds permet d'augmenter la probabilité de transmettre un message de bout en bout. Néanmoins, cela induit aussi une augmentation des coûts financiers. Par conséquent, des algorithmes d'optimisation sont nécessaires.

1.5.3 Maintenance

Pour avoir un réseau évolutif, la maintenance du réseau peut se faire à travers la reconfiguration des paramètres du réseau, la calibration des nœuds pour définir la relation entre les données et les grandeurs physiques et la reprogrammation des nœuds.

De plus, dans un réseau à grande échelle, il devient impossible de traiter manuellement chaque nœud séparément. Il faut créer un système de maintenance automatique.

1.5.4 Coûts

Les coûts financiers d'un réseau de capteurs peuvent être regroupés en deux catégories : les coûts liés au temps de développement et les coûts liés aux équipements et à leur exploitation. Il est très utile d'avoir une approche qui permet de réduire le temps de développement et de limiter les coûts tout au long du cycle de vie.

Comme les réseaux de capteurs peuvent avoir un grand nombre de nœuds, le coût d'un seul nœud est aussi très important pour réduire le coût global.

1.5.5 Environnement physique

Les nœuds de capteurs sont déployés dans un environnement physique. Des exemples de cet environnement incluent un champ contaminé chimiquement, un champ de bataille au-delà des lignes ennemies, une maison ou un grand bâtiment.

Cette liste illustre quelques conditions dans lesquelles les nœuds doivent fonctionner comme dans un champ de bataille ou dans des conditions climatiques extrêmes [67]. Cet environnement influence les conditions de propagation et donc sur la qualité de communication. Pour certaines optimisations, la prise en compte de cet environnement physique dans les simulations est très importante.

1.5.6 Complexité du développement

Le développement du réseau de capteurs est multi-domaines. Il nécessite des compétences en électronique pour la réalisation matérielle des capteurs, en réseau pour gérer un système distribué, en informatique pour le traitement des données ainsi qu'en télécommunications pour gérer les communications entre les capteurs. Par conséquent, plusieurs acteurs interviennent dans le cycle de vie d'un RdC. Les acteurs doivent continuellement collaborer avec un grand nombre d'intervenants du système dont les utilisateurs de réseaux de capteurs, les experts des domaines d'application, les concepteurs de matériel, les développeurs de logiciels, etc. Il n'est donc pas facile d'assurer la cohérence entre les acteurs impliqués.

Le cycle de vie d'un réseau de capteurs nécessite plusieurs étapes qui vont du dimensionnement à l'exploitation des données. Souvent, chaque étape est gérée par un outil particulier. Ces outils sont rarement reliés entre eux. Des descriptions sont requises pour chaque outil utilisé. Ces descriptions multiples sont coûteuses en termes de temps de développement et peuvent entraîner des erreurs de conception.

Le développement logiciel d'une application de réseau de capteurs est lié à des plateformes matérielles spécifiques, ce qui rend difficile la réutilisation du code source et des composants logiciels pour d'autres plateformes.

Une complexité supplémentaire peut être liée aux propriétés des réseaux de capteurs : un réseau hétérogène génère des problèmes de compatibilité et de cohérence dans les communications, la mobilité des nœuds impose une contrainte supplémentaire et particulièrement pour les algorithmes de routage dynamique, la sécurité nécessite le développement d'algorithmes complexes dédiés dans des plateformes caractérisées par des ressources limitées en termes de calcul et d'énergie.

1.6 Conclusion

Dans ce chapitre, nous avons étudié le contexte du réseau de capteurs. En particulier, nous avons présenté une définition d'un RdC, ses domaines d'application, son architec-

tures et ses propriétés.

Bien que les réseaux de capteurs soient de plus en plus utilisés dans les applications modernes, leur développement reste marqué par de nombreux défis tout au long de leurs cycles de vie. Ces défis impliquent la minimisation de la consommation d'énergie pour prolonger la durée de vie du réseau. Il est également difficile de gérer et d'assurer la maintenance d'un réseau à grande échelle. Par ailleurs, l'environnement physique de déploiement a un impact significatif sur les communications entre les nœuds. Il est donc important d'étudier cet impact avant le déploiement. Les développeurs ne sont pas seulement confrontés à des exigences fonctionnelles, mais aussi à des exigences non fonctionnelles telles que la sécurité, la mobilité et l'hétérogénéité.

Afin de simplifier la conception et la configuration du réseau de capteurs dans son ensemble et de surmonter ces défis, un certain nombre d'approches de modélisation des réseaux de capteurs ont été proposées.

Le prochain chapitre sera consacré à la présentation de l'état de l'art en ce qui concerne la modélisation des réseaux de capteurs dans le cadre d'une approche d'ingénierie dirigée par les modèles.

Chapitre 2

État de l'art sur l'utilisation des approches d'ingénierie dirigées par les modèles pour les réseaux de capteurs

« *Success is all about consistency around the fundamentals.* »

Robin Sharma

Sommaire

2.1 Introduction

Au cours des dernières décennies, la modélisation et la simulation ont touché une grande partie des domaines de l'ingénierie, tant dans le monde académique que dans l'industrie. La capacité de la modélisation et de la simulation s'est avérée essentielle dans de nombreux domaines pour accroître notre compréhension des systèmes, à les évaluer ou à prédire leurs évolution. Dans les réseaux de capteurs, un important effort de recherche a été effectué autour de la modélisation et de la simulation. Cependant, il reste encore quelques besoins pour automatiser et améliorer le développement des réseaux de capteurs. Parmi les techniques qui peuvent répondre à ces besoins, dans cette thèse, nous nous sommes appuyés sur des techniques d'ingénierie dirigées par les modèles.

Ce chapitre présente les concepts de base concernant les approches de modélisation dans les réseaux de capteurs. Nous nous focalisons sur les approches d'ingénierie dirigées par les modèles (IDM). Nous avons établi également des critères de comparaison pour évaluer les approches de conception des réseaux de capteurs basées IDM. Les résultats de cette étude permettent d'en identifier les mérites et les limites potentielles.

2.2 Approches de modélisation et de simulation

Différentes techniques de modélisation et de simulation sont couramment utilisées par les concepteurs du RdC pour répondre aux questions de choix du matériel, choix des protocoles de communication et de leurs paramètres ainsi que les choix d'optimisations (MAC/routage). On peut classer ces techniques en : les méthodes analytiques [68], les bancs d'essai physiques [69] et les simulations [70].

Les contraintes imposées aux réseaux de capteurs, telles que les ressources limitées et la tolérance aux pannes, nécessitent l'utilisation d'algorithmes complexes qui rendent généralement les méthodes analytiques impossibles [71]. L'utilisation de bancs d'essais physiques souffre également de certaines limitations importantes, telles que le coût et le problème d'évolutivité. Il est coûteux et fastidieux d'établir un banc d'essai pour un réseau comptant des milliers de nœuds. Cependant, la simulation peut fournir une bonne approximation à moindre coût et souvent en un temps réduit. La simulation fournit également un environnement de débogage facile à utiliser et une meilleure compréhension des comportements du réseau. Par conséquent, la simulation est devenue un moyen courant d'évaluer les performances [72].

De nombreux outils de simulation (simulateurs et émulateurs) pour les réseaux de capteurs ont été développés. L'émulateur est un outil de simulation spécifique à certaines plateformes et permet d'exécuter le même code embarqué sur celles-ci [73, 74] , par exemple, l'outil Avrora permet d'émuler la plateforme Mica2 [75].

Selon les travaux rapportés dans [15], l'ensemble des outils de simulation peuvent être classés en quatre catégories :

— Simulateurs de réseau avec modèles de nœuds : NS-2 [76], OMNeT++ [5], WSNet [77], SENSE [71], GloMoSim [78], Prowler [79].

— Simulateurs de réseau avec émulateurs de nœuds : EMSIM [80], sQualNet [81].

— Émulateurs de nœuds avec modèles de réseau : TOSSIM [82], ATEMU [83], Worldsens[77], COOJA [84], Avrora [75].

— Simulateur de système de nœud avec modèles de réseau : WISENES [85], ATLeS-SN [86], SCNSL[87].

Les développeurs utilisent des simulateurs/émulateurs pour faire des optimisations. Ils développent ensuite le code source à embarquer dans les capteurs. Souvent, ils utilisent des outils spécifiques à chaque étape du cycle de vie d'un RdC.

Cette approche est probablement la méthodologie de développement la plus fréquemment utilisée dans le développement des RdC. Néanmoins, elle souffre d'un certain nombre de limites : les étapes du cycle de vie d'un RdC ne sont pas liées entre elles en raison d'absence de liens entre les outils utilisés à chaque étape. Il est donc nécessaire de redécrire le système dans chaque outil, ce-qui peut conduire à des développements incohérents, à un temps de développement considérable et des coûts importants.

Les approches d'ingénierie dirigées par les modèles (Ingénierie Dirigée par les Modèles (IDM)) permettent de répondre à ces limites; cette thèse s'inscrit dans ce contexte. Nous focalisons donc notre état de l'art uniquement sur ce type d'approche. Nous commençons dans la section suivante par introduire des généralités sur l'approche IDM.

2.3 Ingénierie dirigées par les modèles (IDM)

Au cours des dernières années, le développement et la maintenance des applications logicielles se sont améliorés grâce à une nouvelle approche du génie logiciel appelée Ingénierie Dirigée par les Modèles (IDM) [88] (Model Driven Engineering (MDE)). Elle est proposé par l'Object Management Group (OMG) [1] [89] pour modéliser les applications à un haut niveau d'abstraction, puis de générer le code de l'application à partir des modèles [90]. Cette approche est basée sur les notions de modèles, métamodèles et transformations de modèles.

2.3.1 Modèle

Un modèle est une abstraction, une simplification d'un système qui est suffisante pour comprendre le système modélisé et répondre à un certain nombre de questions sur celui-ci [91–93]. Cette simplification se traduit souvent par la disparition de détails d'ordre technique [94].

1. L'OMG est un consortium international qui représente de nombreuses institutions académiques et industrielles fondé en 1989.

Un modèle peut être une description d'un système existant ou une spécification d'un système à construire. Il vise à mieux comprendre un système en cours de développement. Il peut être exprimé dans différents langages et peut être visualisé de différentes manières.

2.3.2 Métamodèle

Un métamodèle est une spécification d'un ensemble de concepts utilisés pour exprimer des modèles, ainsi que les relations entre ces concepts. Le métamodèle définit donc la terminologie à utiliser pour définir des modèles [95]

Un métamodèle formalise aussi la sémantique donnant un sens aux éléments définis [92]. L'utilisation d'assertions logiques ou d'un moyen de description formel permettent d'exprimer des contraintes au niveau des concepts définis.

La Figure 2.1 illustre un exemple de métamodèle *Library*. Le métamodèle définit les éléments modélisés (*Book*, *Writer*, *Employee*) et leurs paramètres (attributs) associés. Par exemple, l'élément *Book* possède trois attributs dont le titre, le nombre de pages et une catégorie. Ce métamodèle contient également une description des contraintes, par exemple, un auteur (*Writer*) possède au moins un livre (ceci est décrit dans le métamodèle par la contrainte 1..* comme illustré dans la Figure 2.1).

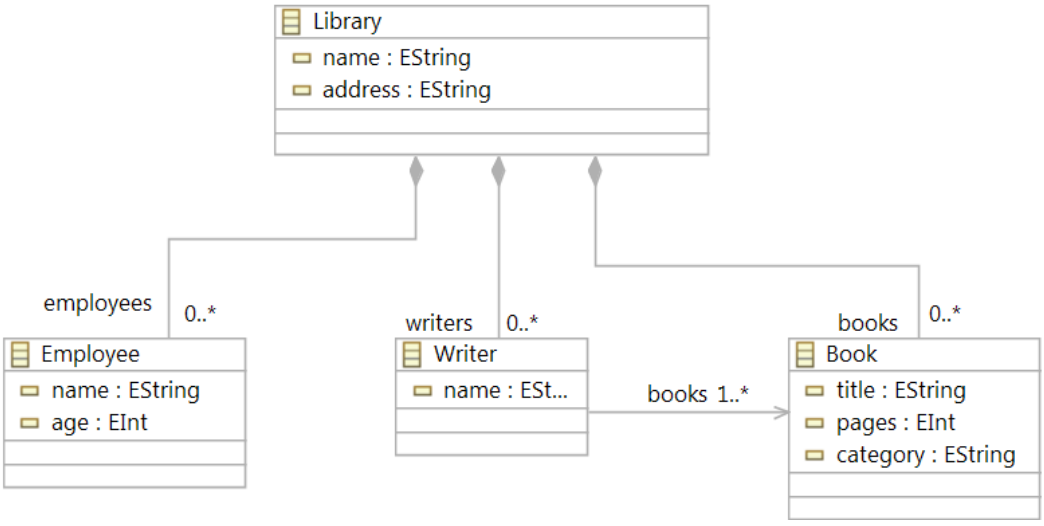


FIGURE 2.1 – Exemple de métamodèle "Library".

En fonction des domaines d'application, des métamodèles peuvent être définis de manière spécifique car il est difficile de créer un métamodèle universel. Ces métamodèles spécifiques introduisent alors la notion de langage de modélisation spécifique à un domaine (Domain Specific Modelling Language (DSML)).

2.3.3 Relation entre modèle et métamodèle

Á l'image de l'approche orientée objet où l'on trouve les deux relations de base "*instance de*" et "*hérite de*", l'approche IDM, définit les deux relations de base "*représenté par*" et "*conforme à*" [96]. La relation "*représenté par*" permet d'exprimer qu'un système est représenté par un modèle alors que la relation "*conforme à*" permet de dire qu'un modèle est décrit dans un langage conforme à son métamodèle.

Dans cette optique, l'OMG a proposé une architecture de modélisation qui décrit le lien entre les notions de modèle et métamodèle. Comme illustré dans la Figure 2.2, cette architecture comporte quatre niveaux de modélisation :

— **Niveau M0 :** il correspond au niveau du système réel que l'on veut modéliser ;

— **Niveau M1 :** il correspond au modèle qui permet représenter le systèmes à M0. Les modèles de ce niveau sont formalisés par le niveau M2 ;

— **Niveau M2 :** il correspond au métamodèle qui est décrit avec des langages de description. Les modèles de ce niveau sont formalisés par le niveau M3 ;

— **Niveau M3 :** il correspond au niveau méta-métamodèle qui peut décrire lui-même sa sémantique.

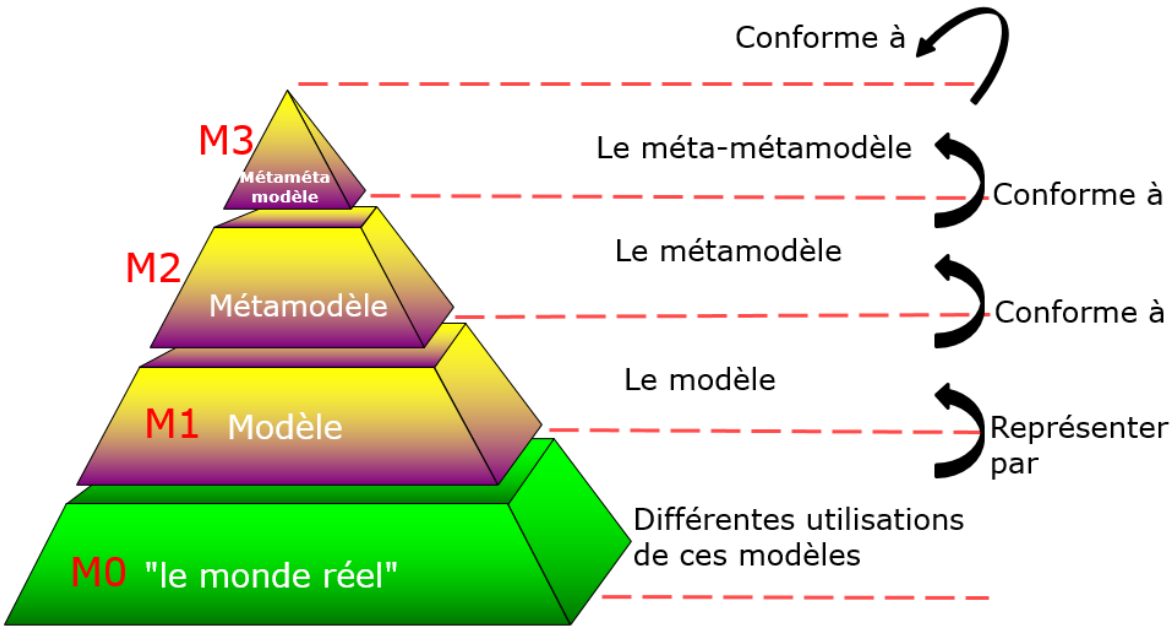


FIGURE 2.2 – Architecture à quatre niveaux.

2.3.4 Transformations de modèles

L'approche IDM supporte des transformations de modèles qui analysent certains aspects des modèles et synthétisent ensuite d'autres représentations de modèles. Le passage d'une représentation vers une autre est basé sur des règles de transformations bien

définies entre les métamodèles source et cible (voir Figure 2.3). L'approche distingue deux types de transformations : les transformations de modèle vers modèle et les transformations de modèle vers texte.

— Les transformations *Model-to-Model (M2M)* permettent de créer des liens entre les modèles. Elles assurent la cohérence des modèles et réduisent le temps nécessaire pour les redécrire ;

— Les transformations *Model-to-Text (M2T)* permettent de transformer des modèles vers du texte (modèles cible) afin de générer automatiquement texte [97].

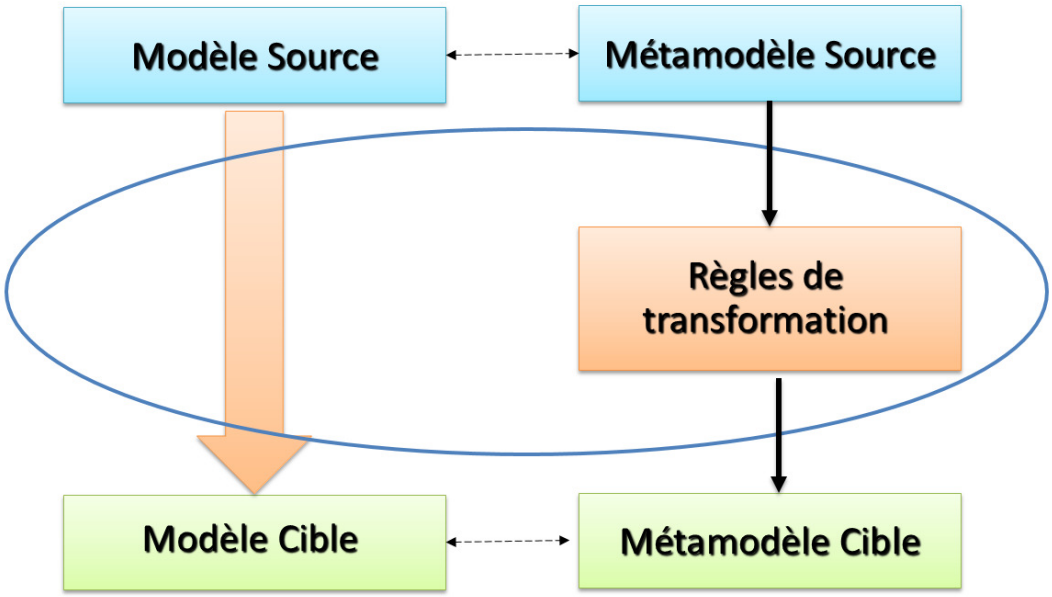


FIGURE 2.3 – Transformations de modèles dans l'approche IDM.

2.3.5 Model-Driven Architecture (MDA)

En novembre 2000, l'OMG a publié l'initiative Model-Driven Architecture (MDA), une variante particulière de l'approche d'ingénierie dirigée par modèle [96]. MDA sépare les besoins applicatifs d'un système des détails de la plateforme utilisée.

Dans l'approche MDA, les modèles de systèmes sont divisés suivant trois points de vue [98] :

— Computation Independant Model (CIM), un modèle spécifiant un système du point de vue de l'utilisateur ;

— Plateform Indépendant Model (PIM), un modèle spécifiant une application indépendamment de la technologie de mise en oeuvre ;

— Platform Specific Model (PSM), un modèle spécifiant une application après projection sur une plateforme technologique spécifique.

Les modèles sont développés à base des profils UML. Les métamodèles peuvent être développés avec des outils dédiés UML comme papyrus [99]. Les transformations de modèles dans MDA sont principalement des transformations de raffinement, c'est-à-dire des transformations qui augmentent le niveau de détails contenus dans un modèle. Une pratique courante dans MDA est de générer un PSM à partir d'un PIM. Pour spécifier les transformations, l'OMG a adopté un ensemble standard de langages appelés Query View Transformation (QVT).

Nous notons que MDA est une approche très restreinte du IDM. Elle met fortement l'accent sur l'UML et la notion de plateforme indépendante et spécifique.

2.3.6 Outils de mise en oeuvre des approches IDM

Au cours de la dernière décennie, un vif intérêt pour l'ingénierie dirigée par les modèles a donné naissance à diverses outils qui facilitent et accélèrent le développement des environnements DSML. Les outils les plus pertinents incluent IBM Rational Software Architect (RSA)[100], Generic Modeling Environment (GME) [101], IBM Rational Rhapsody [102], MetaEdit+ [103], Obeo Designer [104] et Eclipse Modeling Project (EMP) [105].

Par rapport à ces outils, le projet EMP constitue un outil riche en termes de mise en œuvre des concepts de l'IDM. Il est basé sur la plateforme Eclipse, un environnement de développement logiciel libre et open-source créé par IBM. Cette plateforme, hautement extensible notamment grâce au mécanisme des plugins, propose un environnement de définition et de mise en œuvre des outils DSML [106]. C'est pour cette raison que nous avons adopté le projet EMP dans cette thèse.

EMP englobe des outils de développement suivant trois axes principaux, à savoir le développement de métamodèle, d'environnement graphique et de transformations de modèles.

Outil de description de métamodèle : EMF

Eclipse Modeling Framework (EMF) est un framework pour la création, l'édition et la validation des métamodèles. Les métamodèles EMF sont décrits avec un format XMI (XML Metadata Interchange) dans un fichier *ecore*. Comme illustré dans la Figure 2.4, le métamodèle peut être défini de trois façons [105] :

1. Interfaces Java ;
2. Diagramme de classes UML ;
3. Schéma XML.

Sur la base d'une spécification d'un metamodèle, EMF fournit des outils et un support de génération d'un ensemble de classes Java constituant un éditeur de base des modèles. Il fournit un ensemble de classe Java pour manipuler les modèles par programmation.

Une caractéristique importante d'EMF consiste à fournir un noyau pour tous les autres projets de modélisation présents dans Eclipse. Il fournit alors une base de d'interopérabilité avec d'autres outils.

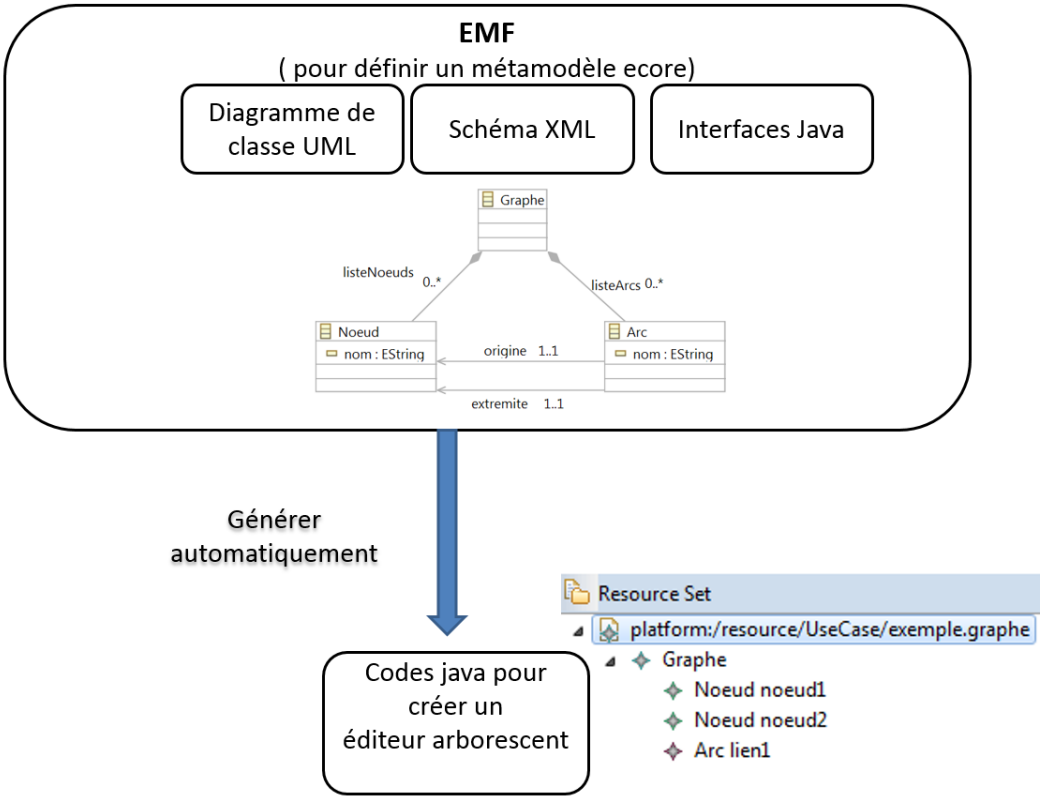


FIGURE 2.4 – Framework de modélisation d'Eclipse : EMF.

Outil de description du syntaxe graphique : GMF

Graphical Modeling Framework (GMF) est un Framework de création d'un environnement graphique de modélisation à partir d'un métamodèle. Il prend en entrée :

— La définition du métamodèle, le fichier ecore, décrit dans EMF ;

— La définition des éléments de la palette de l'environnement graphique ;

— Les représentations graphiques de chaque éléments du modèle. Par exemple, il est possible de décrire un nœud sous forme d'un cercle (objet graphique).

En combinant ces entrées, GMF génère automatiquement des codes java qui constituent l'interface graphique de l'environnement de modélisation (voir Figure 2.5). Pour plus d'informations, François Tanguy [107] fournit un guide détaillé pour créer un éditeur graphique en utilisant GMF.

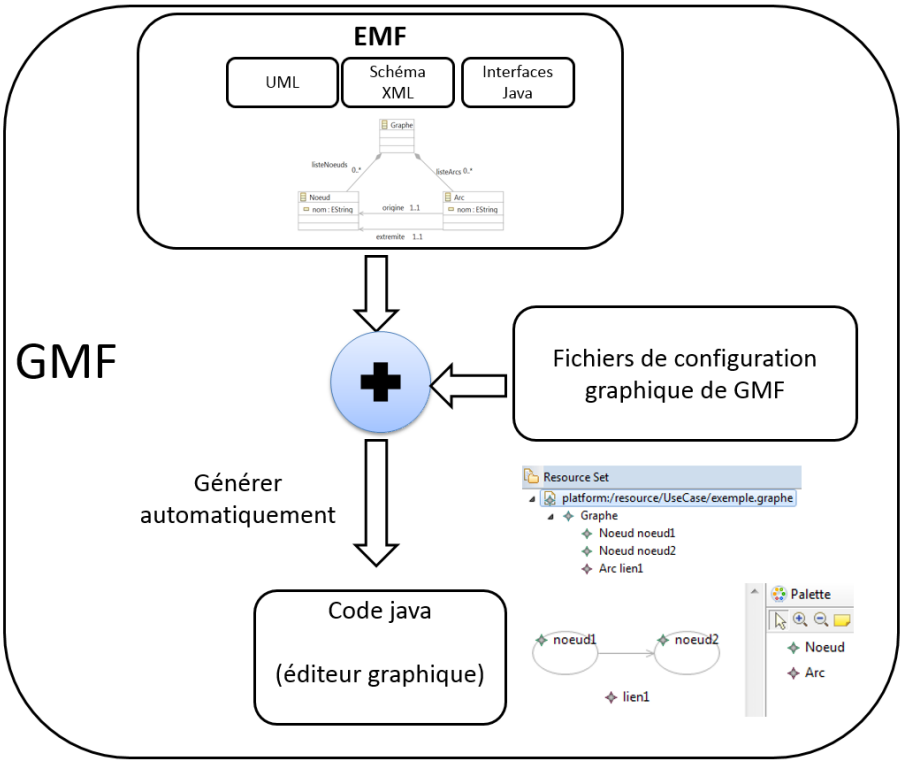


FIGURE 2.5 – Framework de modélisation graphique d'Eclipse : GMF.

Outils de transformations de modèles vers modèles : ATL

Les transformations de modèles vers modèles sont définies par des règles de transformations entre les métamodèles associés. Ces règles sont implémentés dans des outils comme ATLAS Transformation Language (ATL) [108]. ATL est un plugin d'Eclipse et un langage de transformation de modèles. Il transforme un modèle source en modèle cible, les deux sont distincts. Le modèle source est uniquement en mode lecture et le modèle cible est uniquement en mode écriture.

La Figure 2.6 illustre un exemple de transformation de modèle vers modèle. Ces transformations sont décrites dans un ensemble de règles qui définissent la relation entre les éléments du métamodèle source et les éléments du métamodèle cible.

Outils de transformation de modèle vers texte : Acceleo

Les transformations de modèle vers texte sont définies par des règles de transformations par rapport au métamodèle associé. Ces règles sont implémentés dans des outils comme Acceleo, un plugin de Eclipse [109]. Acceleo est souvent utilisé afin de générer du code pour une plateforme technologique spécifique.

La Figure 2.7 illustre un exemple de transformation de modèle vers texte avec Acceleo. Les transformations sont décrites dans un *module* qui lit les éléments du modèle source et les transforme vers du texte.

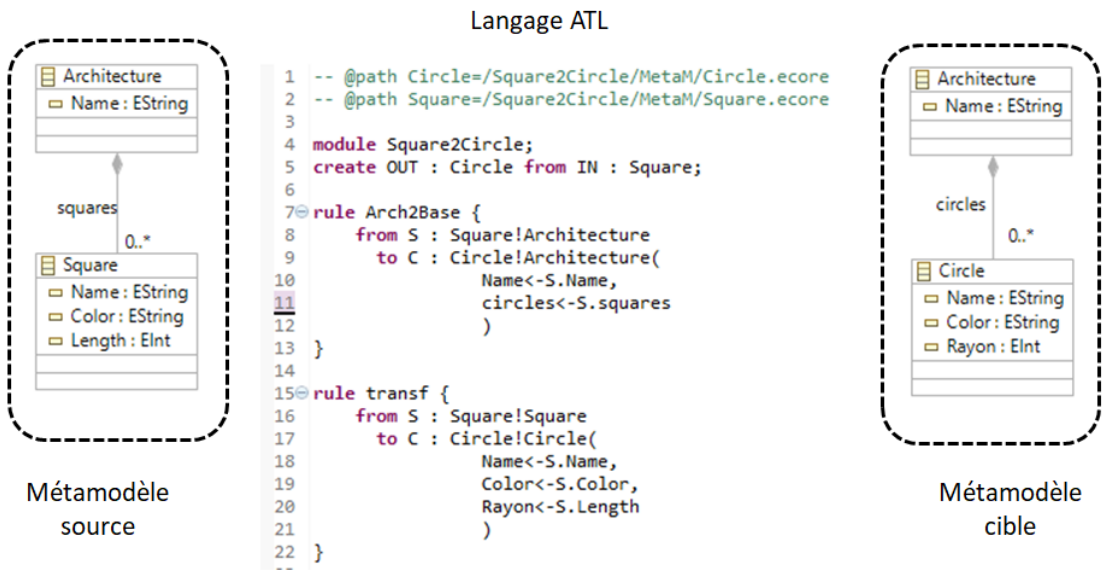


FIGURE 2.6 – Exemple de règles de transformations ATL.

FIGURE 2.7 – Exemple de règles de transformations sous Acceleo.

2.4 Approches IDM utilisées dans les réseaux de capteurs

Pour faire face à la complexité du processus de développement des réseaux de capteurs, de nombreux travaux suggèrent d'utiliser les approches d'ingénierie dirigées par les modèles. Nous proposons d'étudier ces approches de manière à mieux comprendre comment les techniques IDM ont été utilisées pour concevoir et analyser les réseaux de capteurs.

Dans cette étude, nous avons sélectionné 20 approches, référencées de A1 à A20, en nous focalisant sur l'utilisation de l'approche IDM pour la modélisation, la conception, l'analyse ou le développement de réseau de capteurs. Nous nous sommes particulièrement intéressés aux trois aspects suivants :

— Les objectifs de chaque approche et notamment la couverture du cycle de vie des réseaux de capteurs ;

— Les éléments et expressivité des modèles introduits par les approches ;

— La réalisation et la mise en œuvre associées (environnement et outil utilisés).

2.4.1 Couverture du cycle de vie

Cette section traite de la principale motivation pour laquelle les chercheurs ont utilisé l'approche IDM dans les réseaux de capteurs. On classifie principalement ces approches en fonction de la couverture du cycle de vie. Pour cela, nous étudierons ces approches en essayant de répondre aux questions suivantes :

— Est-ce que l'approche modélise le réseau de capteurs ?

— Est-ce que l'approche permet d'analyse ou simuler des réseaux de capteurs ? Quelles métriques sont considérées (consommation énergétique, latence, etc.) ?

— Est-ce que l'approche traite de la génération de code ? Quels sont les langages cibles ?

— Est-ce que l'approche traite de la visualisation et de l'exploitation des données ?

Le tableau 2.1 présente les résultats de notre analyse des principaux objectifs de certaines approches IDM. L'analyse de ce tableau permet de tirer les conclusions suivantes :

— La majorité des approches modélisent le réseau de capteurs avec des langages différents. Certaines approches, telle que A16, se distinguent par une modélisation détaillée. D'autres approches telle que A13 proposent simplement des concepts peu détaillés sur le langage de modélisation ;

— Les approches analysent souvent les performances des réseaux de capteurs. Cette tendance a un sens puisque l'amélioration des performances est parmi les défis les plus importants dans le développement des réseaux de capteurs. La moitié des approches réalisent des analyses ou des simulations en intégrant des simulateurs. La performance liée à la consommation d'énergie est traitée par plusieurs articles

parce qu'elle est considérée comme l'un des principaux défis dans le domaine des réseaux de capteurs ;

— La moitié des approches intègre la génération de code dans son processus de développement pour gagner en temps de développement, éviter les erreurs de codage ainsi que réduire la complexité de développement. NesC est le langage le plus ciblé pour la génération de code car il est le standard pour développer les applications des réseaux de capteurs. Ce choix est lié aussi à utilisation des plateformes capteurs basées sur le langage nesC dans la démonstration de l'approche. Par exemple, les auteurs de l'approches A7 [116] ont défini et développé deux couches de modélisation : PIM et PSM. Ils ont défini la couche PIM pour permettre aux développeurs d'applications de modéliser la logique applicative, tandis que la couche PSM est définie pour modéliser des plateformes cibles. Les modèles PIM sont transformé en PSM à l'aide des transformations de modèles. Le processus de transformation dans A7 permet d'obtenir un code source avec le langage nesC ;

— Quelques approches fournissent à la fois des possibilités d'analyse / simulation et de génération de code. Par exemple, l'approche de A12 [121] présente des modèles de conception UML. Le simulateur de réseau Avrora a été intégré à l'environnement de conception UML pour analyser la consommation d'énergie du processus de communication ;

— En ce qui concerne l'exploitation des données, il convient de noter que ces travaux

TABLEAU 2.1 – Couverture du cycle de vie les approches IDM .

Approche	Modélisation	Simulation	Génération de code	Exploitation de données
A1 [110]	✓	✓	×	×
A2 [111]	✓	×	NesC	×
A3 [112]	✓	×	ANSI C, nesC	×
A4 [113]	✓	×	NesC	×
A5 [114]	✓	×	×	×
A6 [115]	✓	✓	NesC	×
A7 [116]	✓	×	NesC	×
A8 [117]	✓	✓	×	×
A9 [118]	✓	✓	NesC	×
A10 [119]	✓	✓	×	×
A11 [120]	✓	×	×	×
A12[121]	✓	✓	NesC	×
A13 [122]	×	✓	C, nesC	×
A14 [123]	✓	×	C++	×
A15 [124]	✓	✓	×	×
A16 [125]	✓	✓	nesC	×
A17 [126]	✓	×	nesC	×
A18 [127]	✓	×	java, sql	×
A19 [128]	✓	×	nesC	×
A20 [129]	✓	✓	nesC	×

n'ont pas abordé cette étape ou bien ne la décrivent pas de manière suffisamment explicite.

2.4.2 Expressivité des modèles

Cette section se concentre sur les principales caractéristiques du langage de modélisation utilisé et le niveau d'abstraction associé. Nous cherchons à déterminer la capacité et l'expressivité de ces langages selon les critères suivants :

— Est-ce que l'approche permet aux concepteurs de représenter le matériel (HW) et / ou le logiciel (SW) d'un nœud ?

— Est-ce que l'approche représente les propriétés liées au réseau, à la connectivité réseau et à la topologie associée ?

— Est-ce que l'approche décrit l'environnement physique du déploiement pour étudier l'influence de celui-ci sur le réseau de capteurs ?

— Est-ce que l'approche est capable de représenter les infrastructures d'exploitation de données comme la base de données et les services utilisateur ?

Le tableau 2.2 présente les résultats de cette etude. Sur les éléments modélisés, on peut remarquer :

TABLEAU 2.2 – Éléments identifiés dans les approches IDM.

Approche	Noeud	Réseau	Déploiement Physique	Infrastructure de stockage	Services pour utilisateur
A1 [110]	Sw	✓	✓	×	×
A2 [111]	Sw/Hw	✓	✓	×	×
A3 [112]	Sw	✓	✓	×	×
A4 [113]	Sw/Hw	✓	×	×	×
A5 [114]	Sw	✓	×	×	×
A6 [115]	Sw/Hw	✓	×	×	×
A7 [116]	Sw	✓	×	×	×
A8 [117]	Sw/Hw	✓	✓	×	×
A9 [118]	Sw/Hw	×	✓	×	×
A10 [119]	Sw/Hw	✓	✓	×	×
A11 [120]	Sw	✓	×	×	×
A12 [121]	Sw/Hw	✓	×	×	×
A13 [122]	Sw	×	×	×	×
A14 [123]	Sw/Hw	✓	×	×	×
A15 [124]	Sw/Hw	✓	✓	×	×
A16 [125]	Sw/Hw	×	×	×	×
A17 [126]	Sw	✓	×	×	×
A18 [127]	Sw	×	×	×	×
A19 [128]	Sw	✓	×	×	×
A20 [129]	Sw/Hw	✓	✓	×	×

— Certaines approches sont au niveau nœuds et d'autres au niveau réseau. Il est intéressant de noter que l'approche de modélisation proposée dans A17 [126] comprend trois niveaux : nœud, groupe de nœuds ou réseau ;

— La majorité des approches sélectionnées ne permettent pas aux concepteurs de spécifier l'environnement physique du déploiement. Il y a peu d'approches qui soutiennent une définition explicite de l'environnement physique comme le propose par exemple l'approche A9 [118] avec des zones et des obstacles. D'autres approches permettent aux concepteurs de définir des concepts physiques simples comme l'approche A8 [117] avec ses modèles de coordonnées ;

— Aucune approche ne permet de faire la modélisation des infrastructures de stockage et d'exploitation des données.

— Les éléments de modélisation sont souvent spécifiques au cas d'étude. Ces derniers sont menés à petite échelle pour expérimenter seulement des applications simples telles que les systèmes de surveillance médicale à domicile des patients ou bien des systèmes d'alarme incendie. Pour les démonstrations, ils utilisent des motes comme iMote, MicaZ, Z1 mote, Sunspot, Mica2.

2.4.3 Environnement et outil d'aide à la conception

Nous nous intéressons dans cette section à la mise en œuvre de ces approches pour fournir un environnement d'aide à la conception. Notre objectif est de nous concentrer sur les méthodes utilisées pour réaliser cette approche ainsi que sur les outils utilisés pour construire les modèles associés. Nous nous basons sur des critères répondant aux questions suivantes :

— Les modèles sont-ils représentés graphiquement, textuellement ou la combinaison des deux (mixte) ?

— Quels sont les outils de modélisation existants utilisés pour développer ces approches ?

— Quels sont les outils de simulation et d'analyse qui sont utilisés dans ces approches ?

Le tableau 2.3 présente les résultats de cette analyse. Nous pouvons noter que :

— Trois syntaxes sont utilisées : la syntaxe textuelle, la syntaxe graphique et la combinaison des deux (mixte). Cependant, en ce qui concerne la syntaxe graphique, nous remarquons une certaine variabilité. Par exemple, A8 fournit une notation graphique similaire à celle des machines d'états, dans A11 les auteurs réutilisent les diagrammes d'activité UML, A16 présente une notation qui est proche d'un diagramme de flux ;

— Plusieurs approches ont été mises en œuvre via la plateforme Eclipse, une plateforme de développement open-source composée de framework et d'outils extensibles ;

— Pour les simulateurs, nous avons remarqué l'utilisation de plusieurs simulateurs comme TOSSIM, Cooja, Avrora, Omnet, Castalia. Ces approches utilisent Parfois des programmes en java ou Matlab pour effectuer des analyses de performances.

TABLEAU 2.3 – Mise en œuvre des approches IDM dans le domaine des réseaux de capteurs.

Approche	Méthode	Outils	Simulateurs
A1 [110]	Mixte	Ptolemy II	Java
A2 [111]	Graphique	Eclipse	—
A3 [112]	Mixte	Matlab et Simulink	—
A4 [113]	Mixte	Eclipse	TOSSIM
A5 [114]	Graphique	Papyrus UML	—
A6 [115]	Textuelle	Eclipse	TOSSIM
A7 [116]	Mixte	Eclipse	—
A8 [117]	Textuelle	Eclipse	java
A9 [118]	Mixte	Eclipse	omnet++
A10 [119]	Mixte	SysML, Simulink, Modelica	matlab
A11 [120]	Graphique	Eclipse	—
A12 [121]	Mixte	Rhapsody	Avrora
A13 [122]	Mixte	CPN,PetriCode	Mixim
A14 [123]	Mixte	MARTE	—
A15 [124]	Mixte	UML/MARTE, UNIVERCM, SystemC	SCNSL
A16 [125]	Mixte	Eclipse	Cooja
A17 [126]	Textuelle	Eclipse	—
A18 [127]	Mixte	Eclipse	—
A19 [128]	Mixte	Eclipse	—
A20 [129]	Mixte	Ptolemy	TOSSIM

Ces choix sont souvent liés au choix de la plateforme utilisée (Avrora et TOSSIM pour les plateformes TinyOS, Cooja pour les plateformes Contiki). Omnet++ et Castalia sont utilisés indépendamment des plateformes ;

— Les approches étudiées sont en majorité adaptées pour résoudre un problème spécifique lié à la conception des réseaux. Le langage de modélisation fourni peut donc ne pas couvrir toutes les préoccupations possibles. L'une des solutions classiques à cette limitation est de fournir des possibilités d'extension, de sorte que les concepteurs puissent étendre le langage de modélisation ou le Framework sous-jacent afin de répondre à leurs besoins. Dans notre étude, de nombreux articles ne fournissent pas suffisamment d'informations à ce sujet, ce qui ne permet pas de tirer des conclusions sur cet aspect. Toutefois, des approches telles que A9 et A3 indiquent clairement que le Framework de modélisation supportent des possibilités d'extensions.

2.5 Conclusion

Dans ce chapitre, nous avons effectué une étude comparative des approches d'ingénierie dirigées par les modèles pour la conception des RdC. Cela nous a permis de tirer les conclusions suivantes :

- De nombreuses approches proposent leur propre langage de modélisation pour représenter les réseaux de capteurs. Ceci est principalement dû à leur grande diversité en termes d'analyses supportées, de niveau d'abstraction et des objectifs visés. L'absence d'un langage générique satisfaisant pour la modélisation peut être une autre raison. Un tel langage générique nécessitera toutefois des mécanismes d'extension permettant d'adapter le langage aux préoccupations spécifiques des diverses type d'applications ;
- Les approches étudiées sont construites sur un seul langage de modélisation comprenant tous les concepts liés au RdC. Ces approches ne sont donc pas conformes au principe de la séparation des préoccupations ;
- Certaines étapes du cycle de vie (par exemple, l'étape de dimensionnement, d'analyse et de simulation et la configuration logicielle) sont prises en compte. Cependant, il existe d'autres étapes moins traitées comme la phase du déploiement physique et la phase d'exploitation de données.

Comme conclusion de cette étude, nous soulignons l'absence d'un environnement de modélisation générique et extensible qui prend en compte tous les aspects du cycle de vie des réseaux de capteurs (développement, déploiement, et exploitation du réseau).

Afin de surmonter ces limitations, nous visons à proposer une nouvelle méthodologie basée IDM. Elle fera l'objet de notre prochain chapitre.

Chapitre 3

Vue d'ensemble de la méthodologie et de l'approche proposées

« *The greatest ideas are the simplest.* »

William Golding

Sommaire

3.1 Introduction

Comme illustré dans la Figure 3.1, la conception et la mise en œuvre d'un RdC comporte plusieurs étapes au cours desquelles des connaissances de différents domaines sont nécessaires. Ainsi, le développement d'un RdC nécessite la maîtrise de savoirs et de savoir-faire dans des domaines divers comme l'électronique, l'informatique, les radiocommunications et réseaux. Ce développement implique en plus différents acteurs de divers métiers en lien avec les besoins applicatifs [130]. L'exploitation des données, par exemple, peut être réalisée par des experts de domaines non techniques, comme des médecins dans le domaine de la santé.

Tout au long de ces étapes, plusieurs outils/applications de domaines spécifiques sont utilisés. On retrouve ainsi des outils pour la conception du matériel, le codage du logiciel, les spécifications et les simulations des réseaux et protocoles de communication, la simulation de l'environnement et l'évaluation de son impact sur le RdC.

Les outils interviennent à différents niveaux d'abstraction et utilisent des sémantiques et des langages différents. Ils sont généralement indépendants et rarement connectés les uns aux autres. Ainsi, il est nécessaire de décrire les caractéristiques du RdC manuellement dans chacun de ces outils en utilisant les langages qui leur sont propres.

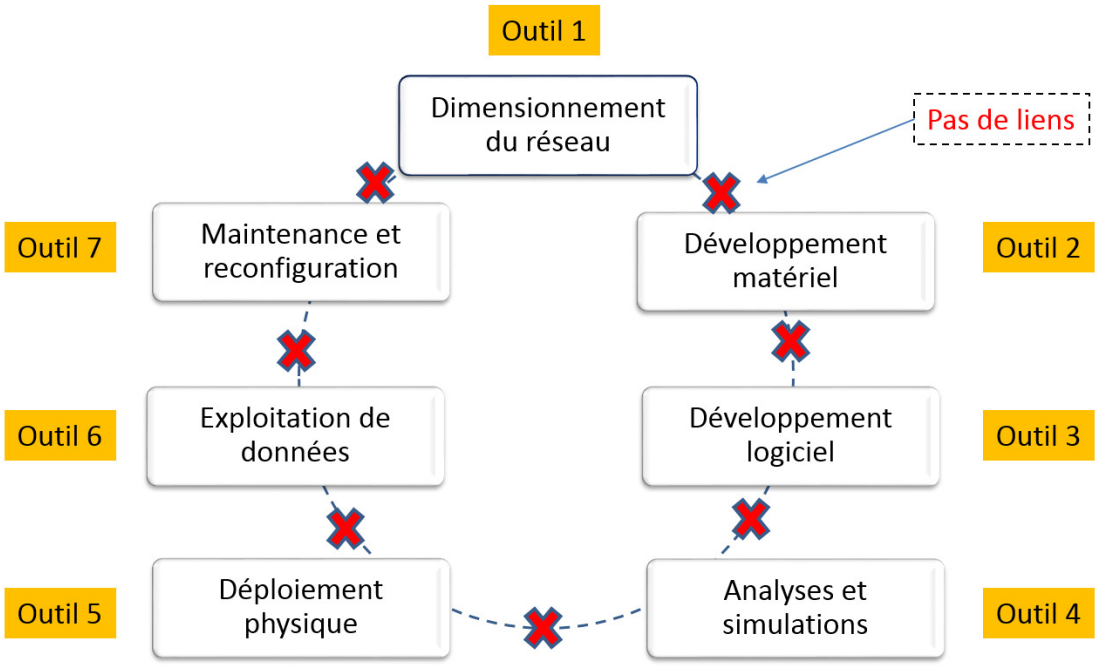


FIGURE 3.1 – Problématiques dans la conception des réseaux de capteurs.

Lorsque le nombre de nœuds du RdC augmente, le temps passé à ces descriptions devient très important. Une modification du réseau doit le plus souvent être répercutée dans chaque outil. Il est très difficile d'assurer la cohérence entre les différentes descriptions. Ce cloisonnement conduit à des erreurs de conception, à de mauvaises interprétations et

éventuellement à des dysfonctionnements.

Un certain nombre d'approches de modélisation des réseaux de capteurs ont été proposées. Comme nous l'avons souligné dans le chapitre précédent, ces approches ne permettent pas de relever tous les défis.

Afin de surmonter ces limitations, nous proposons :

1. Une structuration de l'ensemble des étapes dans un cycle de vie ;
2. Une nouvelle méthodologie qui vise à couvrir ce cycle proposé.

3.2 Cycle de vie d'un réseau de capteurs

La conception des réseaux de capteurs implique différentes étapes : du dimensionnement à l'exploitation de données. Nous proposons de regrouper ces étapes dans un cycle de vie. Il peut être représenté par trois sous-cycles, à savoir le cycle de développement, le cycle de déploiement et le cycle d'exploitation du réseau (Figure 3.2).

3.2.1 Cycle de développement

Dans le cycle de développement, il est nécessaire de définir les objectifs et les exigences, le dimensionnement du réseau et ensuite l'identification des éléments du réseau.

Dans un premier temps, les spécifications du réseau sont identifiées comme les objectifs, le type d'application (suivi, collecte de données), la nature de l'environnement physique du déploiement (indor, outdoor), l'échelle du réseau (petite échelle, grande échelle), les contraintes de l'application (le temps de latence max, débit de données, sensibilité des données, la durée de vie du réseau à envisager et le budget financier).

Le dimensionnement du réseau consiste à fixer une architecture du réseau en définissant le nombre et les positions des nœuds. Des choix d'architecture matérielle et logicielle sont ensuite effectués à chaque nœud. Par exemple, l'architecture matérielle est liée au choix des technologies de communication. Une étape d'analyse et de simulation permet de mesurer les métriques de performance comme la consommation d'énergie, les pertes de paquets, et le temps de latence. Une analyse des résultats obtenus permet de valider les choix déjà formulés à l'étape précédente si les contraintes de l'application sont respectées ou bien de réitérer les analyses avec d'autres paramètres.

Après la validation du dimensionnement du système, le concepteur du réseau vérifie la disponibilité des éléments du réseau dans un marché pour les acheter. Dans le cas contraire, une phase de prototypage de nouveaux composants est nécessaire. Cette phase est réalisée par des schémas électroniques, la conception matérielle, la spécification logicielle et des tests pour la validation. Il convient de noter qu'il n'y a pas qu'une seule séquence possible d'étapes dans ce cycle.

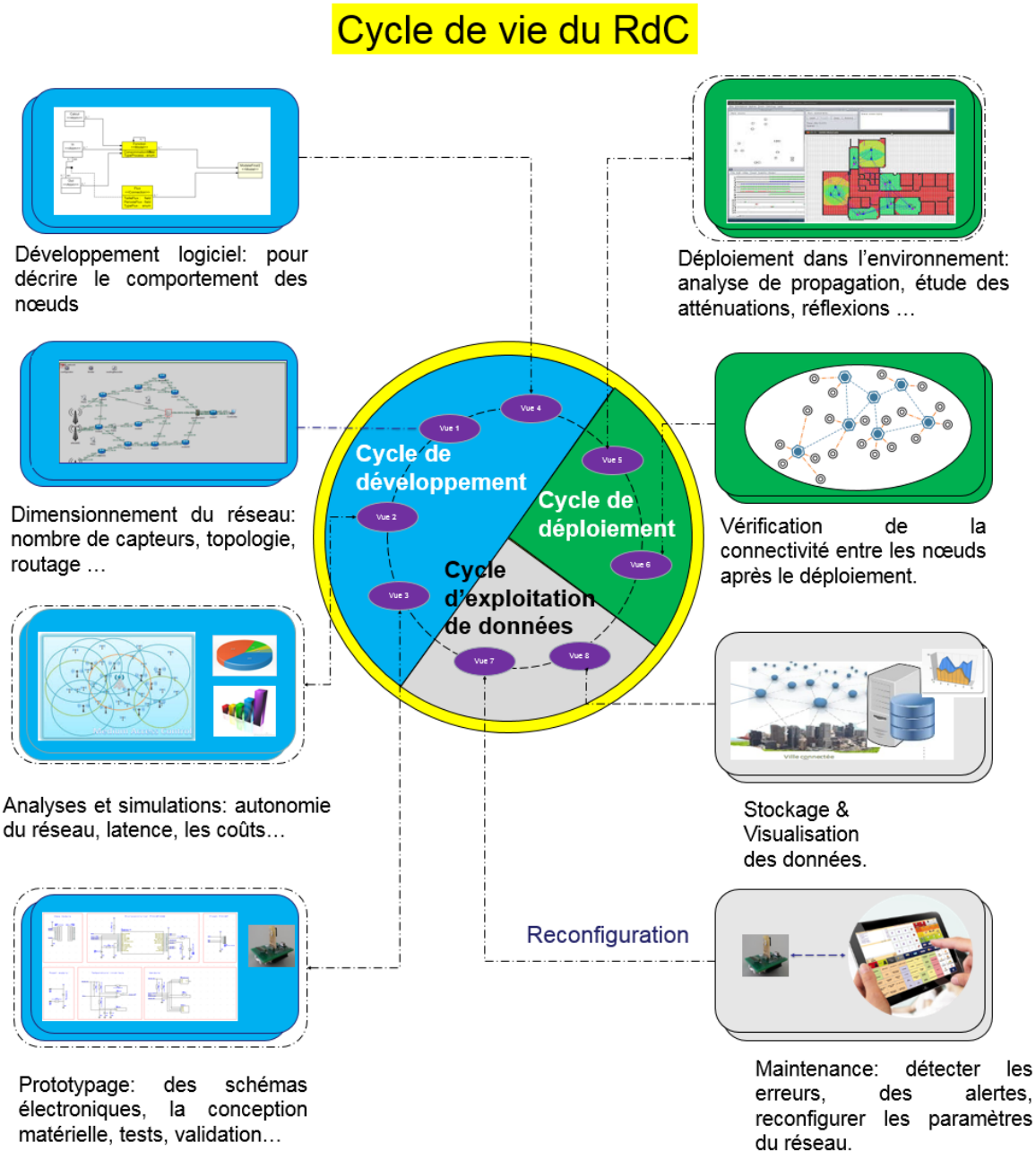


FIGURE 3.2 – Cycle de vie d'un réseau de capteurs.

3.2.2 Cycle de déploiement

Le déploiement physique signifie que chaque nœud sera affecté à un emplacement physique. Les nœuds de capteurs peuvent être déployés un par un dans l'environnement physique ou aléatoirement en utilisant un avion par exemple. Dans le cas d'un réseau à grande échelle, le nombre total de capteurs et leur environnement de déploiement rendent cette phase difficile.

La défaillance du nœud est un événement probable en raison des caractéristiques spécifiques du réseau de capteurs. Il est également possible d'avoir des réseaux de capteurs avec des nœuds hautement mobiles. Par conséquent, des tests permettent de vérifier les fonctionnalités du réseau dans la phase de déploiement.

Des nœuds de capteurs supplémentaires peuvent être redéployés à tout moment pour remplacer les nœuds défaillants. L'ajout de nouveaux nœuds rend nécessaire la réorganisation du réseau et la vérification de ses performances.

3.2.3 Cycle d'exploitation du réseau

Dans cette phase, l'utilisateur peut faire un traitement de données reçues des différents capteurs et peut également configurer son réseau.

L'utilisateur peut visualiser les données des capteurs sous forme de courbes ou de diagrammes. Il peut déterminer des informations telles que le minimum et le maximum ou bien développer des algorithmes de traitement plus avancés.

Un autre aspect à considérer est la maintenance du réseau. Ainsi en cas de constat d'un dysfonctionnement du réseau, il est nécessaire d'en déterminer d'abord la cause et procéder par la suite à sa reconfiguration. Par exemple, si un nœud consomme trop d'énergie, il est possible d'adapter ses paramètres pour réduire sa consommation en augmentant sa période de transmission des données.

3.3 Objectifs de la nouvelle méthodologie

Les approches proposées actuellement ne répondent pas à l'ensemble des difficultés rencontrées lors de la conception des réseaux de capteurs. Ainsi, nous visons :

- ✓ La couverture complète du cycle de vie d'un RdC ;
- ✓ La possibilité de décrire différentes architectures sans se limiter à des architectures particulières ;
- ✓ La minimisation du temps de développement en permettant des analyses et des simulations de modèles à l'aide des outils existants.

3.3.1 Couvrir le cycle de vie

La synthèse de l'état de l'art sur les approches existantes montre que les solutions proposées se limitent toutes à une partie du cycle de vie. Nous cherchons donc à proposer une méthodologie de conception qui s'intéresse à l'ensemble du cycle de vie. Ceci permet une optimisation croisée de toutes les étapes du cycle de vie.

Chaque étape du cycle de vie nécessitera d'utiliser des outils/modèles adaptés à sa problématique. Si on est capable de transformer les modèles manipulés dans chaque étape, il sera possible de relier automatiquement les outils sur l'ensemble du cycle de vie pour éviter de redécrire des modèles existants. Le passage d'une étape du cycle de vie à une autre pourra être automatique. On ne perdra donc plus de temps à répercuter manuellement des modifications et on réduit les erreurs manuelles de traduction ou d'interprétation lors du passage d'un outil à un autre. Ceci permet notamment d'assurer la cohérence.

Dans chaque étape, il est ainsi possible d'utiliser l'outil le mieux adapté sans avoir à réécrire à chaque fois le modèle du réseau de capteur avec le langage spécifique proposé par chaque outil. Une modification faite dans un outil utilisé lors d'une étape du cycle de vie peut être répercutée automatiquement dans l'ensemble des outils utilisés dans le cycle de vie du réseau de capteur.

3.3.2 Développer des modèles ouverts et génériques

La plupart des travaux proposés ont été conçus pour prendre en compte des architectures spécifiques. Ceci entraîne des problèmes de flexibilité et de réutilisation. Par rapport à notre travail, nous cherchons donc à proposer une méthodologie permettant de manipuler des modèles génériques et ouverts.

Les modèles génériques permettent de couvrir plusieurs architectures possibles. Les modèles ouverts permettent de les étendre facilement pour tenir compte des spécificités. Ces derniers favorisent la réutilisation des outils existants tout au long du cycle de vie. De plus, l'utilisation des modèles ouverts génériques doit permettre de prendre en compte des réseaux hétérogènes composés de différentes technologies de communication et de différentes architectures matérielles et logicielles. Il est ensuite possible de combiner ces architectures (matérielles et logicielles) pour aboutir à une solution plus adaptée par rapport au problème abordé. A partir de modèles génériques ouverts, il est possible d'obtenir plus rapidement des modèles spécifiques en partant d'une base qui pourra être adaptée. Ceci permet de réduire le temps et les coûts de développement.

3.3.3 Analyser/simuler les modèles

Dans nos travaux, nous cherchons aussi à proposer des améliorations sur les méthodes d'analyse des réseaux de capteurs. Les approches actuelles permettent des ana-

lyses et des optimisations localisées d'un problème. En proposant une approche qui fournit des liens entre les outils et les étapes du cycle de vie d'un RdC, il est possible d'envisager des analyses et des optimisations plus fines ou globales en combinant les informations de chaque modèle. Par exemple, sur la base de ces modèles, une estimation de l'autonomie énergétique des nœuds du réseau pourrait être réalisée en recoupant la modélisation du logiciel, le modèle de l'environnement de communication, le modèle de l'architecture matérielle et les médias de communications utilisés. La consommation énergétique du nœud dépend en effet du volume et de la fréquence de communication des données, de la consommation énergétique des composants, de la puissance d'émission et réception nécessaire dans un environnement donnée, du protocole de communication utilisé ainsi que de la capacité de la source d'énergie.

A travers ces analyses, la méthodologie offre une aide à la conception des réseaux de capteurs. En particulier, elle permet au concepteur de mieux choisir les éléments matériels de son réseau, le dimensionnement du réseau, les technologies de communication et la topologie. Il est possible d'analyser l'impact de certains choix sur le coût global de l'application, voire de définir des algorithmes qui dimensionneront le réseau de capteurs afin d'optimiser le coût pour une qualité de service donnée.

3.4 Approche de la méthodologie proposée

L'analyse de l'état de l'art nous a amené à proposer une méthodologie basée sur une conception dirigée par les modèles Ingénierie Dirigée par les Modèles (IDM). Elle vise à couvrir toutes les phases du cycle de vie d'un RdC en formalisant l'ensemble des concepts et des caractéristiques d'un RdC à l'aide d'un *métamodèle* commun (voir Figure 3.3). A partir de ce référentiel, il est possible d'extraire, à chaque étape du cycle de vie du RdC, une vue partielle (un modèle métier) où seules les caractéristiques concernées par la problématique appréhendée sont présentes. Cette vue, appelée facette, est alors présentée et éditée avec l'outil/langage métier le mieux adapté au domaine concerné.

Cette séparation claire des préoccupations donne lieu à une démarche de conception *multi-facettes* couvrant l'ensemble des étapes de conception d'un RdC. Ces facettes peuvent être manipulées avec des outils tiers reconnus, moyennant des passerelles vers ces outils et des règles de *transformations M2M (model-to-model)*. Ces liens automatiques avec les outils reconnus permettent donc une exportation des modèles d'un RdC tout au long de ses étapes de conception sans les redécrire dans chacun de ces outils. Ces passerelles permettent aussi d'importer et d'intégrer les résultats d'analyse/simulation réalisés dans ces outils. Ceci favorise des analyses/optimisations globales des caractéristiques du RdC et une meilleure cohérence des choix de conception (architecture, topologie réseau, technologie de communication) avant le déploiement physique du RdC.

Afin d'accélérer ce déploiement et faciliter les phases d'exploitation et de maintenance d'un RdC, nous proposons, à l'aide de *transformations M2T (Model-to-Text)*, de

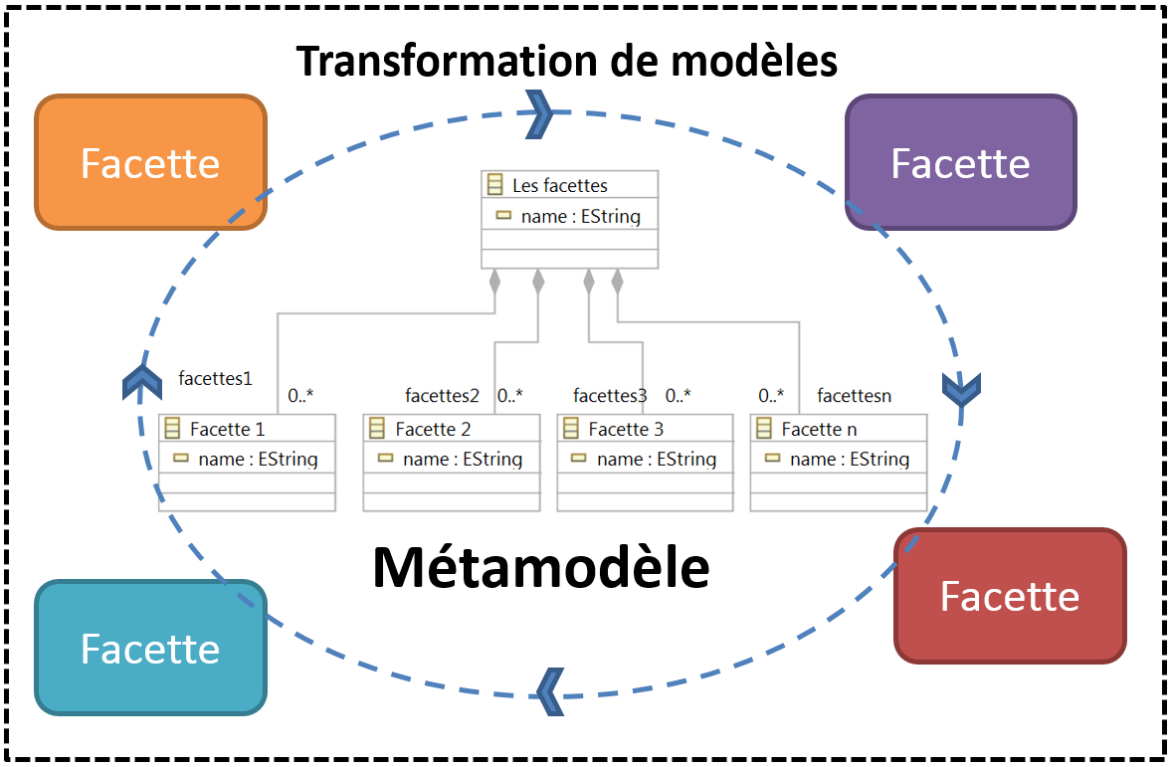


FIGURE 3.3 – Vue d'ensemble de la méthodologie proposée pour la conception des réseaux de capteurs.

générer du code pour l'ensemble des nœuds d'un RdC (des capteurs jusqu'au point de collecte, bases de données et les interfaces Homme-Machine d'exploitation de données). La Figure 3.3 résume cette méthodologie en se concentrant sur trois points principaux :

- ✓ Utilisation des métamodèles ;
- ✓ Démarche multi-facettes ;
- ✓ Transformations de modèles.

3.4.1 Approche basée métamodèle

Notre métamodèle est basé sur le diagramme de classe UML et les contraintes OCL. Les classes UML permettent d'obtenir un modèle flexible et extensible en se basant sur les notions d'héritage. Il permet donc de représenter tout élément ou fonction figurant dans un RdC. De plus, le langage UML est un langage standard compréhensible par d'autres outils (basé sur XML Metadata Interchange (XMI)). Il est possible de ce fait d'importer/exporter des modèles entre les outils. A travers UML, on peut définir des éléments génériques basiques pour pouvoir par la suite construire des modèles complexes.

Dans le métamodèle, nous séparons les différents concepts présents dans le réseau de capteurs comme le matériel, le logiciel et l'environnement physique. Cette séparation rend les modèles réutilisables à travers des projets et des organisations différentes. Elle

permet également d'avoir une certaine flexibilité pour décrire les différents types d'architectures logicielles et matérielles des nœuds.

Chaque modèle contient des propriétés qui servent à configurer et à analyser le modèle. Ces propriétés permettent aussi de valider les exigences fonctionnelles et non-fonctionnelles pendant la phase de conception et de simulation. La mise en œuvre de cette séparation est illustrée dans un environnement à multi-facettes qui sera abordé dans la prochaine section.

3.4.2 Environnement multi-facettes

Nous avons choisi une approche de modélisation multi-facettes. Chaque facette permet d'extraire une vue du système [131, 132]. La modélisation multi-facettes permet à un développeur de décrire un système à partir de plusieurs points de vue structurels et comportementaux et en utilisant différents langages de modélisation.

Nous définissons une facette comme une partie du système décrite avec un langage adapté et qui peut éventuellement permettre d'analyser ou de simuler un réseau de capteurs. La facette peut également être définie comme une intersection avec un sous ensemble d'un modèle complet en utilisant un langage spécifique orienté métier. Cette structure permet de réduire la complexité du système en se concentrant sur ses différentes parties séparément.

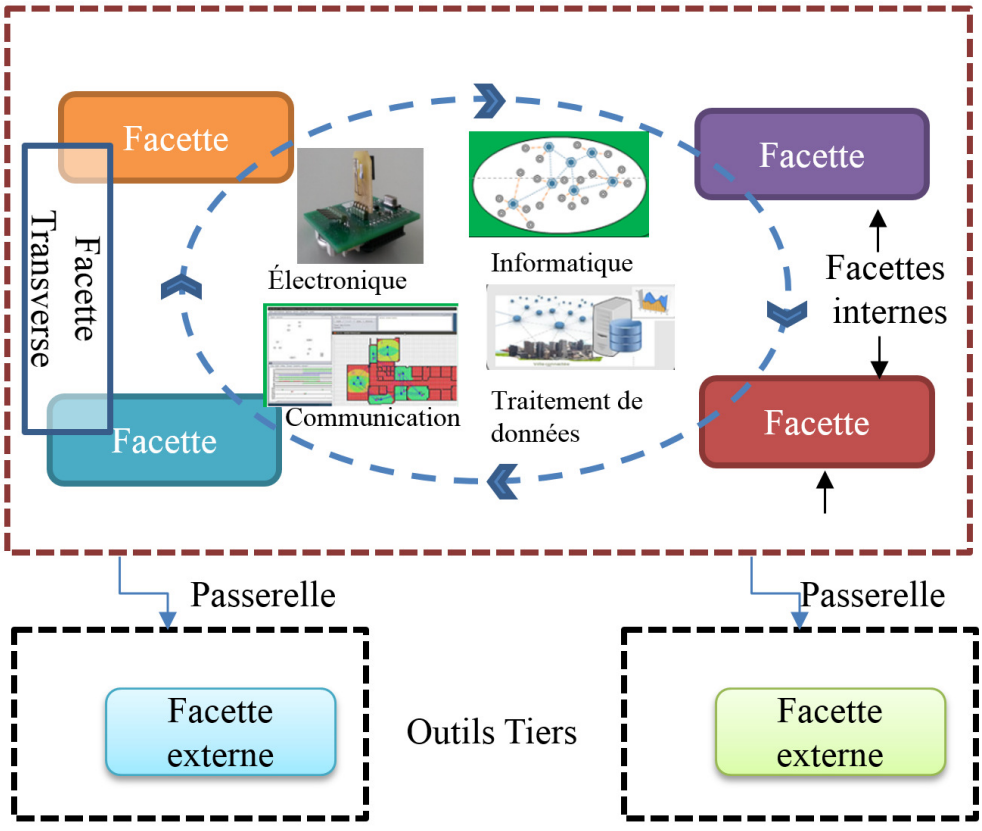


FIGURE 3.4 – Approche de modélisation multi-facettes : internes, externes et transverses.

La conception et la mise en œuvre des réseaux de capteurs nécessitent des compétences dans différents domaines dont l'électronique, l'informatique, le réseau, la communication et le traitement de données. Les facettes fournissent des interfaces de modélisation de tous les domaines liés aux réseaux de capteurs. Les facettes sont formalisées à l'aide d'un métamodèle implémenté dans un environnement de modélisation de base. Par rapport à cet environnement, on distingue des facettes internes et des facettes externes comme illustré dans la Figure 3.4.

Les facettes internes fournissent une interface spécifique dans notre environnement de modélisation. Elles sont développées avec l'outil de modélisation de base qui est utilisé pour décrire le métamodèle. Il est possible d'effectuer différentes analyses et procéder à plusieurs simulations sur chaque facette.

Les facettes externes sont développées avec des outils tiers moyennant des passerelles vers ces outils. Par exemple, pour estimer les performances d'un RdC comme la consommation et les coûts, nous mettons en place des passerelles entre les simulateurs et notre environnement de modélisation.

De plus, ces facettes peuvent être locales, propres à un domaine ou des *facettes transverses* à plusieurs domaines. Les facettes transverses sont utilisées pour représenter une propriété qui s'étale sur plusieurs facettes du système ou une fonctionnalité dite orthogonale sur une structure. Par exemple, la propriété de consommation est liée à l'aspect matériel, logiciel et les configurations et protocoles réseau.

3.4.3 Transformations de modèles

Nous envisageons d'utiliser des transformations de modèles pour analyser certains aspects des modèles et synthétiser ensuite divers types de sorties comme le code source, les entrées de simulation ou d'autres représentations de modèles. Dans cette approche, nous distinguons les transformations M2M et M2T.

Les transformations de modèles M2M permettent d'assurer la cohérence entre les modèles et les facettes tout au long du cycle de vie du réseau de capteurs, vu que ces transformations sont automatiques et basées sur des règles bien définies. Elles réduisent aussi le temps de développement. Dans notre méthodologie, il y a deux types de transformations de modèles : (1) les transformations automatiques internes à l'outil de modélisation en se basant sur des règles de transformation entre les métamodèles, (2) les transformations externes vers des outils externes pour créer des passerelles entre notre environnement de modélisation et ces outils.

Les transformations M2T sont développées à partir de squelette de code/bibliothèques spécifiques aux technologies des modèles. Les templates doivent être développés en fonction d'une plate-forme cible. Les transformations M2T permet la génération du code à implémenter dans les nœuds ainsi que la documentation afin de faciliter la compréhension et la manipulation de l'environnement de modélisation.

Après cette vue globale de notre méthodologie, nous détaillons dans les prochaines sections les facettes identifies.

3.5 Identification des facettes

Les réseaux de capteurs couvrent plusieurs domaines tels que l'électronique, l'informatique, les télécommunications et le traitement des données. A partir de ces domaines, cinq facettes sont identifiées : la facette réseau, la facette matérielle, la facette logicielle, la facette flux de données et la facette d'environnement physique (voir Figure 3.5). Ci-dessous, nous présentons une brève description de chaque facette.

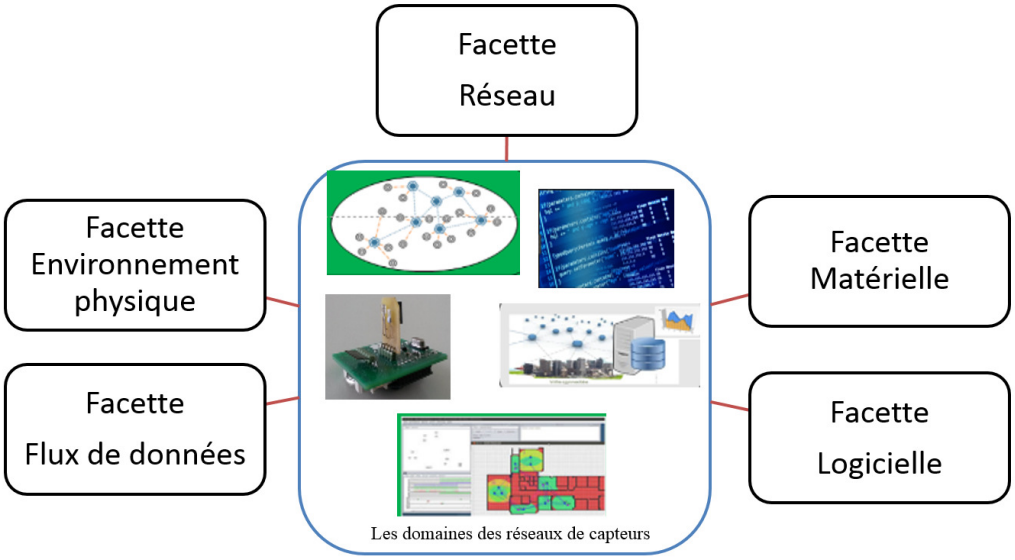


FIGURE 3.5 – Facettes identifiées dans la méthodologie proposée.

3.5.1 Facette réseau

La facette réseau définit l'architecture du réseau de capteurs et ses éléments. Elle décrit la connectivité entre les éléments du réseau. Elle permet également d'associer des adresses aux éléments du réseau.

Dans ce contexte, des acteurs peuvent raisonner sur la topologie la plus adaptée au problème. Ils peuvent analyser les protocoles MAC pour l'accès au canal et éviter les problèmes de collision et de perte de données. Des protocoles de routage peuvent également être investis pour améliorer les performances du réseau de capteurs, par exemple pour réduire la consommation d'énergie ou le temps de latence entre un nœud capteur et la station de base. Cette facette peut être utilisée pour lancer des vérifications de la connexion de bout en bout.

3.5.2 Facette matérielle

Cette facette est utilisée pour décrire les détails matériels de chaque type de nœud pouvant être utilisé dans un réseau de capteurs. En effet, différentes applications peuvent réutiliser les mêmes modèles matériels ou les paramétrer différemment en fonction des besoins de l'application. La facette matérielle contient les spécifications matérielles de bas niveau des nœuds, y compris les unités d'acquisition, de communication et d'alimentations. Dans chaque unité, on spécifie des informations pour la génération de code et aussi pour les analyses et les simulations.

3.5.3 Facette logicielle

Cette facette décrit le comportement d'un nœud indépendamment du matériel. Tous les nœuds ayant le même comportement partagent le même modèle de logiciel.

A travers cette facette, nous pouvons spécifier le comportement d'un nœud à l'aide d'un formalise adapté. Par exemple, nous pouvons décrire le comportement sous forme de différents modes. Pour le noeud capteur, cela se traduit par des modes d'acquisition, de transmission, de réception et de veille, en incluant des informations temporelles par exemple.

Cette facette est utilisée dans le processus de simulation/analyse et dans la génération de code. Elle est complétée par la facette flux de données.

3.5.4 Facette flux de données

La facette flux de données est utilisée pour décrire les échanges de données entre les nœuds. Dans cette facette, nous représentons tous les nœuds qui ont le même flux de données par une seule instance. Cela permet de s'adresser à des systèmes de grandes échelles parce que nous nous intéressons uniquement aux types de nœuds. Par exemple, tous les nœuds de capteurs qui produisent les mêmes types de mesures peuvent être décrits par un seul élément.

3.5.5 Facette d'environnement physique

La facette d'environnement physique concerne le site réel où le RdC sera déployé. Elle peut être utilisée pour localiser et placer les nœuds dans l'environnement. Suite à cet emplacement, l'utilisateur peut corréler les données reçues comme la température avec l'environnement associé (par exemple une chambre). Elle permet d'étudier la fiabilité des liaisons de communication dans le réseau et aussi les différents phénomènes liés aux communications entre les nœuds comme les atténuations et les réflexions des signaux par des obstacles. Cette facette peut être particulièrement utile pour les développeurs et les ingénieurs systèmes lorsqu'ils doivent raisonner sur la topologie du réseau et la présence d'obstacles physiques dans la zone de déploiement du réseau.

3.5.6 Facette de projection

Cette facette regroupe des informations transverses aux différentes facettes. Elle est utilisée pour créer des liens entre les éléments des différentes facettes comme illustré dans la Figure 3.6.

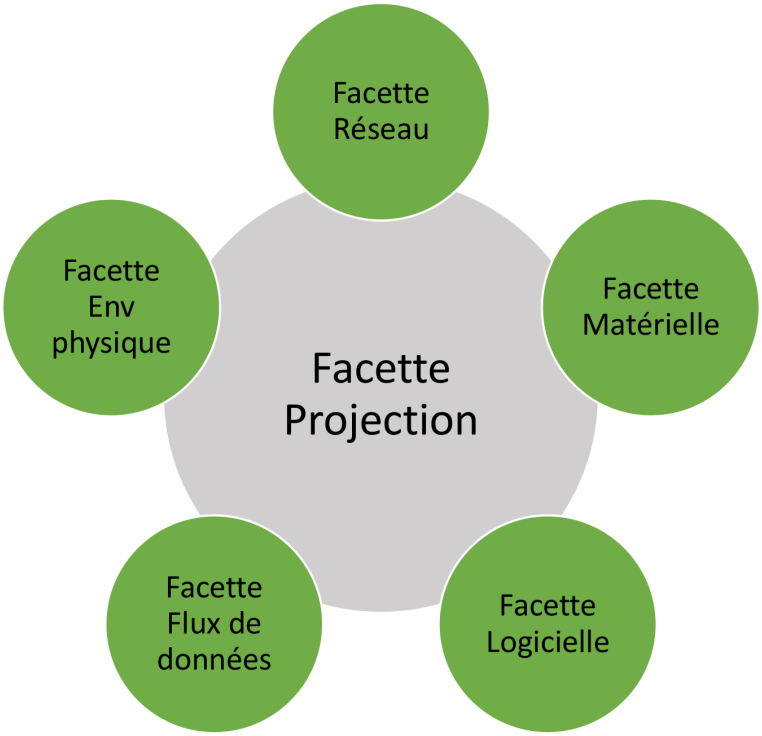


FIGURE 3.6 – La facette projection.

Dans notre méthodologie, nous pouvons avoir plusieurs projections où chacune est constituée d'une instance de chacune des cinq facettes comme une combinaison possible d'une solution. A travers plusieurs projections, nous pouvons configurer ou représenter plusieurs réseaux de capteurs avec différentes combinaisons possibles. Le choix entre ces solutions est basé sur des analyses et des simulations.

La Figure 3.7 montre un exemple de plusieurs projections : A, B et C. Chaque projection est composée de cinq facettes différentes. Les projections A et B partagent : la facette environnement physique 1, la facette flux de données 1, la facette logicielle 1 et la facette réseau 1. Cependant, elles se distinguent par la facette matérielle 1 et 2. Quant à la projection C, elle combine cinq autres facettes.

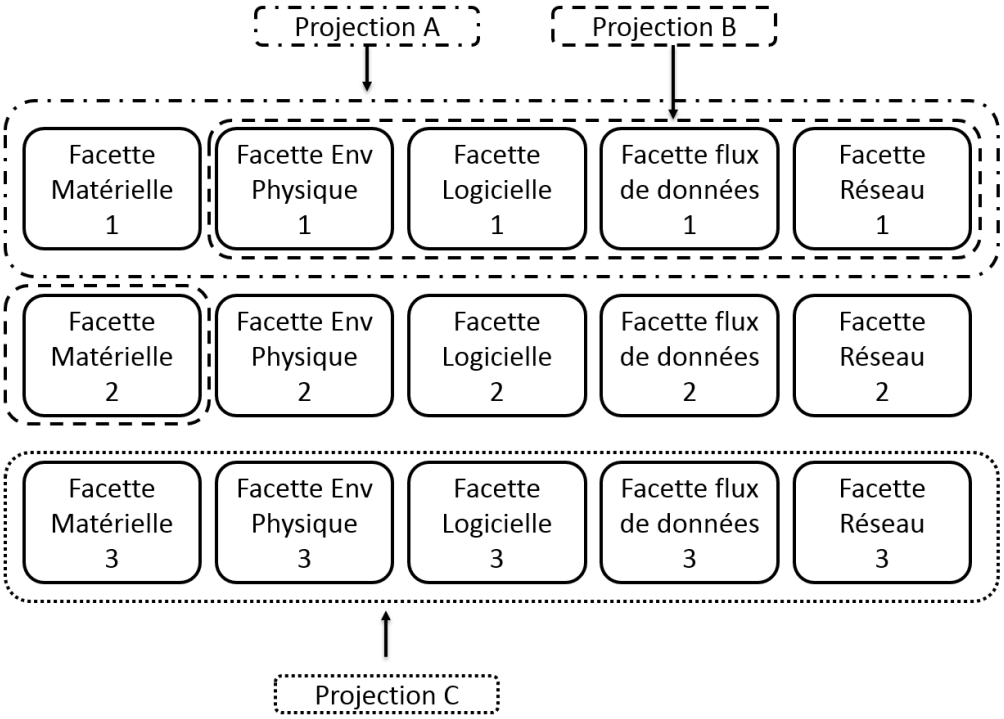


FIGURE 3.7 – Liens entre les facettes pour configurer différentes projections.

3.6 Transformations de facettes

Notre approche supporte des transformations de modèles. Une transformation se compose d'un script dans un langage bien défini qui possède en entrée une instance d'un modèle (et son métamodèle associé) et en sortie un autre modèle. En utilisant les transformations, les facettes sont analysées, combinées et traduites en :

1. Scripts de simulation ;
2. Code source.

3.6.1 Simulations et analyses

Notre approche vise à supporter des analyses et des simulations. Pour cette raison, nous utilisons des transformations de modèles pour développer des passerelles avec des simulateurs comme illustré dans la Figure 3.8. Ces transformations peuvent être de modèle vers modèle de telle sorte que nous prenons les modèles des facettes en entrée et nous les transformons en modèles de simulateurs cibles. Certains simulateurs utilisent des scripts de simulation en entrée, nous pouvons aussi utiliser des transformations de modèle vers texte de sorte que les modèles de facettes soient transformés en scripts pour les simulations.

Les transformations sont utilisées pour faciliter les processus d'analyse et de simulation et éviter les erreurs de conception. Elles peuvent également réduire le temps de

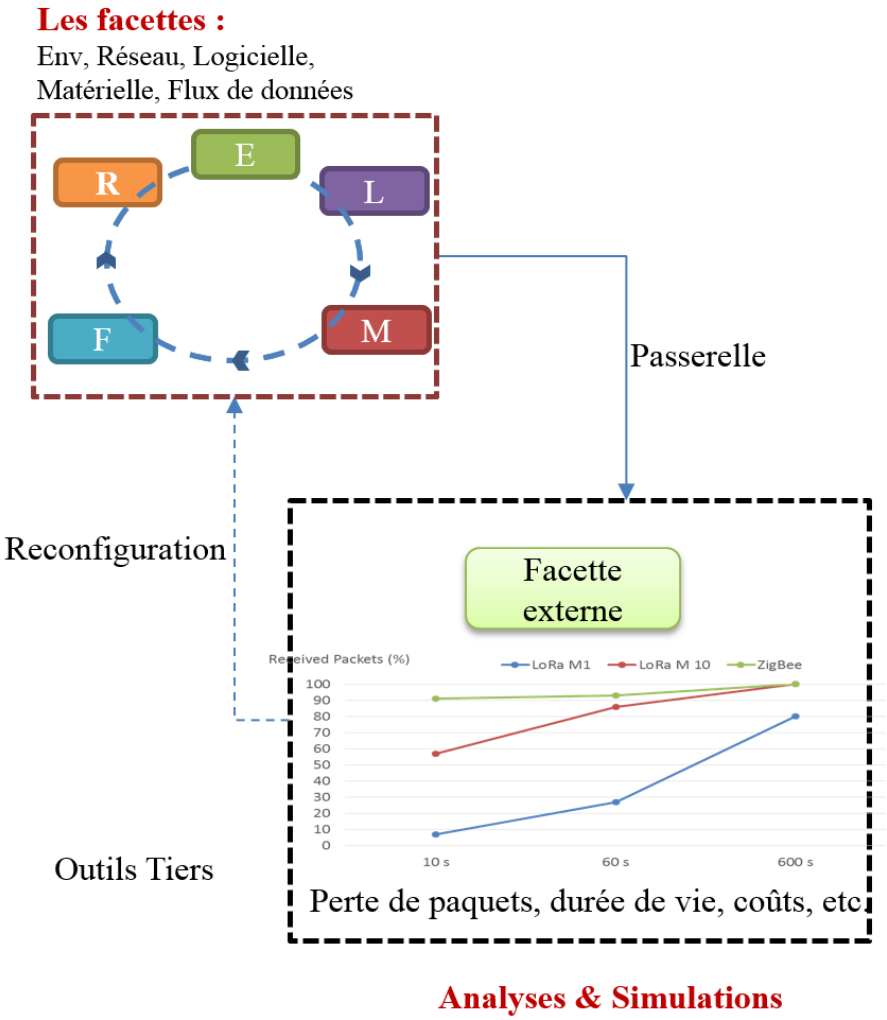


FIGURE 3.8 – Processus de simulation dans notre méthodologie.

développement puisque les transformations sont automatiques.

3.6.2 Génération de code

Les techniques de génération de code permettent à partir de la transformation de modèles d'obtenir automatiquement du code embarqué. Nous proposons d'utiliser aussi souvent que possible des techniques de génération de code tout au long du cycle de vie du réseau de capteur.

En général, le développement logiciel d'applications se produit presque entièrement au niveau matériel. Cette façon de faire engendre plusieurs problèmes dont une incapacité à tirer profit du cycle de vie du développement logiciel et la réduction des chances de détection précoce des erreurs. Elle entraîne également le non-respect des exigences de conception et d'amélioration des performances du réseau de capteurs avant le déploiement du code. L'implémentation qui comprend des phases de développement et de dé-

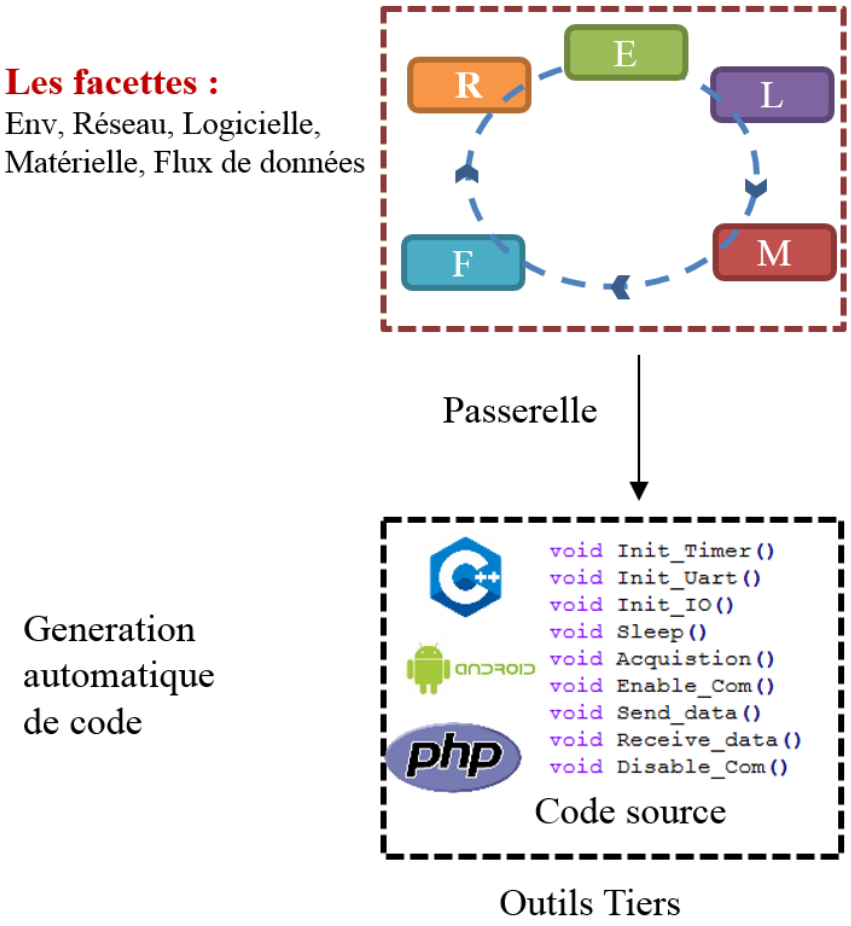


FIGURE 3.9 – Passerelle de génération de code.

bogage prend souvent beaucoup de temps. De plus, elle est spécifique à une plate-forme qui est sujette aux erreurs et est développée lors de la dernière étape de la conception.

Nous cherchons donc à générer du code automatiquement sur les nœuds qui composent les réseaux de capteurs. Ce code est généré à partir des descriptions d'un haut niveau d'abstraction qui sont indépendantes de la technologie utilisée. La Figure 3.9 illustre la passerelle de génération automatique du code source à partir des facettes.

Dans notre méthodologie, les codes sources sont générés automatiquement en combinant les informations modélisées dans les facettes. La Figure 3.10 illustre le processus de génération de code dans notre méthodologie. Dans un premier temps, les modèles des facettes sont transformés en macrocode. Ainsi, le macrocode est combiné avec des bibliothèques et des fichiers de définition liés à un langage et à une plateforme spécifiques.

— Le macrocode est indépendant de la plateforme et il décrit le comportement et les fonctions exécutées par un nœud ;

— Les fichiers de bibliothèque décrivent des fonctions spécifiques et dépendantes des plateformes et dans un langage de programmation ;

— Les fichiers de définition ont pour but de faire des liens entre les noms de fonctions

dans le macrocode avec celles qui sont implémentées dans les bibliothèques.

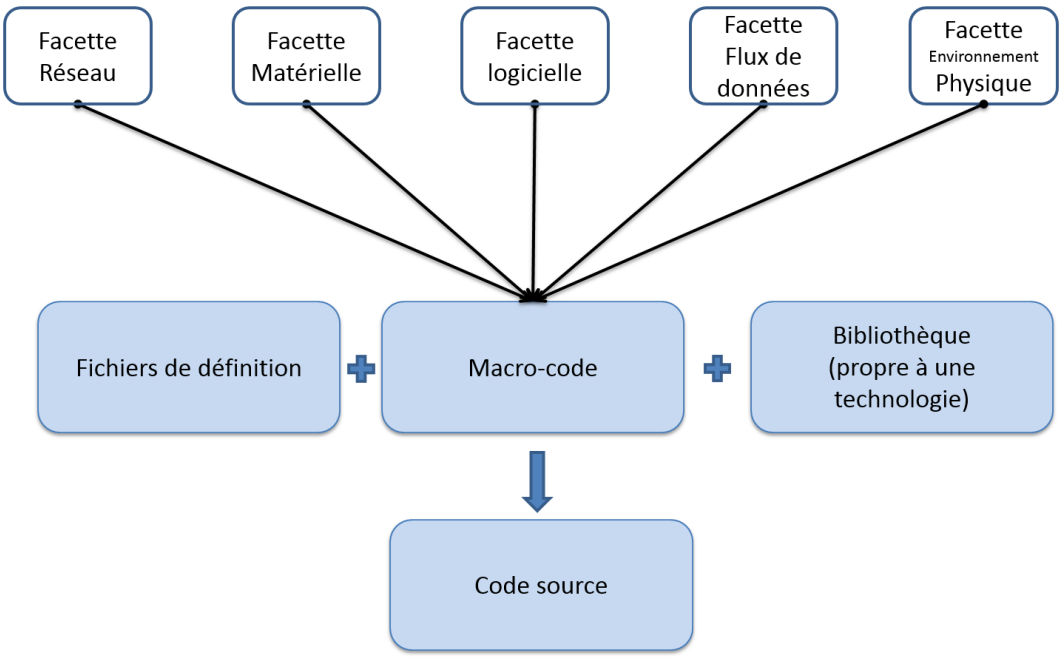


FIGURE 3.10 – Processus de génération de code avec le concept du macrocode.

3.7 Conclusion

La conception et la mise en œuvre des réseaux de capteurs sont confrontées à de nombreux défis liés au cycle de vie d'un RdC. Pour les surmonter, nous proposons une méthodologie basée sur une conception dirigée par les modèles. Elle vise à couvrir le cycle de vie d'un RdC en formalisant l'ensemble des concepts et des caractéristiques d'un RdC à l'aide des techniques de métamodélisation. Ainsi, il est possible d'extraire un ensemble de facettes qui donne lieu à une démarche de conception *multi-facettes* couvrant l'ensemble des étapes de conception d'un RdC. Nous avons identifié cinq facettes :

— La facette de l'architecture réseau qui se focalise sur la composition et la topologie du réseau ;

— La facette de l'architecture matérielle qui modélise les détails de bas niveau des différents types de nœuds ;

— La facette de l'architecture logicielle qui décrit le comportement d'un ensemble de nœuds ;

— La facette du flux de données qui se focalise sur les échanges de données entre les différents types de nœuds ;

— La facette de l'environnement physique qui spécifie l'environnement de déploiement du réseau.

Par le biais de transformations de modèles, ces facettes peuvent être liées. Dans cette méthodologie, nous pouvons aussi utiliser ces transformations pour pouvoir faire des analyses et des simulations en développant des passerelles vers des simulateurs existants. Á l'aide de transformations de modèle vers texte, nous utilisons les facettes modélisées afin de générer du code pour les différents nœuds du réseau. La génération automatique du code permet de réduire le temps de développement, d'éviter les erreurs de codage et de faciliter le développement des applications des RdC.

Nous présentons et discutons dans le prochain chapitre les modèles associes à cette méthodologie.

Chapitre 4

Facettes et transformations de modèles dans notre méthodologie

« *A good idea is about ten percent and implementation and hard work, and luck is 90 percent.* »

Guy Kawasaki

Sommaire

4.1 Introduction

Dans le troisième chapitre, nous avons présenté une vue d'ensemble d'une nouvelle méthodologie qui vise à couvrir l'ensemble du cycle de vie d'un RdC. Elle est basée sur les approches d'ingénierie dirigée par les modèles (IDM).

Ce chapitre fournit une description détaillée de cette méthodologie. Nous commençons dans la section 4.2 par une description des modèles de toutes les facettes et leurs métamodèles associés. Dans la section 4.3, nous abordons les transformations de modèles. Nous présentons en particulier les transformations entre les facettes et les transformations vers les simulateurs pour l'estimation des performances et les transformations de génération automatique de code.

4.2 Modèles des facettes

Notre méthodologie permet de développer un environnement de modélisation multi-facettes. Chaque facette est basée sur un langage de modélisations. Chaque langage permet à l'utilisateur de décrire un réseau de capteurs d'un point de vue particulièr (le réseau, le matériel, le logiciel, le flux de données et l'environnement physique). Il est important de souligner que les langages de modélisation ont été développés en vérifiant l'état de l'art en matière de modélisation de réseaux de capteurs comme nous l'avons souligné dans la section 2.4.

Dans les sections suivantes, nous formalisons la structure et les concepts de tous les langages de modélisation en définissant leurs modèles et métamodèles associés.

4.2.1 Modélisation du réseau

La facette réseau décrit l'architecture du réseau. Elle est composée de nœuds, de base de données, d'utilisateurs, d'infrastructure réseau externe et des liens de communications. Ces composants sont détaillés ci-après :

Nœud

Un RdC peut être composé de nœuds de différentes architectures (matérielle et logicielle), possédant des caractéristiques différentes. La notion de nœud permet de factoriser les caractéristiques communes d'un ensemble de nœuds d'une catégorie donnée (par exemple : capteur, passerelle, routeur et sink.). Chaque nœud possède des ports pour spécifier le type de la technologie de communication de manière à pouvoir représenter un réseau hétérogène. Chaque noeud est identifié par un *ID* et *PanId* spécifique à son réseau. Les couches MAC et réseau peuvent être décrites avec les attributs : MAC et *Routing*. Les coûts de la plateforme sont indiqués avec l'attribut *costs*. En utilisant le mécanisme d'instanciation, on peut définir des nœuds à partir d'une classe de nœuds. La Figure 4.1 illustre

un exemple de deux nœuds associés à des ports.

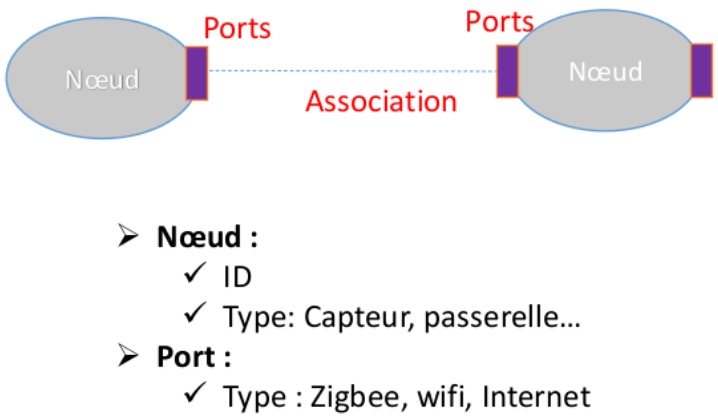


FIGURE 4.1 – Exemple d'association de deux nœuds dans le modèle réseau.

Base de données

Pour répondre aux besoins de stockage et de visualisation de données, une base de données est nécessaire dans un RdC. Deux types de base de données sont modélisées : une base de données locale et une base de données externe type cloud.

La base de données locale est entièrement configurable comme la base de données définie par SQL (Structured Query Language). Elle est caractérisée par des attributs de connexions comme le nom, le mot de passe et son adresse. Elle est composée de plusieurs tableaux (*Table*). Chaque tableau est identifié par ID. Il comporte des champs (*Field*) pour stocker différents types de données tels que la température ou l'humidité.

La base de données externe (cloud) est utilisée pour permettre l'interfaçage avec des plateformes de stockage existantes type IoT. Elle est caractérisée par des attributs de connexions (*name*, *pwd*, *address*) et un attribut (*Constraint*) pour la spécification des contraintes de taille de données et de cadence de stockage. Elle est composée de plusieurs canaux (*Channel*). Chaque canal, identifié par un *ID*, possède des champs (*Field*) pour stocker les différents types de données (température, humidité, etc). Un canal peut être privé ou public (accès libre). Un canal privé est protégé par deux clés de lecture et d'écriture (*WriteAPIKeys*, *ReadAPIKeys*).

Utilisateur

Dans un réseau de capteurs, on peut avoir un ou plusieurs utilisateurs. La modélisation de l'utilisateur permet de représenter les usagers du réseau et aussi de les notifier sous certaines conditions. Le modèle utilisateur possède deux attributs, dont le nom et

les paramètres de notification d'utilisateur. Par exemple, chaque utilisateur peut avoir un champ pour spécifier une adresse mail afin de recevoir des notifications.

Infrastructure réseau externe

L'infrastructure réseau externe représente un réseau externe tel que Internet. Elle possède un attribut *Setting* permettant de spécifier ses paramètres. Ce réseau est accessible via des liens qui peuvent nécessiter la présence d'un *proxy* décrit dans son port. Les informations spécifiées sont utiles en particulier dans le processus de génération de code.

Lien de communication

Les liens de communication permettent d'associer les éléments du réseau entre eux. Associer deux éléments signifie qu'ils peuvent communiquer avec le même média de communication. Chaque lien (*Association*) est identifié par son nom.

Dans la Figure 4.2, nous représentons un exemple simple d'un RdC composé de deux nœuds capteurs qui envoient des données à un nœud passerelle. Ce dernier nœud transmet des données vers une base de données pour le stockage à travers l'infrastructure réseau. Chaque entité de ce modèle possède des spécifications concernant son identifiant et son coût. Ces spécifications sont utilisées dans les transformations de modèles.

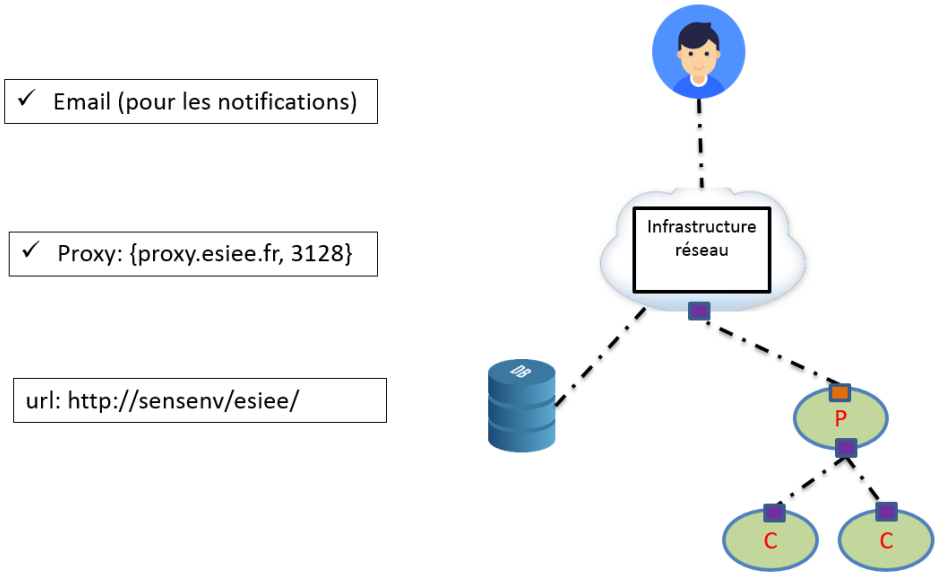


FIGURE 4.2 – Exemple d'un modèle de la facette réseau.

La Figure 4.3 représente les éléments du métamodèle de la facette réseau qui se compose de nœuds, d'une infrastructure réseau pour représenter un réseau externe, de deux types de bases de données (locale et externe) et d'un utilisateur. Les nœuds et le réseau externe possèdent des ports. Les port, les tables, les canaux et les utilisateurs héritent d'une classes *LinkedObjects* pour pouvoir interconnecter.

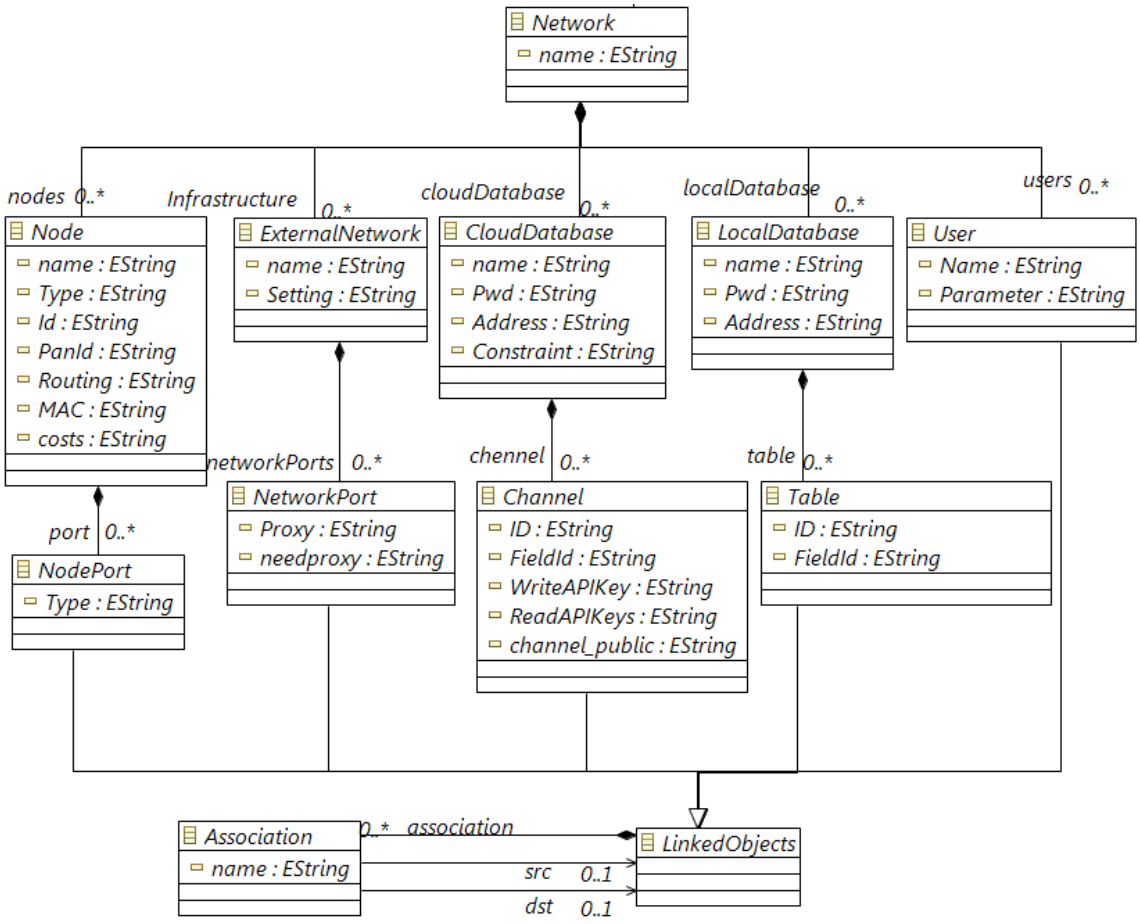


FIGURE 4.3 – Métamodèle de la facette réseau.

4.2.2 Modélisation de l'architecture matérielle

Les ressources matérielles utilisées dans une plate-forme matérielle sont modélisées sous la forme de plusieurs composants. La figure 4.4 illustre ces composants qui partagent des attributs communs tels que les noms et les coûts d'achat. Ces composants sont décrits plus en détail ci-dessous :

Capteur

Le nœud peut n'être équipé d'aucun ou bien de plusieurs capteurs. Chacun d'eux étant adapté pour acquérir des données spécifiques. Par exemple, un capteur peut détecter le niveau de la température, la présence de fumée ou de gaz ou la localisation géographique. Dans ce modèle, un capteur est caractérisé par un type de mesure (par exemple la température, l'humidité et la luminosité), une précision de mesure (*Accuracy*), une spécification du signal de sortie (analogique ou numérique), une consommation d'énergie qui représente la quantité d'énergie dont il a besoin pour effectuer une seule mesure et un temps pour réaliser la mesure.

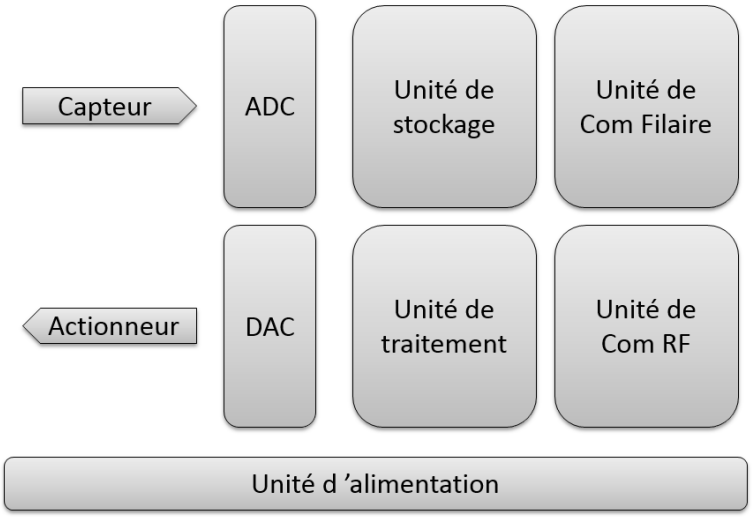


FIGURE 4.4 – Composants matériels d'un nœud dans un réseau de capteurs.

ADC

Analog-to-Digital Converter (ADC) sert à convertir un signal analogique en une valeur numérique. Pour l'ADC, nous pouvons spécifier son nom, ses canaux pouvant être connectés avec des capteurs analogiques et sa consommation énergétique.

Unité de traitement

Elle contrôle les activités du nœud depuis la détection jusqu'à l'envoi des données. Elle gère également l'énergie des nœuds et alterne entre les modes actif et en veille en fonction du comportement de l'application. Cette unité est caractérisée par un type (microcontrôleur, carte arduino, etc.), une fréquence de fonctionnement, un langage de programmation et sa consommation énergétique.

Unité de stockage

Elle permet de stocker les données reçues soit de façon permanente ou bien de façon temporaire pour une utilisation à court terme. Elle est caractérisée par une capacité de stockage en spécifiant les adresses de début et de fin de stockage.

Unité de communication filaire

Dans un RdC, nous pouvons avoir une unité de communication filaire comme c'est le cas dans un réseau de capteurs qui utilise le réseau Ethernet. Dans cette unité, on peut spécifier sa consommation, le débit de données, la taille et le type de la file d'attente.

Unité de communication RF

Le nœud peut utiliser un dispositif radio (Radio frequency (RF)) pour communiquer avec d'autres nœuds (zigbee, LoRa). Techniquement, un dispositif de communication RF représente un émetteur-récepteur qui peut fonctionner sur des bandes spécifiques, telles que la bande de 2,4 GHz. Un périphérique RF est caractérisé par sa puissance de transmission, sa sensibilité de réception, le gain de son antenne, la fréquence de fonctionnement, le débit de données, sa modulation et sa consommation d'énergie dans les modes de fonctionnement tels que les modes actif et en veille. Ces informations sont utilisées dans les simulations et dans la génération de code.

Actionneur

Le noeud peut être équipé d'un actionneur pour agir sur l'environnement. L'actionneur est déclenché par l'unité de traitement. Un nœud peut n'être équipé d'aucun ou de plusieurs actionneurs. Parmi les exemples d'actionneurs, on peut citer des systèmes d'éclairage, de climatisation et des serrures de portes électroniques. L'actionneur est caractérisé par son type et sa consommation d'énergie.

DAC

Digital Analog Converter (DAC) convertit des valeurs numériques en signaux physiques continus. Pour le DAC, nous pouvons spécifier sa consommation d'énergie et ses canaux en liens avec les actionneurs.

Unité d'alimentation

L'unité d'alimentation représente tout équipement ou dispositif utilisé pour fournir de l'énergie électrique au nœud. Un nœud capteur peut être alimenté en énergie par trois types de sources : par batterie, par secteur ou par récupération d'énergies [133–135]. L'unité d'alimentation est caractérisée donc par le type de source d'énergie et sa capacité.

La Figure 4.5 représente notre métamodèle qui comporte un ensemble d'unités matérielles comme l'unité de traitement, l'unité de communication filaire et non filaire, l'unité de stockage, les capteurs, les actionneurs ainsi que les convertisseurs ADC et DAC. Ces unités héritent de la classe *LinkableUnit* afin de pouvoir interconnecter tous les composants matériels. Elles ont des caractéristiques qui peuvent être personnalisées par l'utilisateur pour pouvoir par la suite utiliser ces informations dans la phase de génération de code source et pour lancer des analyses et des simulations.

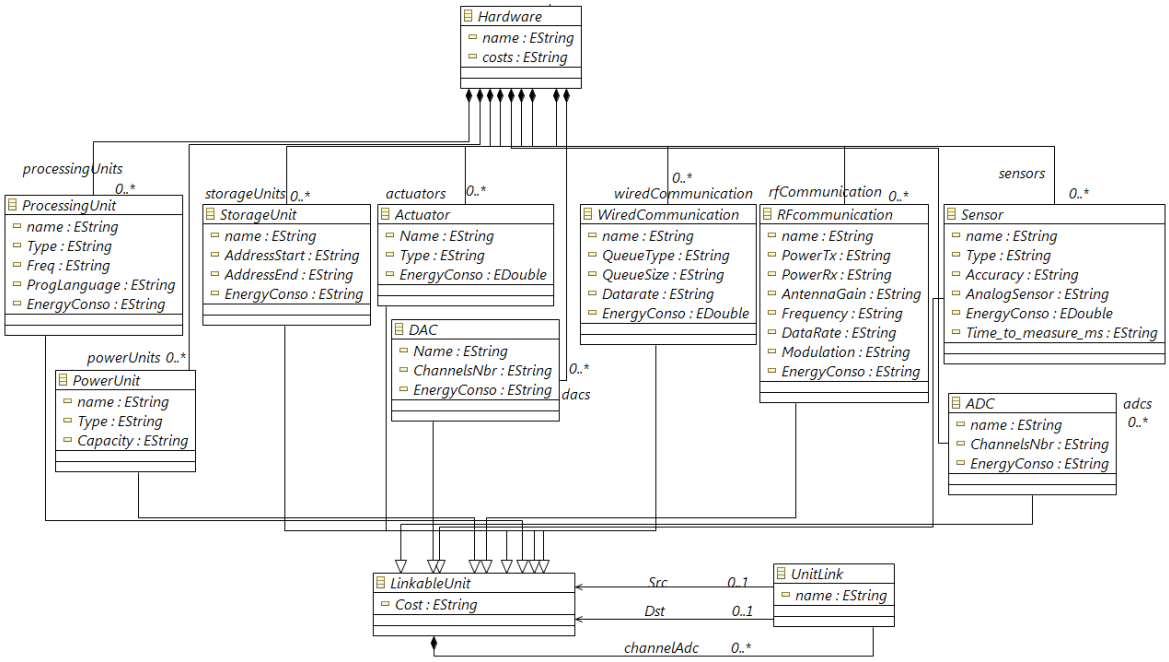


FIGURE 4.5 – Métamodèle de la facette matérielle.

4.2.3 Modélisation du comportement logiciel

La facette logicielle est décrite indépendamment de la facette matérielle de manière à pouvoir décrire un logiciel pour différentes architectures matérielles. Elle permet de spécifier et associer un modèle de comportement à chaque type de nœud.

Comme décrit dans la section 3.5.3, le logiciel peut être décrit avec plusieurs formalises. Dans le cadre de cette thèse, l'actuel formalise choisi est la machine à états finis. Il permet de décrire graphiquement le comportement d'un noeud sous forme de graphe d'états et transitions comme illustré dans la Figure 4.6. Ce formalisme peut être complété ou étendu par d'autres formalises afin de spécifier des comportements plus complexes.

États

Un état est une situation qui satisfait certaines conditions, effectue certaines activités ou attend un événement. On distingue un état particulier qui est l'état initial, qui sert de point de démarrage du processus. Chaque état est caractérisé par des fonctions qui s'exécutent à l'entrée (*Entry*), à l'intérieur (*Do*) et à la sortie (*Exit*) de l'état.

Transitions

Les transitions traduisent les relations entre deux états pour indiquer qu'un nœud dans le premier état entrera dans le second état lorsqu'un événement spécifique se produit. Chaque transition possède les attributs suivant :[Évènement] [Condition] [/ Action] :

— **Événement** : Un événement se réfère à quelque chose qui se produit pendant l'exé-

cution et qui mérite d'être pris en compte, par exemple un événement de réception d'un signal ou un événement temporel (un minuteur après cinq minutes tm(5min)).

— **Condition** : Si la condition est évaluée à vrai, la transition se produira, par exemple, un message de notification peut être envoyé si la condition suivante est vérifiée (batterie < seuil).

— **Action** : elle doit être effectuée avant la transition vers l'état d'arrivé. Dans les transitons, on retrouve des actions d'acquisition, d'encodage, de transmission et de réception.

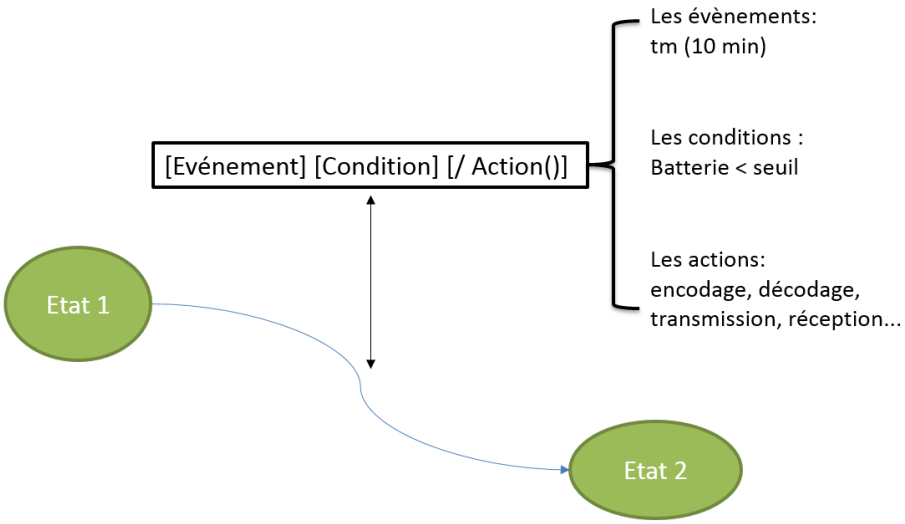


FIGURE 4.6 – Exemple de modèle de la facette logicielle.

La Figure 4.7 illustre le métamodèle de la facette logicielle. Elle est constituée d'un seul état initial et des états ordinaires. Les deux états héritent de la classe *linkedState* qui leur permet d'être interconnectés par des transitions. Chaque transition possède des informations de déclenchement pouvant actionner des fonctions.

4.2.4 Modélisation du flux de données

Cette facette décrit le flux de données dans un RdC. Un RdC peut avoir trois types de nœud de flux de données, à savoir des générateurs de données, des transformateurs de donnés et des consommateurs de donnés :

— Le générateur de données représente un groupe de nœuds capteurs qui ont le même flux de données. Il a un ou plusieurs ports d'acquisition des grandeurs physiques. Il a aussi un port de transmission des mesures ;

— Le transformateur de données représente un groupe de nœuds passerelle ou routeur qui ont le même flux. Il a un ou plusieurs ports de réception des flux entrants. Il possède aussi un ou plusieurs ports de transmissions des flux ;

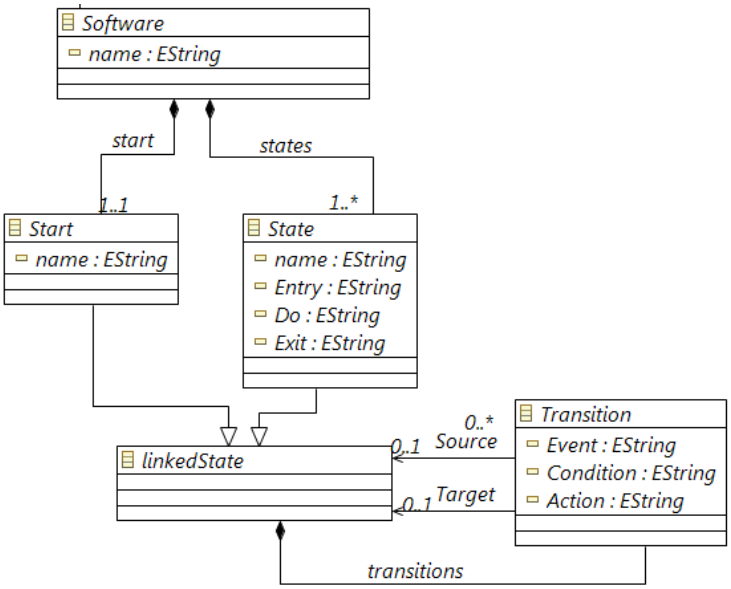


FIGURE 4.7 – Métamodèle de la facette logicielle.

— Le consommateur de données représente un groupe de nœuds actionneurs ou des pointes de collecte. Il possède un ou plusieurs ports de réception des flux entrants. On peut considérer que ce nœud est un transformateur de données qui ne produit pas de flux.

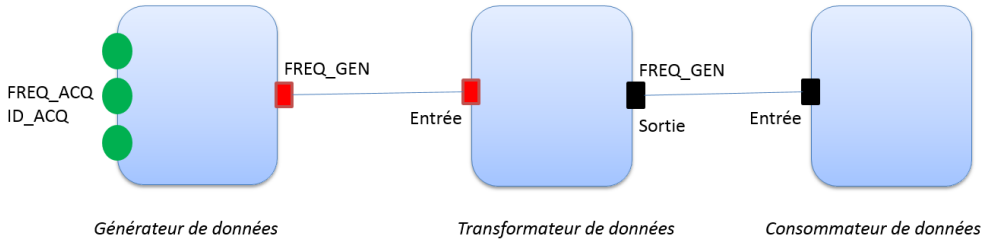


FIGURE 4.8 – Générateur et transformateur de données dans la facette flux de données.

Les trois types de nœuds peuvent être décrits à l'aide d'un seul modèle que nous décrivons dans cette section. Le modèle d'un noeud flux de données est composé de flux d'acquisition, flux entrant, flux sortant, de constantes, de paramètres et de fonctions qui communiquent des flux.

Flux d'acquisition

Le générateur de données peut avoir un ou plusieurs ports de flux d'acquisition. Chaque port est caractérisé par un identifiant de mesure (*ID*) et une fréquence d'acquisition.

Flux d'entrée

Un noeud transformateur ou consommateur de données peut avoir un ou plusieurs ports d'entrée de flux de communication. Chaque port est caractérisé par un *ID*.

Flux de sortie

Le transformateur de données peut avoir un ou plusieurs ports de transmissions des flux. Chaque port de transmission est caractérisé par un *ID*, une fréquence d'envoi et une adresse pour spécifier la destination des flux.

Constante

L'un des objectifs de la modélisation flux de données est de pouvoir décrire la structure de la trame de données. Dans cette trame, on peut avoir des constantes. Chaque constante est caractérisée par sa valeur, sa taille et son type. La structure de la trame est utilisée dans la génération de code.

Paramètre

Un noeud peut avoir un ensemble de paramètres à renseigner par l'utilisateur, chaque paramètre est caractérisé par une référence, une taille et un type.

Fonction

Chaque noeud flux de données possède des fonctions, par exemple, l'acquisition de mesures, encodage et décodage de la trame. Nous définissons une fonction générique (*Function*) à personnaliser par l'utilisateur. Chaque fonction possède des entrées (*In*) et des sorties (*Out*).

Flux

Le flux permet d'interconnecter les fonctions, les paramètres, les constantes et les ports par des liens de flux. Chaque lien contient des informations sur ce qu'ils communiquent comme le type, la taille et la structure des données.

La Figure 4.9 montre un exemple de générateur de données composé de deux ports d'acquisition (id_1 et id_2) caractérisés par une fréquence d'acquisition de 10 minutes. Les deux ports sont connectés à la fonction d'acquisition qui produit deux mesures M_1 et M_2 de taille respective T_1 et T_2. Les deux mesures ainsi que l'*ID* et les constantes (*FF* et *FE*) sont codés dans une trame définie par la fonction d'encodage. Enfin, la trame est alors envoyée toutes les 10 minutes.

La Figure 4.10 représente le métamodèle de la facette du flux de données. Un nœud flux de données peut se composer d'un flux entrant (*InputFlow*) et d'un flux de sortie

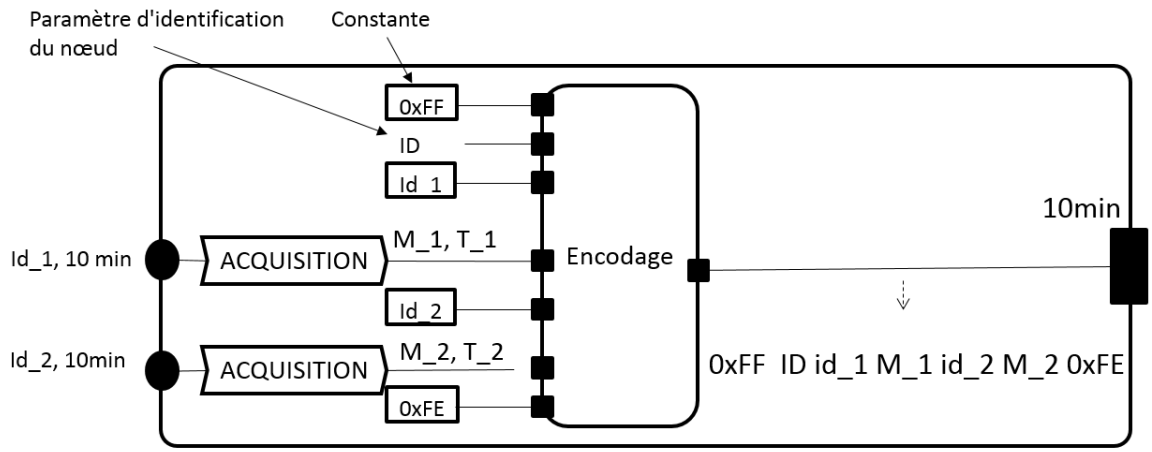


FIGURE 4.9 – Exemple de générateur de données.

(*OutputFlow*), des fonctions, des constantes et des paramètres. Dans ce métamodèle, l'ensemble des éléments héritent de la classe *Linkable* pour pouvoir s'interconnecter.

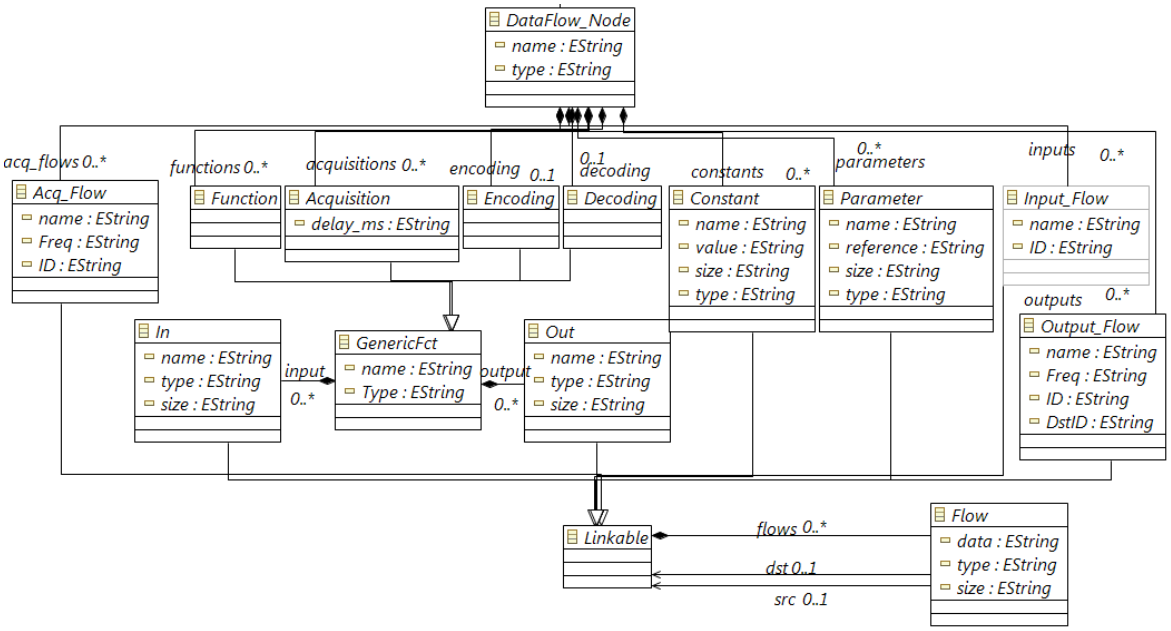


FIGURE 4.10 – Métamodèle de la facette du flux de données.

4.2.5 Modélisation de l'environnement physique

L'environnement physique concerne le site réel où le RdC sera déployé. Il influence les réseaux de capteurs, en particulier lorsque on considère les différents phénomènes liés aux communications entre les nœuds comme les atténuations et les réflexions des signaux par des obstacles.

La facette d'environnement physique est constituée de trois élements :

— **Zones physiques** : elles représentent les champs physiques dans lesquels les nœuds seront déployés. Chaque zone définit les positions des nœuds et aussi celles des obstacles. Elle est caractérisée par ses dimensions (*Length, Width*), ses coordonnées (*Latitude, Longitude*), l'image de la zone pour la visualisation, le type de déploiement, le modèle de propagation et un facteur d'atténuation pour les propagations radio ;

— **Obstacles** : ils représentent une partie de l'environnement qui peut influencer les communications entre les nœuds. Chaque obstacle possède des attributs de son emplacement, ses dimensions et l'atténuation engendrée selon le type du matériau (bois, béton, etc.) ;

— **Instances de nœuds** : elles représentent une instance de l'élément nœud du réseau. Elle est caractérisée par un identifiant, un emplacement. Dans le cas où le nœud est mobile, on peut préciser sa mobilité dans la zone, par exemple, en donnant un vecteur de coordonnées possibles.

Dans cette facette, un éditeur est utilisé pour créer des zones au niveau desquelles les nœuds peuvent être positionnés. La Figure 4.11 illustre un exemple de deux zones avec des nœuds positionnés dans la deuxième zone. En bas de la Figure 4.11, deux exemples de phénomènes de propagation du signal et leurs effets sur les signaux sont présentés. Il s'agit de la diffraction et de la réflexion des ondes électromagnétiques.

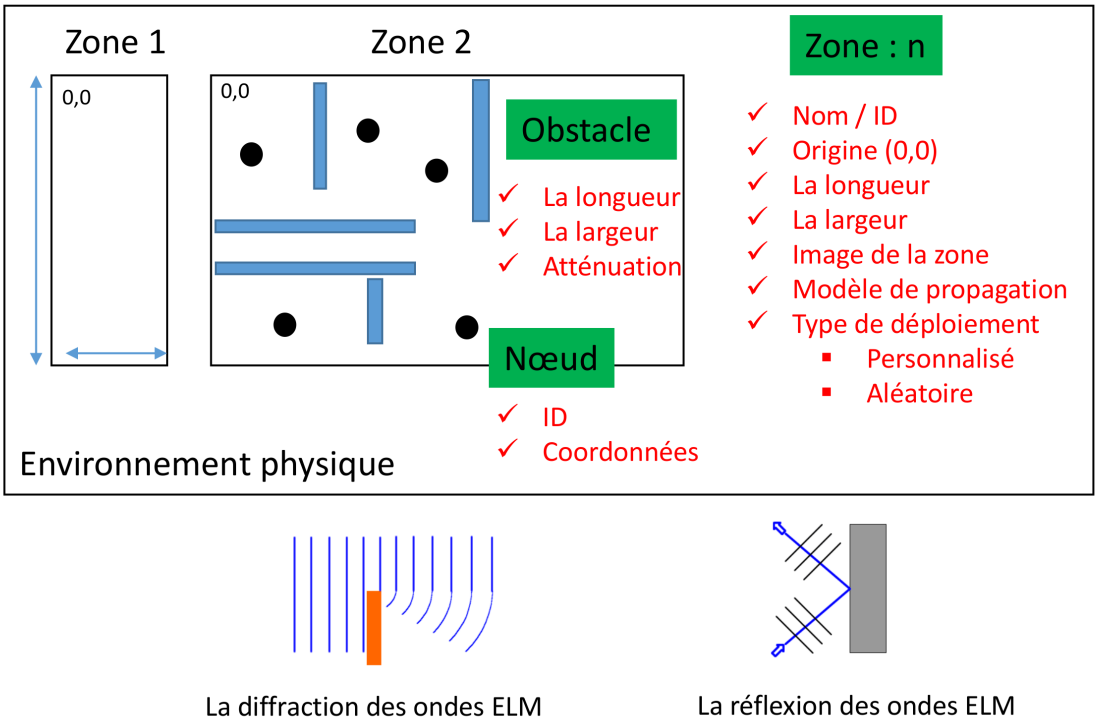


FIGURE 4.11 – Exemple de la facette de l'environnement physique.

La Figure 4.12 présente le métamodèle de la facette d'environnement physique en mo-

délisant les nœuds, les obstacles physiques et les zones de déploiement. Chaque élément est caractérisé par un ensemble d'attributs qui sont utilisés dans les analyses de cette facette. Parmi ces attributs, on retrouve le modèle de propagation radio qui est utilisé pour prédire la puissance du signal reçu de chaque paquet. Chaque émetteur-récepteur a une sensibilité de réception qui représente un seuil. Si la puissance d'un signal capté par un récepteur est inférieure à la sensibilité de réception de cet émetteur-récepteur, il ne sera pas possible de le décoder avec succès.

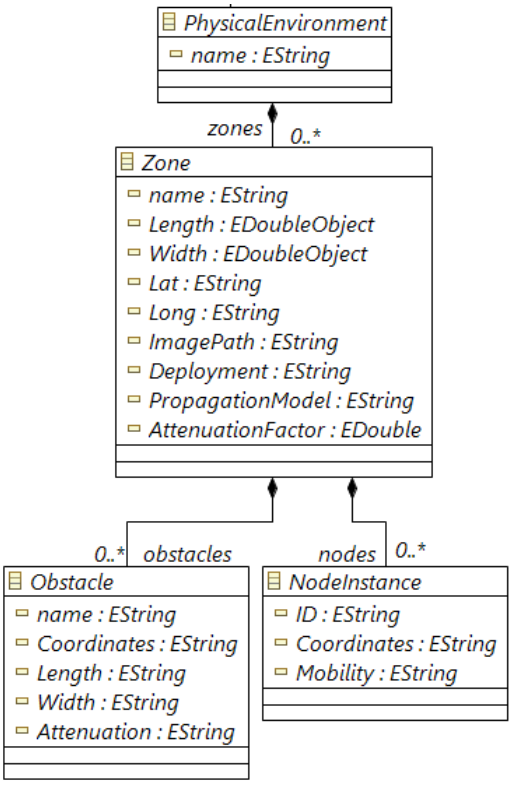


FIGURE 4.12 – Métamodèle de la facette d'environnement physique.

Actuellement, plusieurs modèles de propagation sont développés tels que le modèle de propagation en espace libre et le modèle de propagation indoor de Union Internationale des Télécommunications (UIT) [136]. Les deux modèles présentent deux environnements de travail typiques des applications de réseaux de capteurs, c'est-à-dire des scénarios intérieurs et extérieurs. Les utilisateurs peuvent choisir un modèle de propagation pour étudier l'influence de l'environnement sur les communications entre les nœuds. Nous pouvons intégrer aussi les informations liées aux atténuations des obstacles pour avoir une estimation plus précises.

4.2.6 Instanciation et combinaison des facettes : les projections

Dans la section 3.5.6, nous avons introduit la notion de facette projection qui permet de construire et d'explorer différentes solutions possibles (projections) en définis-

sant des combinaisons différentes de facettes. La figure 4.13 illustre un aperçu du métamodèle composé de plusieurs *ProjectionFacet* (la notation "1.. *" dans le métamodèle signifie "zéro à plusieurs") et cinq autres facettes (*NetworkFacet*, *HardwareFacet*, *SoftwareFacet*, *DataflowFacet*, *physicalEnvFacet*) peuvent être instanciées une seule fois (les spécifications "0.1" dans le métamodèle). Les cinq facettes sont une sorte de bibliothèque de modèles, par exemple, *NetworkFacet* peut contenir plusieurs modèles *Network*. (0..*). Chaque modèle est identifié par son nom (*Network* possède un attribut pour spécifier le nom). Ainsi, *ProjectionFacet* utilise les noms des modèles dans ses attributs pour définir une combinaison possible.

Il est à noter que les contraintes de la multiplicité des facettes et de leurs instances comme modèles permettent d'assurer la cohérence de l'environnement de modélisation.

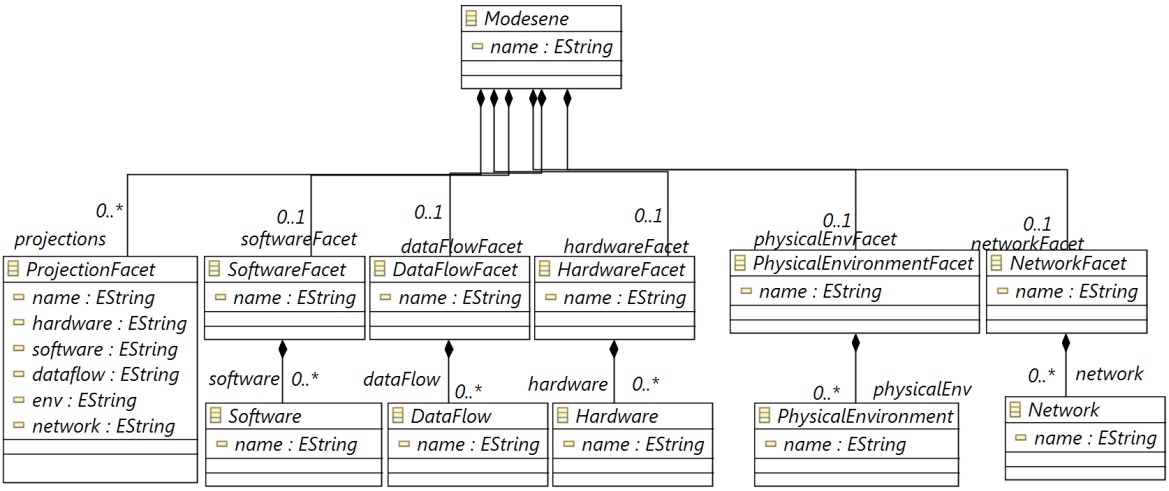


FIGURE 4.13 – Metamodèle de la facette de projection.

4.3 Transformations de modèles

Après avoir défini les modèles de chaque facette, nous décrirons les différents types de transformations de modèles que nous utilisons dans notre méthodologie à savoir :

— Des transformateurs entre les facettes permettant d'éviter de ré-décrire manuellement les modèles dans chaque facette ;

— Des transformations vers des simulations permettant de développer des passerelles vers des simulateurs pour l'estimation des performances ;

— Des transformations pour la génération automatique du code source embarqué dans les noeuds permettant de faciliter le développement logiciel et d'éviter les erreurs de programmation.

4.3.1 Transformations inter-facettes

Dans notre méthodologie, nous pouvons transformer des modèles d'une facette à une autre. Dans cette section, nous présentons trois exemples de transformations entre facettes :

1. Des transformations de la facette réseau vers la facette environnement physique.

2. Des transformations de la facette matérielle vers la facette flux de données.

3. Des transformations de la facette matérielle vers la facette logicielle.

Les transformations de la facette réseau vers la facette environnement physique ont pour but déployer virtuellement des noeuds réseau dans l'environnement physique. Cette transformation permet d'instancier chaque nœud du modèle réseau dans une zone définie par un modèle d'environnement physique. Ceci permet aux concepteurs d'éviter de redécrire manuellement des noeuds dans la facette d'environnement physique et donc de réduire le temps de développement.

Au sein d'une zone donnée, l'ensemble de nœuds peut être distribué de trois manières différentes :

— Aléatoirement : chaque nœud est placé de façon aléatoire dans la zone ;

— Maillé : les nœuds sont placés sur une grille avec un certain nombre de lignes et de colonnes (N*M) ;

— Personnalisé : chaque nœud est placé manuellement dans la zone. Dans ce cas, chaque nœud est déployé en utilisant des coordonnées(x,y).

Le type de déploiement est spécifié dans la *zone* comme nous l'avons souligné dans la description de la facette d'environnement physique (voir section 4.2.5). Cette automatisation permet de faciliter et d'accélérer le déploiement des noeuds.

L'algorithme 1 décrit un exemple de transformations en transformant de la facette

réseau vers la facette environnement physique.

Algorithm 1: Transformations de la facette réseau à la facette environnement physique.

```
Input  : modèle de la facette réseau
Output: modèle de la facette environnement physique
zone = zone de déploiement;
mode = type de déploiement de la zone;
for i ← 0 to sizeof(NœudsRéseau) do
    ListeNoeuds(i)= ajouter le noeud i;
end
if mode == aleatoire then
    for i ← 0 to sizeof(listeNoeuds) do
        X=rand(longueur de la zone);
        y=rand(largeur de la zone);
        instancier ListeNoeuds(i) dans la zone (X, Y);
    end
end
if mode == maillé then
    k=0;
    pasX=longueur de la zone/(N+1);
    pasY=largeur de la zone/(M+1);
    for i ← 0 to N do
        for j ← 0 to M do
            instancier ListeNoeuds(k) dans la zone ((i + 1) * pasX, (j + 1)* pasY) ;
            k=k+1;
            if k == sizeof(listeNoeuds) then
                break;
            end
        end
    end
end
if mode == personnalisé then
    X []=Lire les coordonnées X ;
    Y []=Lire les coordonnées Y ;
    for i ← 0 to sizeof(listeNoeuds) do
        instancier le ListeNoeuds(i) dans la zone avec les coordonnée (X(i), Y(i)) ;
    end
end
```

La deuxième transformation consiste à établir un lien entre la facette matérielle et la facette du flux de données. Chaque capteur (température) présent dans la facette maté-

rielle est automatiquement transformé en port et fonction d'acquisition dans la facette flux de données. Les fonctions sont instanciées avec des noms de capteurs et un préfixe get (get_temperature). L'algorithme 2 décrit les transformations entre les deux facettes.

Algorithm 2: Transformations de la facette matérielle vers la facette flux de donnée.

Input : la facette matérielle.
Output: la facette flux de donnée.
for *chaque capteur Ci dans la facette matérielle* **do**
    ajouter un port d'acquisition ;
    ajouter une fonction Fi (get_nom_capteur);
end

Les transformations de la facette matérielle vers la facette logicielle permettent de décrire automatiquement le comportement d'un noeud capteur. Ainsi, chaque capteur (Humidité) présent dans la facette matérielle est transformé en un état d'acquisition (acquisition_Humidité). Ces transformations sont décrites dans l'algorithme 3.

Algorithm 3: Transformations de la facette matérielle vers la facette logicielle.

Input : la facette matérielle.
Output: la facette logicielle.
for *chaque capteur Ci dans la facette matérielle* **do**
    ajouter un état acquisition_nom_capteur;
    Spécifier la fonction enable_nom_capteur;
    if $Time\_to\_mesure\_ms != 0$ **then**
        Spécifier la fonction delay($Time\_to\_mesure\_ms$);
    end
    Spécifier la fonction get_nom_capteur;
    Spécifier la fonction disable_nom_capteur;
end

4.3.2 Analyses et simulations

Notre méthodologie supporte des analyses et des simulations qui peuvent être internes ou externes via des passerelles vers des simulateurs tels que Omnet++ [5] et NS2 [76]. Un exemple de simulation comprend l'estimation des pertes de paquets, la latence et la durée de vie. Nous exploitons les facettes de modélisation afin de fournir une estimation des performances des réseaux de capteurs. Toutes les facettes de modélisation sont analysées, combinées et traduites en scripts de simulation qui peuvent être exécutés par des simulateurs. Cette transformation permet de vérifier que les modèles proposés présentent un niveau de détail suffisant pour la simulation.

La Figure 4.14 illustre un exemple de passerelle vers un simulateur en utilisant des transformations de modèles pour générer un script de simulation. A travers les cinq facettes, nous obtenons les informations nécessaires aux simulations telles que :

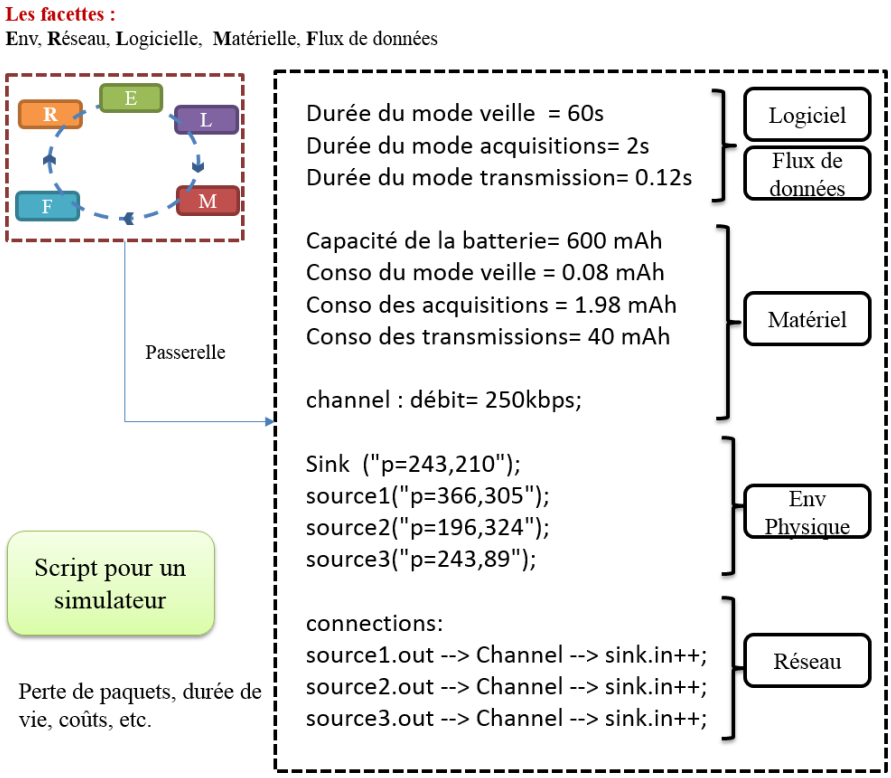


FIGURE 4.14 – Transformations de modèles pour les analyses/simulations.

— La facette réseau permet de spécifier les protocoles MAC et de routage utilisés. Le nœud peut spécifier des protocoles de routage à sauts multiples ou à sauts uniques. Un routage statique peut également être défini en spécifiant explicitement les liens entre les nœuds. Ces paramètres peuvent être utilisés pour estimer la latence ou la perte de paquets dans un réseau ;

— La facette matérielle permet de définir les consommations d'énergie pour toutes les unités matérielles présentes dans le noeud. Le modèle de cette facette spécifie aussi la capacité de la batterie ;

— La facette logicielle décrit le comportement du nœud. Cela comprend les activités de sommeil, d'acquisition et de transmission. Ces informations sont complétées par la facette flux de données qui spécifie les fréquences d'acquisition et d'envoi. Ainsi, il est possible d'estimer la durée de vie d'un noeud en combinant les informations des facettes ;

— La facette environnement physique peut être utilisée pour générer des paramètres et des scripts permettant d'estimer l'énergie nécessaire pour la transmission des données sur le canal physique ou pour vérifier la portée et la qualité des liens entre les nœuds.

4.3.3 Génération de code

La génération automatique de code source à partir des modèles est une autre caractéristique importante de l'approche IDM. Ces transformations combinent à la fois les modèles de code qui sont indépendants de la plateforme cible et ceux qui en sont dépendants.

Dans notre environnement, nous générons le code en deux étapes. La première consiste à transformer les modèles des facettes en un macrocode indépendant de la plateforme. Cette étape permet de réduire la difficulté de programmation en développant des modèles génériques par rapport à un langage et plateforme cible. Les transformations automatiques vers le macrocode permettent de réduire le temps de développement.

La Figure 4.15 illustre le métamodèle du macrocode qui se compose de :

— **Bibliothèque (*Library*) :** elle représente les noms des bibliothèques à inclure dans le macrocode ;

— **ID du nœud (*NodeId*) :** il correspond à l'identifiant du noeud à coder ;

— **Trame (*ApplicationFrame*) :** elle décrit la structure de la trame à envoyer par le noeud. Elle est caractérisée par un type et une taille, par exemple, une trame de type double avec 20 champs de données ;

— **Fonction (*Function*) :** elle décrit les fonctions du macrocode. Elle possède un attributs *Parameters* pour identifier ses paramètres ;

— **Variable (*Variable*) :** elle décrit les variables échangées entre les fonctions. Chaque variable est caractérisée par un nom, un type et une taille selon le type ;

— **Boucle** : elle est composée de deux balises (*LoopStart*, *LoopEnd*). Chacune d'entre elles est caractérisée par une condition ;

— **Commentaire (*Comment*) :** il facilite la compréhension du code.

Il faut noter que ce macrocode peut être initialisé automatiquement à partir des facettes. Par exemple, nous pouvons lire le modèle de la facette matérielle et nous déclarons automatiquement toutes les bibliothèques nécessaires à décrire dans le macrocode. Ainsi, l'utilisateur peut modifier le macrocode.

La deuxième étape consiste à transformer le macrocode en code source en intégrant des bibliothèques et des fichiers de définition par rapport à une plateforme choisie et dans un langage défini.

La Figure 4.16 illustre le métamodèle des fichiers de définition. Un fichier de définition (*Definefile*) peut être composé de plusieurs définitions (*Definition*). Chaque définition inclut un ensemble de fonctions (*Function*). Chaque fonction est caractérisée par un nom de fonction qui correspond à celui du macrocode et un nom qui correspond au nom implémenté dans les fichiers de la bibliothèque. Par ce fait, il est possible de faire un lien entre les deux fonctions.

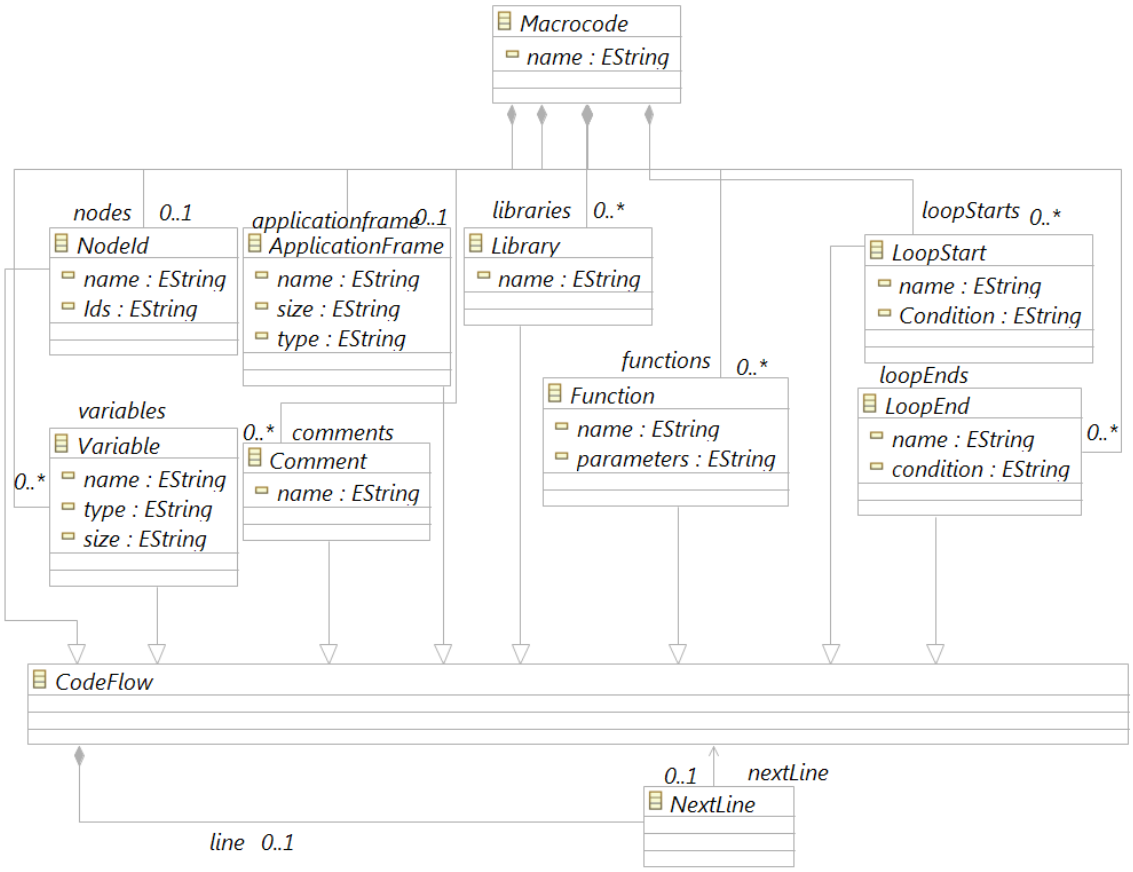


FIGURE 4.15 – Métamodèle du macrocode utilisé dans la génération de code.

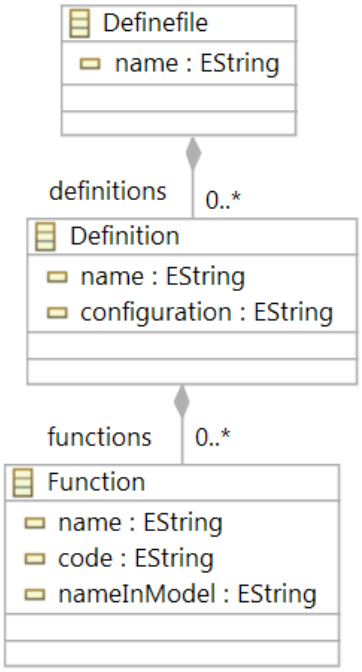


FIGURE 4.16 – Métamodèle des fichiers de définition utilisé dans la génération de code.

4.4 Conclusion

Ce chapitre décrit les éléments de modélisation de notre méthodologie pour fournir un environnement d'aide à la conception des RdC. Cet environnement est formalisé à l'aide des techniques de métamodélisation et des transformations de modèles. Il offre un environnement de description de modèles à base de plusieurs facettes qui décrivent l'architecture réseau, l'architecture matérielle, le comportement logiciel, les flux de données et l'environnement physique. Nous avons également mis en place une facette transverse (la facette projection) qui nous permet de faire des liens entre les facettes.

Grâce aux transformations de modèles, la méthodologie supporte une génération automatique des scripts de simulations pour l'estimation du performance du RdC. Elle supporte aussi une génération automatique des codes sources des nœuds en passant par un macrocode (un nouveau concept que nous avons également définit dans la méthodologie). Ces transformations facilitent la programmation et le développement des applications des réseaux de capteurs.

Chapitre 5

Validation de la méthodologie sur un cas d'étude

« *Knowledge is useless without consistent application.* »

Julian Hall

Sommaire

5.1 Introduction

Les systèmes informatiques sont largement utilisés dans les villes intelligentes pour accroître leur efficacité opérationnelle et améliorer la qualité des services publics [137]. Dans ce contexte, le bâtiment intelligent est un élément important du concept de la ville intelligente. Il est défini comme un bâtiment avec une grande efficacité énergétique, qui utilise une gestion intelligente des équipements consommateurs et producteurs d'énergie [138].

Pour réduire la consommation d'énergie dans les bâtiments, il est nécessaire de développer des solutions intelligentes capables de fournir des systèmes à très haute efficacité énergétique. Ces solutions intelligentes s'appuient sur le déploiement de réseaux de capteurs pour la surveillance des équipements.

Après avoir proposé une nouvelle méthodologie pour la conception des réseaux de capteurs, nous avons implémenté cette méthodologie dans un framework appelé *MOdel Driven Environment for SEnsor NEtwork (MODESENE)*. Pour valider les fonctionnalités de ce framework, nous avons développé un cas d'étude dans le domaine de bâtiments intelligents. Dans les sections suivantes, nous allons détailler ce cas d'étude, puis montrer sa description avec notre framework.

5.2 Cas d'étude : mesure climatique pour le bâtiment

Pour développer un système de mesures pour le bâtiment, nous avons conçu des capteurs modulaires capables d'effectuer plusieurs mesures dans des zones spécifiques. Ces mesures sont transmises et stockées dans des bases de données pour être traitées par les utilisateurs via des interfaces graphiques. L'environnement physique de déploiement de ce réseau est notre école *ESIEE-Paris*. Comme le montre la Figure 5.1, il se compose de six bâtiments, chaque bâtiment compte cinq étages contenant plusieurs pièces. Il y a aussi un hall principal, des amphithéâtres, des bureaux administratifs, une cantine, une salle de gymnase et une bibliothèque. Pour assurer la mesure climatique et couvrir l'ensemble des bâtiments, nous avons besoin d'un réseau de capteurs d'une centaine de nœuds capteurs. Il possède une architecture hiérarchique à trois niveaux :

1. Niveau capteurs pour la mesure de données environnementales et l'envoi de ceux-ci aux nœuds passerelles ;
2. Niveau passerelles (cartes Raspberry Pi) pour la réception et l'acheminement des mesures vers une station de base ;
3. Niveau station de base pour la collecte et le stockage des données dans une base de données interne et/ou externe (le cloud).

La Figure 5.2 montre deux topologies possibles pour ce RdC : une topologie en étoile utilisant la technologie LoRa et une topologie en cluster utilisant la technologie ZigBee . La topologie en étoile nécessite un seul nœud passerelle alors que la solution basée ZigBee

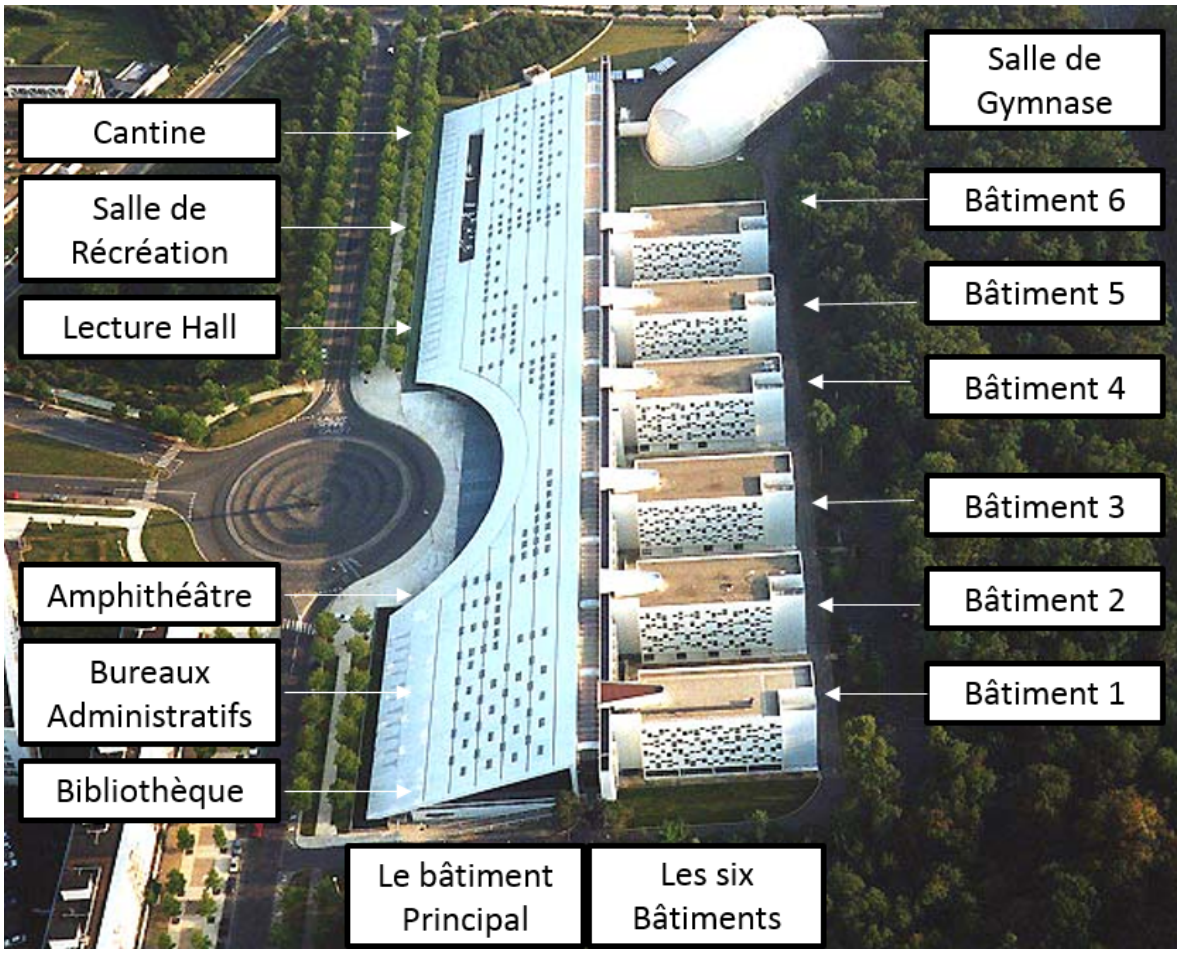


FIGURE 5.1 – L'environnement physique du déploiement du réseau de capteurs.

implique plusieurs nœuds passerelles : à chaque étage, il faut une passerelle pour couvrir une zone d'une dizaine de mètres, ce qui donne au total 6*5=30 cartes Raspberry Pi pour les six bâtiments.

Pour déployer les architectures proposées, nous avons besoin de nœuds capteurs. De nombreuses plateformes de nœuds capteurs ont été conçues et mises en œuvre dans le cadre de nombreux projets [139]. En ce qui concerne notre application, ces plateformes ont des inconvénients liés à la communication entre les nœuds. Chaque plateformes ne peut pas communiquer avec une autre plateformes de famille différente (par exemple, MicaZ ne peut communiquer qu'avec MicaZ), il est de ce fait difficile de mettre en œuvre un réseau hétérogène comme c'est le cas dans notre réseau. De plus, ces plateformes sont dépassées et ne conviennent pas à de nouvelles applications. La programmation est également limitée à des systèmes d'exploitation spécifiques (par exemple, TinyOS et Contiki) qui sont difficiles à optimiser. Ces plateformes sont compliquées à modifier pour supporter de nouveaux capteurs ou de nouvelles technologies de communication pour nos applications.

Afin de surmonter ces inconvénients, nous avons développé notre propre nœud capteur avec comme objectifs une faible consommation d'énergie, une architecture géné-

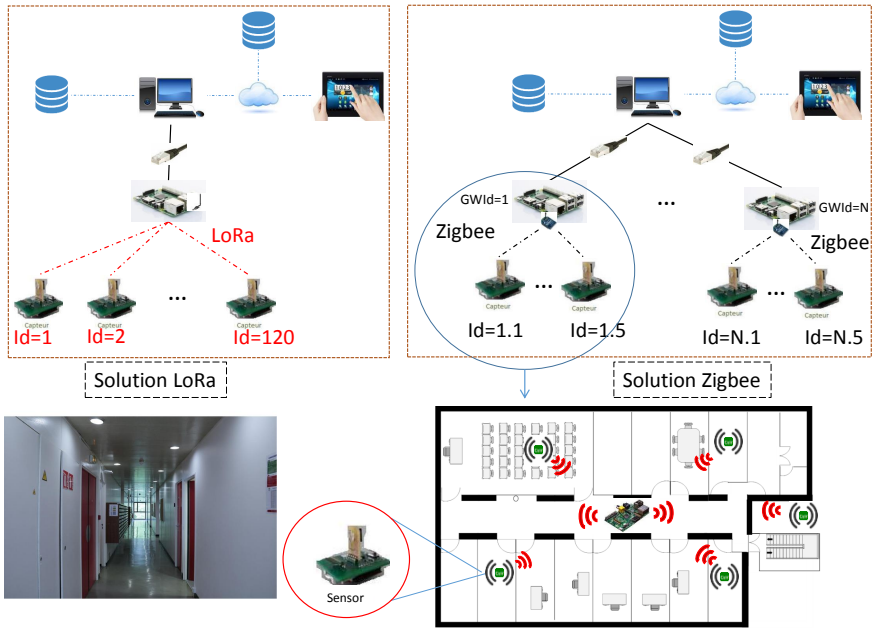


FIGURE 5.2 – Les deux solutions proposées pour le scénario d'utilisation du bâtiment intelligent.

rique, ouverte, évolutive et flexible. Dans les sections suivantes, nous discutons de la conception matérielle et des spécifications logicielles de cette plateforme.

5.2.1 Conception matérielle du capteur

Le nœud capteur possède des composants pour collecter, traiter et transmettre des données. Comme le montre la Figure 5.3, nous avons choisi pour notre capteur une architecture générique et modulaire. Cette architecture comporte quatre couches : (1) une couche d'acquisition, (2) une couche de traitement, (3) une couche de communication et (4) une couche d'alimentation.

La couche d'acquisition est composée de plusieurs dispositifs de détections ou de mesures qui ont une faible consommation d'énergie, un encombrement réduit, une bonne précision et une large dynamique de mesure. Cette couche peut supporter des capteurs de température, d'humidité, de luminosité, de monoxyde de carbone *CO* et de composés organiques volatils *COV*. Dans notre cas d'étude, le capteur de lumière ambiante peut être utilisé pour gérer la consommation d'énergie des systèmes d'éclairage dans les bâtiments. Les capteurs de *COV* et de *CO* peuvent mesurer le niveau de pollution et la qualité de l'air. Les dispositifs de température et d'humidité permettent d'étudier les performances et l'efficacité du système de climatisation.

La couche de traitement nécessite peu de calculs. Elle inclut la détection et l'envoi de quelques octets à basse fréquence. Pour cela, nous avons choisi le microcontrôleur *PIC18F2480* caractérisé par une faible consommation d'énergie, un faible coût et une mise en œuvre peu complexe [140].

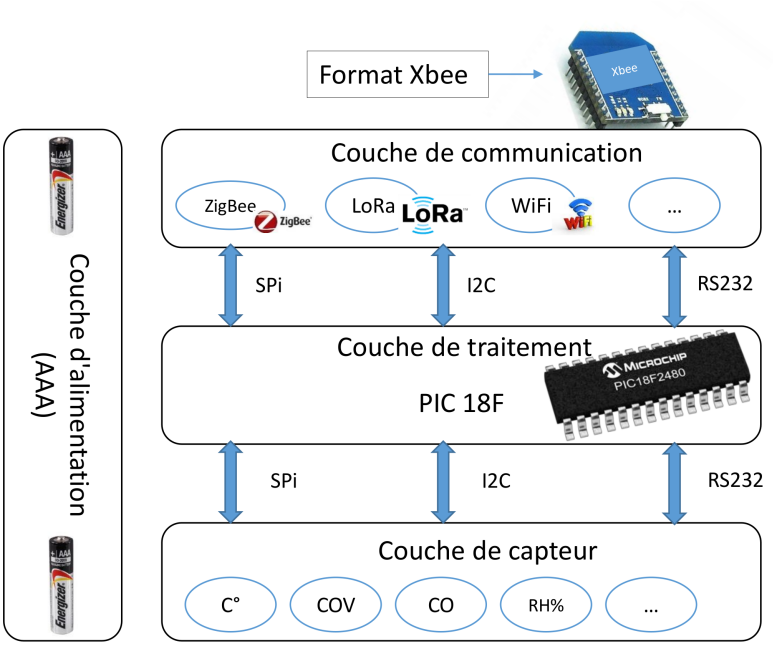


FIGURE 5.3 – L'architecture matérielle multicouche pour le nœud de capteur.

La couche de communication est basée sur des modules radios au format *Xbee* comme le montre la Figure 5.3. Grâce à ce format, le module de communication d'une technologie donnée peut être facilement remplacé par un autre module d'une technologie différente sans remettre en cause tout le nœud capteur. Ainsi, afin d'offrir de meilleures performances en termes de portée, nous pouvons, par exemple, changer un module de communication ZigBee par un module LoRa . Dans ce format xbee, il existe différentes technologies de communications : WiFi, Bluetooth, BLE, ZigBee, LoRa. Cette variété rend notre nœud capteur générique et réutilisable avec différentes technologies de communications. Pour notre RdC, nous avons opté pour les technologies ZigBee [141] et LoRa [142]. Ce choix se justifie par les inconvénients des autres technologies en termes de coût, d'autonomie et de consommation d'énergie.

Pour la couche d'alimentation, le nœud capteur peut avoir trois sources d'alimentation : une batterie, un panneau solaire ou être alimenté par le secteur. Dans notre nœud capteur, nous utilisons deux piles AAA ; elles ont une autonomie importante, un faible coût et une disponibilité sur le marché. Ces deux batteries peuvent fournir 18720 joules [143]. Cependant, le capteur de COV doit être alimenté par secteur car il nécessite une consommation d'énergie conséquente.

5.2.2 Étalonnage et conditionnement

Après la conception matérielle du nœud capteur (voir Figure 5.4), nous l'avons étalonné en comparant les acquisitions des mesures à celles de l'équipement d'étalonnage.

Afin de protéger chaque nœud capteur, nous avons aussi conçu des boîtiers reproduc-

tibles avec une imprimante 3D.

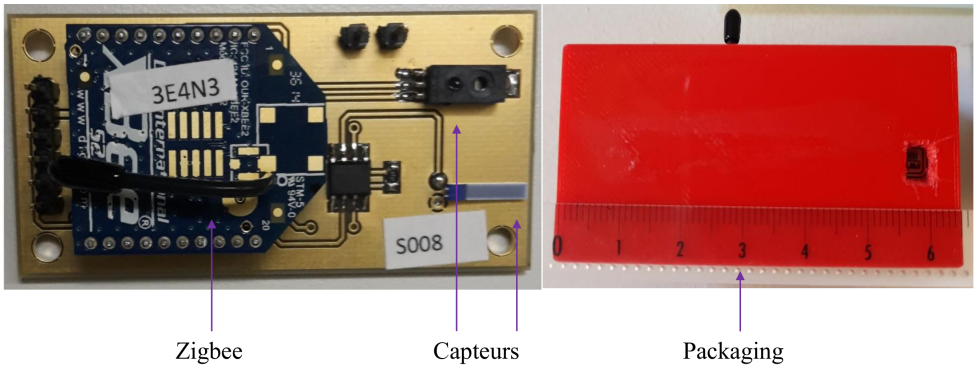


FIGURE 5.4 – Quelques vues de notre nœud de capteur.

La taille physique de chaque nœud de capteur a un impact significatif et direct sur la facilité de déploiement. Des nœuds plus petits peuvent être placés à plus d'endroits et utilisés dans davantage de scénarios. Nous avons alors suivi un processus de miniaturisation de notre nœud capteur, la Figure 5.5 illustre la dernière version.

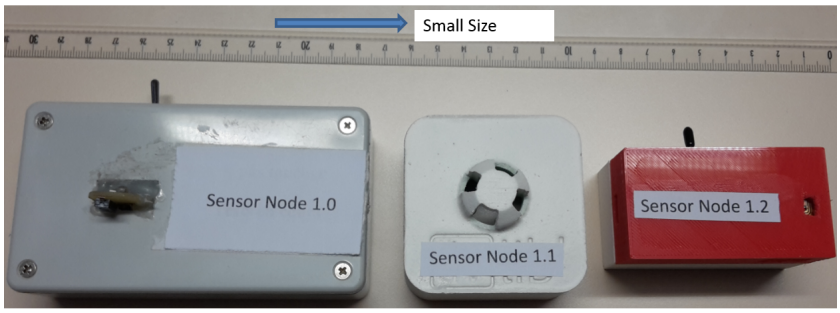


FIGURE 5.5 – Le processus de miniaturisation de notre nœud de capteur : vers la petite taille.

5.2.3 Spécification du logiciel

Après la conception matérielle, les nœuds capteurs sont programmés pour mesurer et envoyer périodiquement les données avec un intervalle de temps configurable. La Figure 5.6 décrit un exemple de comportement de notre nœud capteur. Après l'initialisation des entrées/sorties, du module de communication et la définition du temps de veille, le nœud capteur active ses entrées/sorties et effectue l'acquisition des mesures, puis il active le module de communication pour la transmission des données. Le nœud capteur attend ensuite de recevoir les commandes en provenance de la passerelle pour une éventuelle reconfiguration dynamique du capteur. En effet, il existe deux types de commandes : la première permet la reconfiguration de la fréquence d'échantillonnage (modification de la

durée du mise en veille), la seconde donne l'ordre de désactiver le nœud pour une durée déterminée. Après la réception de ces commandes ou si aucune commande n'est reçue dans un délai imparti, le capteur passe à l'étape de mise en veille pendant une durée fixe (temps de veille). Les entrées/sorties et le module de communication sont également désactivés pour minimiser la consommation d'énergie du capteur .

Les communications entre les nœuds capteurs et les nœuds passerelles sont structurées en trames. Pour optimiser la consommation d'énergie, nous avons également défini un nouveau protocole de communication pour la couche application : il permet d'envoyer toutes les mesures sur une seule trame décrite comme suit :

1. FF (First Field) : il indique le début de la trame.
2. ID : il représente l'identificateur du nœud capteur.
3. Data : ce champ inclut toutes les mesures.
4. FE (Field End) : il indique la fin d'une trame.

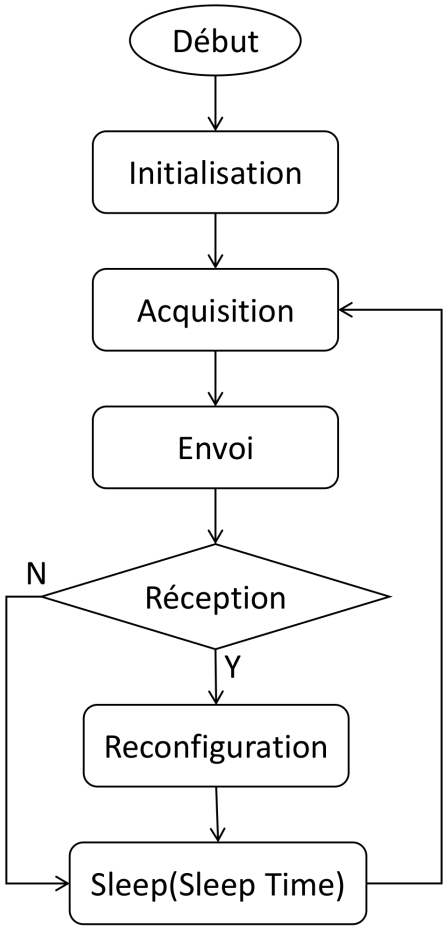


FIGURE 5.6 – Le comportement de nos nœuds de capteurs peut être décrit comme un organigramme.

Comme illustré dans la Figure 5.7, chaque champ de données (*Data*) est composé d'un identificateur indiquant le type de mesure (*Measurement-type-id*) et la valeur mesurée. Le champ *Measurement-type-id* identifie le type de données tels que la température (*température-type-id*= 1) ou l'humidité (*humidité-type-id* = 2). Dans le but de minimiser le traitement sur le nœud capteur et réduire la consommation d'énergie induite par son module de communication, le capteur envoie à la passerelle des données brutes (sans aucune interprétation). Ces données sont envoyées sur deux octets (valeur de poids fort et poids faible : *low value, high value*). Ce protocole peut prendre en charge plusieurs champs de données dans le cas où un nœud contient plusieurs dispositifs de détection. Ceci permet donc d'envoyer plusieurs mesures dans une seule trame. La trame de notre capteur peut inclure par conséquent la tension de la batterie utile pour estimer la durée de vie des nœuds capteur. Grâce à cette structuration, le ratio entre la taille des données utiles et la taille de toute la trame favorise l'efficacité des communications dans le RdC.

5.2.4 Stockage des données

Pour pouvoir enregistrer et exploiter les données du réseau de capteurs, nous avons conçu une architecture générique prenant en compte deux types de supports de stockage : une base de données locale et une base de données externe (cloud).

La base de données locale est utilisée pour stocker les données et la configuration du réseau de capteurs. Elle est gérée et hébergée dans un serveur mis en œuvre à l'aide de l'outil *WampServer*. Ce serveur, considéré comme une station de base, est équipé d'une connexion Ethernet à deux flux : un flux véhiculant les données envoyées par les nœuds passerelles et un autre flux fournissant aux utilisateurs un accès aux mesures.

La base de données externe (cloud) est utilisée pour permettre l'interfaçage avec des plateformes existantes de l'IoT telle que *ThingSpeak* [144]. Il s'agit d'une plateforme open source et d'une API pour la récupération et le stockage de données utilisant le protocole Internet. Cette solution accélère ainsi la mise en œuvre du RdC puisque le concepteur n'a plus qu'à instancier et personnaliser sa base de données.

Dans le cadre du cas d'étude, nous avons créé un compte sur *ThingSpeak* et configuré les canaux pour recevoir les données. Le compte utilise une licence annuelle gratuite (période d'accès et taille de message limitées à 15 secondes et 3ko respectivement). Il peut avoir plusieurs canaux comme le montre la Figure 5.8. Chaque canal contient 8 champs de données. Pour la sécurité des données, ThingSpeak offre deux types de canaux, un canal public et un canal privé. Le canal public est accessible à tous les utilisateurs sur le cloud via une url (*https ://thingspeak.com/channels/public*). Le canal privé est protégé par deux mots de passe (readkey, writeKey).

Après avoir décrit notre étude de cas (du capteur jusqu'à la base de données), nous allons décrire la mise en oeuvre de notre méthodologie et présenter les modèles associés à ce cas d'étude.

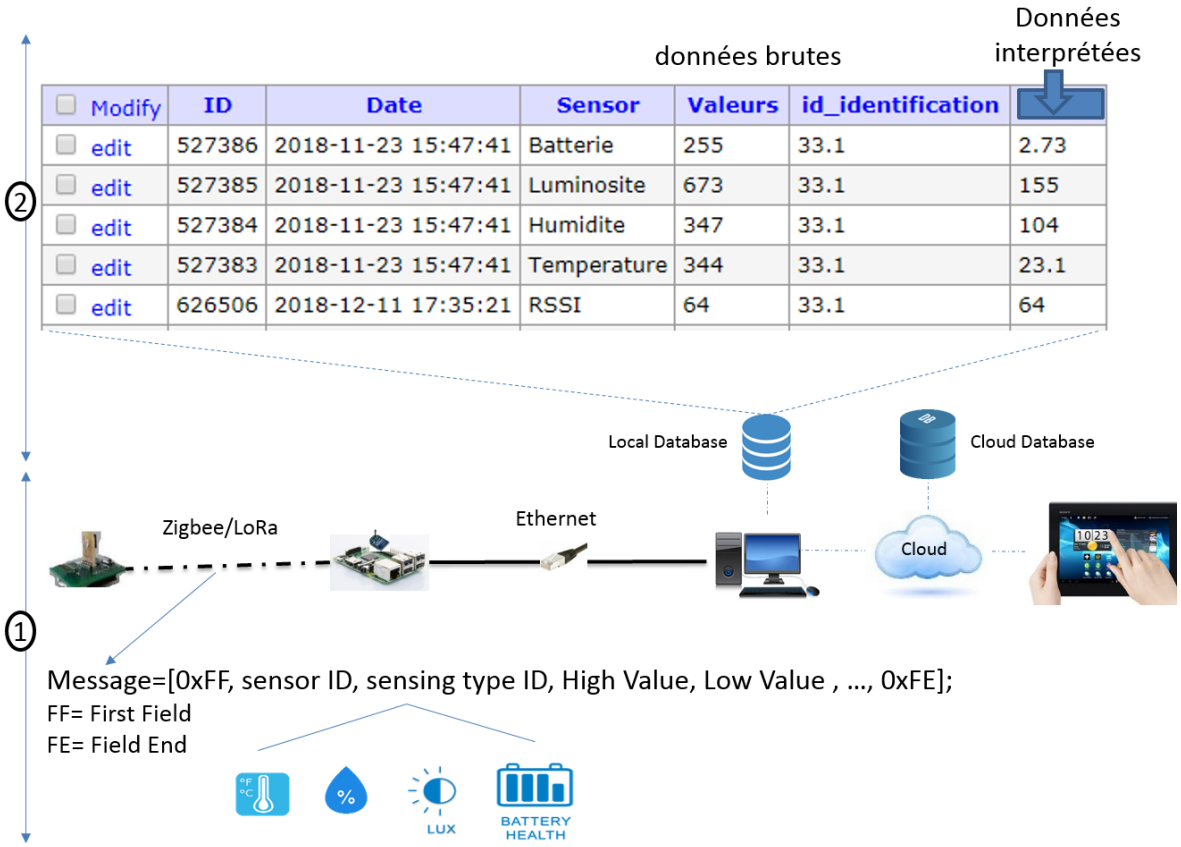


FIGURE 5.7 – (1) Format de la trame envoyée par les capteurs, (2) structure de la base de données.

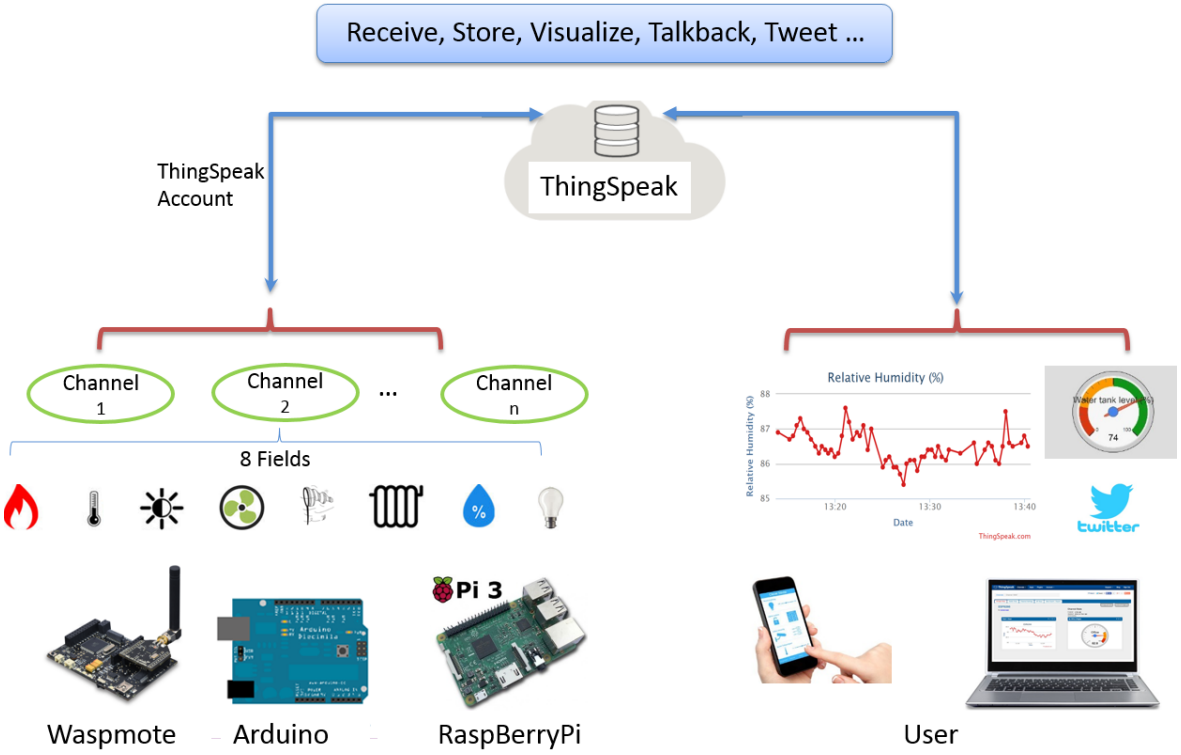


FIGURE 5.8 – La base de données ThingSpeak pour le stockage et la visualisation des données via le cloud.

5.3 Conception du RdC : descriptions des facettes

La Figure 5.9 illustre le processus de conception et d'utilisation de l'environnement MODESENE. Le processus comporte trois phases : modélisation des facettes, des analyses et des simulations et génération automatique du code.

- ✓ Tout d'abord, les différentes facettes sont décrites, puis mises en cohérence grâce aux transformations de modèles. Par exemple, les nœuds créés dans la facette réseau peuvent être automatiquement instanciés dans la facette environnement physique ou vice-versa.
- ✓ Ces descriptions sont ensuite validées par des analyses et des simulations internes, ou dans des outils tiers, via les passerelles associées. Cette étape permet d'évaluer les performances réseau telles que la durée de vie, le coût, les pertes de paquets, etc. En traitant ces analyses, les utilisateurs peuvent évaluer différentes configurations possible et choisir la plus adaptée.
- ✓ Enfin, des transformations de modèles sont appliquées pour générer automatiquement le code source embarqué dans le nœuds. Ces transformations permettent aussi de configurer les interfaces d'exploitation de données.

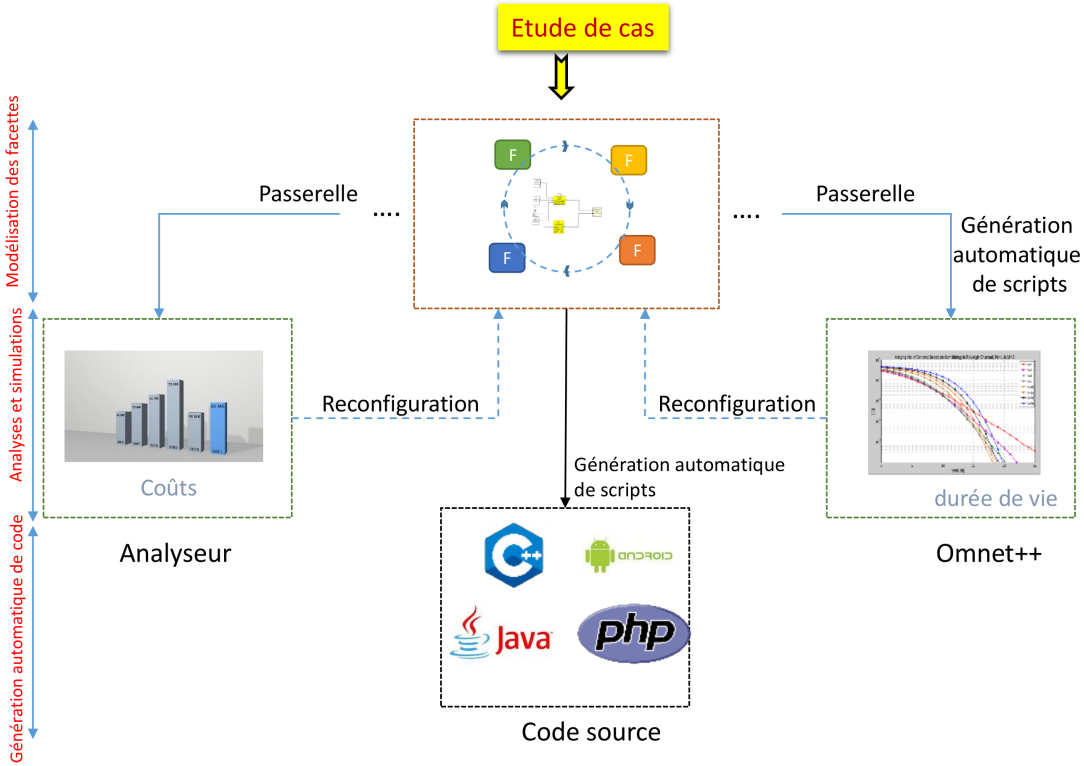


FIGURE 5.9 – Le processus de conception à travers Modesene.

Après avoir défini et présenté les métamodèles dans le chapitre précédent, nous avons configuré les fichiers qui nous permettent de gérer les palettes et leurs représentations

graphiques de l'outil GMF. Les configurations sont faites sur l'ensemble des facettes. Dans les sections suivantes, nous présentons des exemples de facettes développées en lien avec le cas d'étude.

5.3.1 Facette projection

Nous commençons par décrire une vue d'ensemble de toutes les facettes de MODESENE à travers l'étude de cas (Figure 5.10). Comme nous avons deux architectures du nœud capteur (capteur LoRa, capteur ZigBee), la facette matérielle présente deux instances comme le montre la Figure 5.10. Pour définir l'architecture matérielle de chaque noeud, il suffit d'un double-clic sur chaque facette (*HwLoRa*, *HwZigBee*). La Figure 5.10 illustre aussi deux projections : *ProjectionZigBee* et *ProjectionLoRa*. La facette *ProjectionZigBee* regroupe les facettes *Network*, *HwZigBee*, *SwCapteur*, *Dfcapteur*, *Env1*. La facette *ProjectionLoRa* regroupe les facettes *Network*, *HwLoRa*, *SwCapteur*, *Dfcapteur*, *Env1*.

Il est possible de passer de cette interface présentée dans la Figure 5.10 à la facette réseau en double-cliquant sur l'icône *Network* et nous aurons l'interface qui sera présentée dans la section 5.3.2.

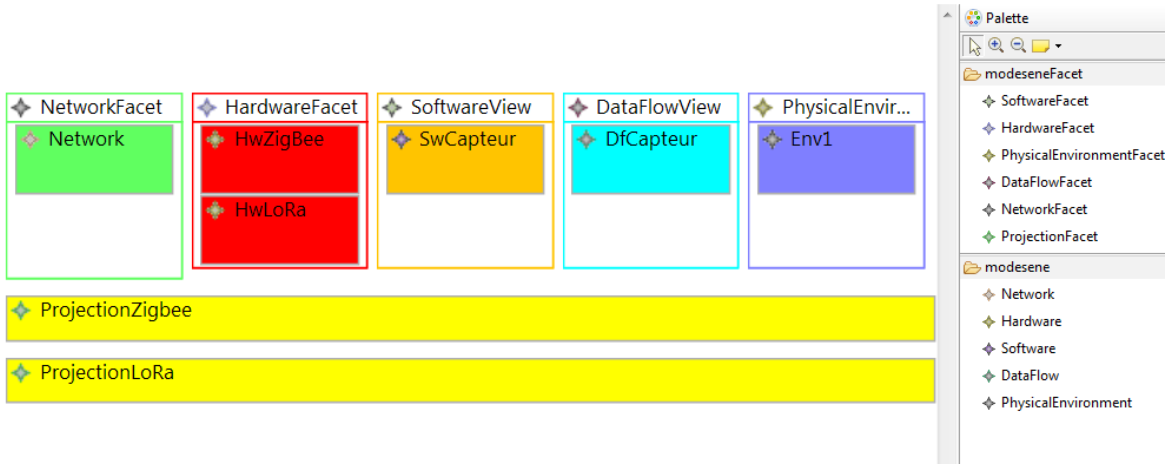


FIGURE 5.10 – La facette projection de l'étude de cas.

5.3.2 Facette réseau

La facette réseau est utilisée pour modéliser les nœuds capteurs, passerelles, bases de données et utilisateur. Elle permet aussi de décrire la topologie du réseau. La Figure 5.11 montre un exemple simple d'un RdC correspondant à une topologie étoile. Dans cet exemple, nous avons trois noeuds capteur (1, 2 et 3), un noeud passerelle (42), deux bases de données et un utilisateur connectés par un réseau Internet.

Des informations telles que l'*ID* des nœuds, la connectivité entre ces nœuds, le type de port et des paramètres de connexion (url, nom, pwd) pour les bases de données sont

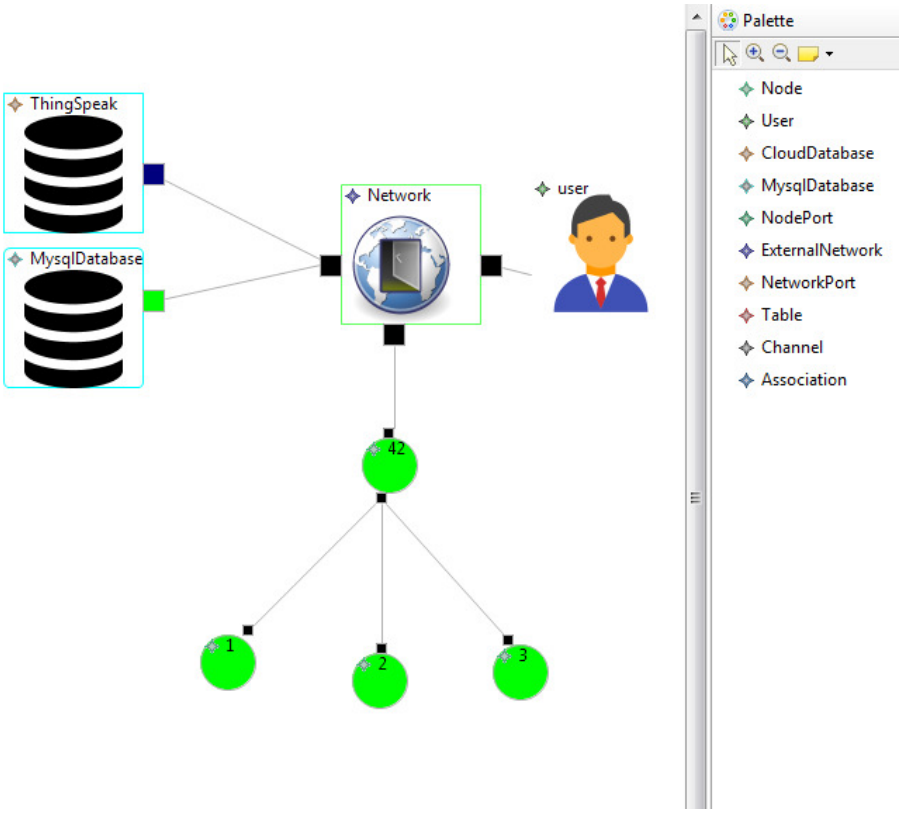


FIGURE 5.11 – La facette réseau de l'étude de cas.

spécifiées dans les éléments de cette facette. Elles seront utilisées dans les analyses et les simulations ainsi que dans le processus de génération de code.

5.3.3 Facette matérielle

La facette matérielle décrit les composants matériels existants dans chaque type de nœud. La Figure 5.12 illustre le nœud capteur ZigBee. Ce noeud est composé de capteurs de température/humidité (sht31[145]) et de luminosité (opt3001 [146]) qui sont connectés à l'unité de traitement par le bais des bus i2c. Il y a également un bloc d'alimentation avec deux piles *AAA* connectées à travers un bus ADC. Le nœud comporte également une unité de traitement (*pic18f25k42* [147]) et une unité de communication (ZigBee). Dans chaque unité, nous introduisons des informations telles que le coût et la consommation d'énergie. Ces informations permettent des estimations globales des coûts énergétiques et financiers.

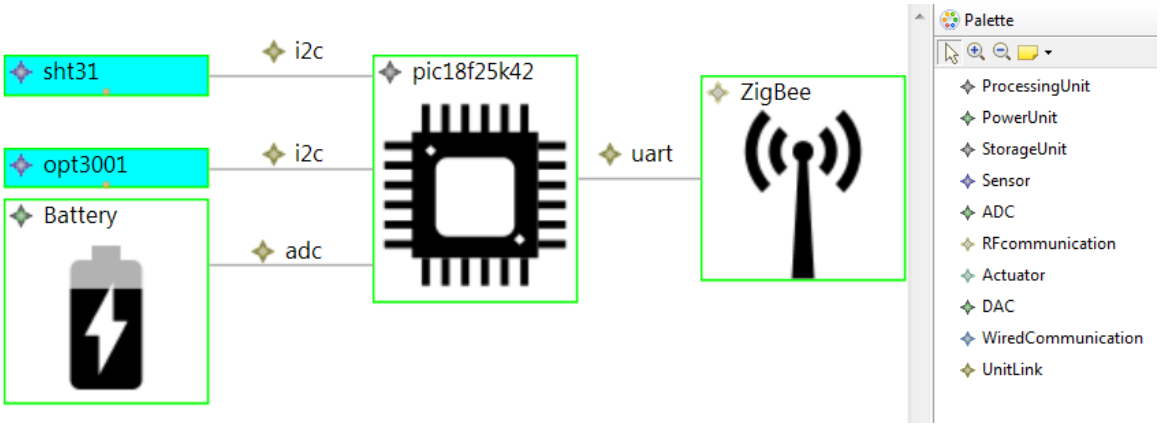


FIGURE 5.12 – La facette matérielle de l'étude de cas.

5.3.4 Facette logicielle

La facette logicielle décrit le comportement de notre noeud comme étant une machine d'état. La Figure 5.13 illustre la facette logicielle qui décrit le comportement de notre nœud capteur à travers plusieurs états. Dans les transitions entre les états, des conditions de transition et des fonctions déclenchées sont spécifiées. Si l'on considère le passage de l'état de veille à l'état *enable_sensors*, l'événement de transition est déclenché par un *timer* (tm (10min)) fixé à dix minutes. La fonction d'acquisition est ensuite déclenchée.

Il convient de noter que les fonctions appelées ou référencées dans les transitions sont identiques à celles modélisées dans la facette du flux de données.

5.3.5 Facette flux de données

La facette flux de données permet de décrire les flux et la structure des données dans un noeud. La Figure 5.14 illustre le flux de données dans notre nœud capteur. Nous avons trois ports d'acquisition similaires aux trois capteurs intégrés dans le nœud capteur. Chaque port possède un identifiant et il se caractérise par une fréquence d'acquisition (10min). Les ports sont connectés à des fonctions d'acquisition : get_sht31, get_opt3001, get_battery. Chaque fonction a des entrées et des sorties : les entrées sont les acquisitions et les sorties sont les mesures de température, d'humidité, de luminosité et du niveau de la batterie comme illustré dans la Figure 5.14.

Dans cette facette, on trouve des constantes (*NODE_SOF_VALUE, TEMPERATURE_ID_VALUE, HUMIDITY_ID_VALUE, LIGHT_ID_VALUE, VOLTAGE_ID_VALUE, NODE_EOF_VALUE*). Ces constantes permettent de définir la structure de la trame à envoyer vers la passerelle. Le paramètre *Node_ID* est aussi présent dans cette description. Ce noeud possède un port de transmission caractérisé par une fréquence d'émission (10 min). Les données issues des constantes, des paramètres et des fonctions forment la trame. La structure de la trame suivent l'ordre des ports d'entrées de la fonction d'encodage comme le montre la Figure 5.14. Par exemple, le premier champs de la trame est *NODE_SOF_VALUE* et le

FIGURE 5.13 – La facette logicielle de l'étude de cas.

dernier champs est *NODE_EOF_VALUE*.

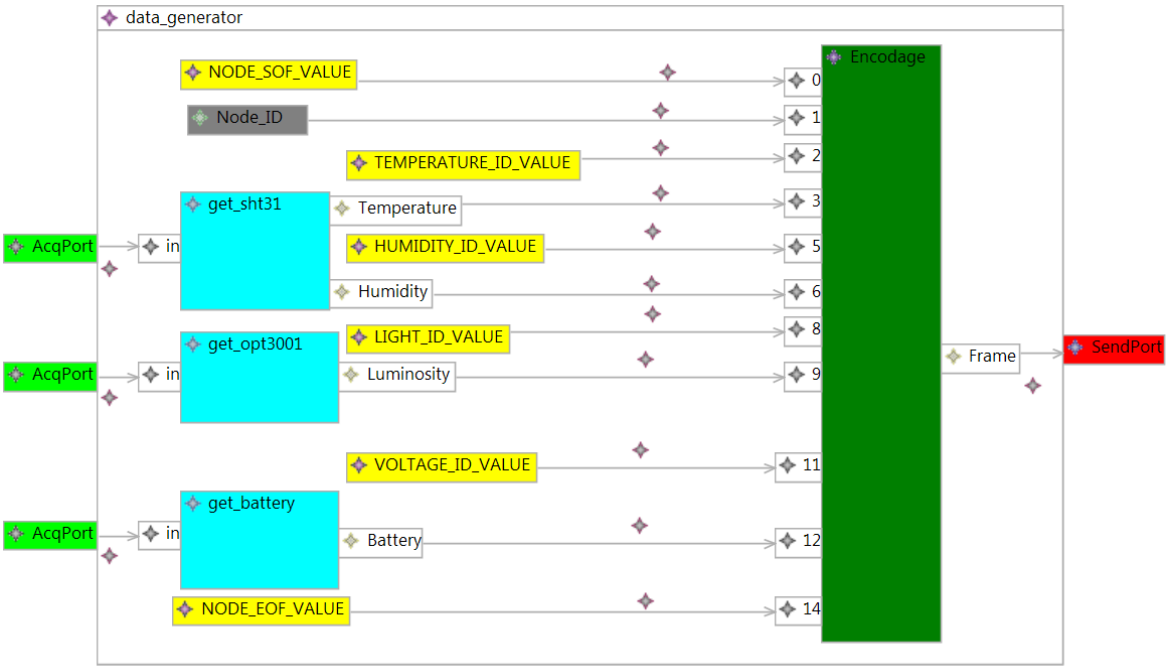


FIGURE 5.14 – La facette flux de données de l'étude de cas.

5.3.6 Facette environnement physique

La facette environnement physique peut être utilisée pour localiser et placer des nœuds dans l'environnement de déploiement. Il permet d'étudier la qualité des liens de communication entre les nœuds.

Dans notre étude de cas, nous avons identifié les zones par un numéro à 2 chiffres : le premier chiffre est le numéro du bâtiment et le second est l'étage. Par exemple, la zone 42 correspond au quatrième bâtiment et au deuxième étage.

Nous commençons par créer ces zones, puis les obstacles (murs) et les nœuds sont positionnés dans les zones. La Figure 5.15 illustre la zone 42 qui correspond au deuxième étage du quatrième bâtiment. Chaque obstacle se caractérise par ses dimensions, ses positions et ses paramètres d'atténuations.

Après avoir défini ces facettes, nous allons les utiliser pour :

— Dimensionner le RdC le mieux adapté à la problématique en se basant sur des simulations et des analyses.

— Générer du code source embarqué dans les nœuds du réseau à l'aide des techniques de transformations de modèles

— Configurer des interfaces d'exploitation de données.

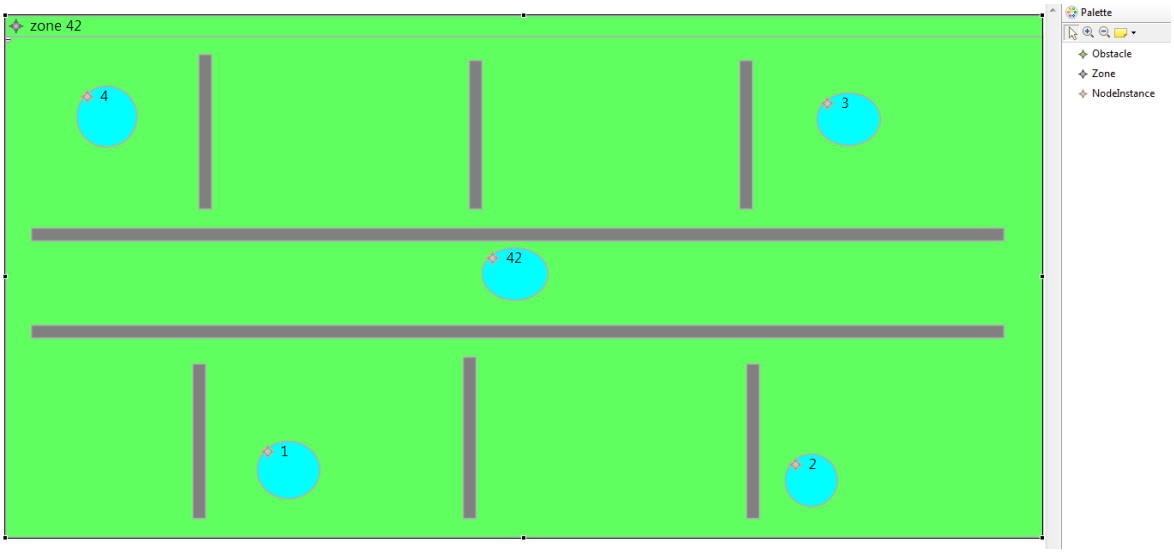


FIGURE 5.15 – La facette environnement physique de l'étude de cas : zone 42.

5.4 Dimensionnement du RdC par des analyses et des simulations

MODESENE définit des modèles de facettes qui peuvent être édités avec des transformations de modèles, nous pouvons avoir trois implémentations de ces transformations :

1. Des transformations de modèle vers modèle en utilisant le transformateur ATL,
2. Des transformations de modèle vers texte en utilisant le transformateur Acceleo,
3. Des transformations mixtes en utilisant des transformations à base de plugin disponible dans Eclipse.

Grâce à ces transformations, MODESENE supporte des analyses/simulations internes et/ou externes :

— Les analyses internes peuvent être développées à l'aide du concept de plugins d'Eclipse. Chaque plugin déclare ses dépendances, ses points d'extension et ses classes Java (à développer) dans un fichier de configuration XML. Une description détaillée de ces concepts est fournie dans le guide d'Eclipse [148].

— Les analyses externes peuvent être développées par la mise en place de passerelles entre notre Framework et des simulateurs.

Ci-dessous, deux exemples d'analyse internes pour l'estimation des coûts énergétiques et financiers et un exemple d'analyse externe pour l'évaluation des pertes de paquets :

5.4.1 Durée de vie du nœud capteur

Pour répondre aux exigences de l'application en termes d'autonomie, chaque nœud doit minimiser sa consommation énergétique. En utilisant le nœud capteur avec module

radio ZigBee, nous avons remarqué que l'étape d'envoi consomme une énergie importante par rapport aux étapes de mise en veille et d'acquisition (40 mA pour l'envoi, 100 uA en mode veille et 6,1mA en acquisition). En termes de durée d'exécution, la phase de veille dure plus longtemps que les autres phases. Pour cela, nous avons réduit le niveau de consommation des étapes de mise en veille et d'acquisition en changeant la source d'horloge externe (quartz) en source interne ainsi que l'optimisation du code source du microcontrôleur (Figure 5.16).

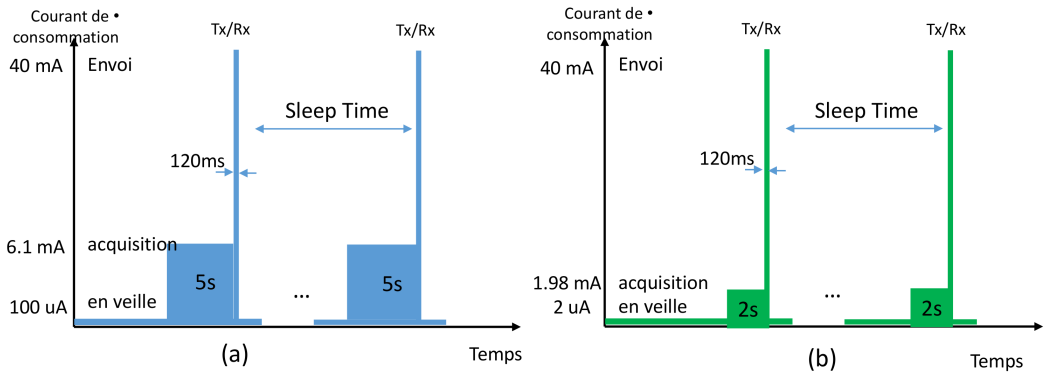


FIGURE 5.16 – La consommation d'énergie du nœud capteur avec un module de communication ZigBee : (a) ancienne version, (b) version optimisée.

Un nœud capteur est généralement équipé d'une source de batterie limitée. Dans certains cas, la reconstitution ou le remplacement des ressources énergétiques peut être coûteux, complexe ou impossible. La durée de vie du nœud est donc une caractéristique essentielle du RdC. Pour estimer la durée de vie des nœuds, une analyse est nécessaire afin de savoir quand il serait envisageable de remplacer la batterie. Quelques éléments nécessaires pour réaliser cette analyse sont énumérés dans le tableau 5.1, les valeurs des courants et des durées sont mesurées sur notre noeud capteur décrit dans la section 5.2.1.

TABLEAU 5.1 – Paramètres de simulation.

Parameters	**ZigBee**	**LoRa**
Nombre de nœuds	120	120
Courant consommé en mode : veille, acquisition, envoi	0.002mA, 1.98mA, 40mA	0.002mA, 1.98mA, 41mA
Durée exécutée en mode : veille, acquisition, envoi	600s, 2s, 0.23s	600s, 2s, 1.5s
Capacité de la batterie	600mAh	600mAh
Coût du capteur	20 euros	54 euros
Coût de la passerelle	80 euros	114 euros

En utilisant les niveaux de consommation d'énergie en mode veille, acquisition et envoi; la durée de vie des nœuds peut être estimée. La consommation d'énergie du nœud pour l'envoi d'un seul paquet est spécifiée, en normalisant par rapport à la tension d'alimentation, comme suit (E : énergie, C : courant, T : temps) :

$$E = C_{sleep} \times T_{sleep} + C_{sense} \times T_{sense} + C_{send} \times T_{send} \quad (5.1)$$

La Figure 5.17 estime la durée de vie des nœuds selon les technologies utilisées. Nous avons utilisé la technologie LoRa[149] et aussi des nœuds avec la technologie ZigBee [150]. Nous avons remarqué que les nœuds ZigBee ont une durée de vie plus longue du fait que ZigBee a un débit de données élevé et une faible consommation d'énergie comparé à la technologie LoRa qui a un faible débit (selon Libelium[149]). Un débit de données élevé permet de transmettre la même quantité de données en peu de temps par rapport à l'utilisation d'un débit de données faible. Un temps de transmission faible entraîne une faible consommation pour chaque cycle de transmission et donc une longue durée de vie.

Par ailleurs, il faut noter qu'il existe des émulateurs de certaines cartes capteurs qui peuvent fournir des estimations de consommation d'énergie plus précises comme l'émulateur PowerSim pour les plateformes basées TinyOS comme micaz[151]. Dans le cas où nous utilisons ce type de mote, nous pouvons développer une passerelle entre MODESENE et cet émulateur.

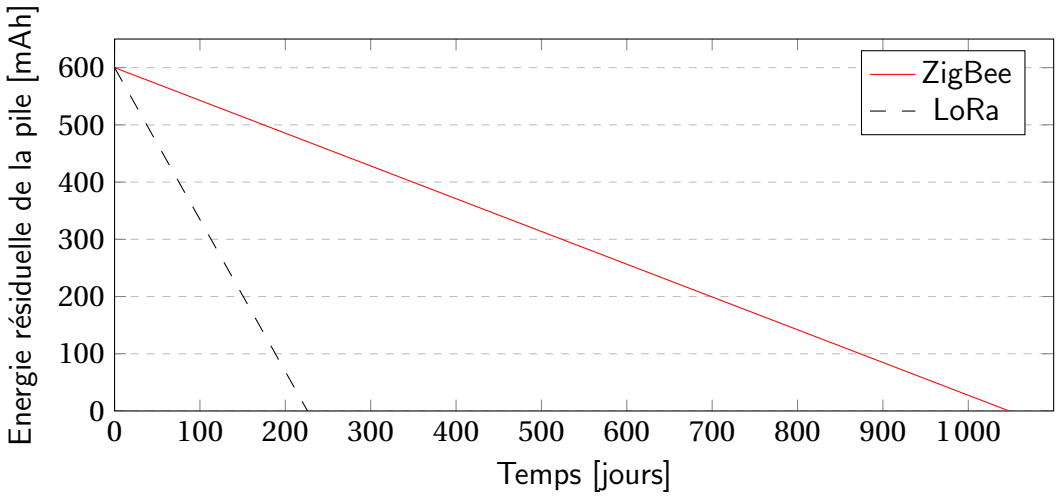


FIGURE 5.17 – Durée de vie des nœuds selon les technologies utilisées.

5.4.2 Analyse des coûts

Il est avantageux d'élaborer une approche qui permet d'estimer les coûts du réseau de capteurs. Les coûts du réseau de capteurs peuvent être regroupés en deux catégories : les coûts des plateformes et les coûts de fonctionnement. Pour notre cas d'étude, nous avons six bâtiments et chaque bâtiment a cinq étages avec quatre capteurs par étage. Par consé-

quent, le nombre total de capteurs est estimé à (5x4x6x6)=120. Pour la solution LoRa, nous avons besoin d'une seule passerelle, mais pour la solution ZigBee, il y a une passerelle à chaque étage, donc, nous avons besoin de 6x4=24 passerelles. Sur la base du coût des nœuds, les coûts des cartes peuvent être estimés à 4320 euros et 6594 euros pour les solutions ZigBee et LoRa, respectivement. Les coûts de réseau augmentent avec le temps en raison des coûts de fonctionnement.

La Figure 5.18 estime les coûts financiers du réseau. Au début du projet, nous avons remarqué que la solution LoRa coûte beaucoup plus cher que la solution ZigBee. Après cinq ans, la technologie LoRa coûte encore plus que la solution ZigBee en raison du fait que les nœuds LoRa consomment beaucoup plus d'énergie et par conséquent ont une faible durée de vie, ce qui implique des changements de batterie assez souvent.

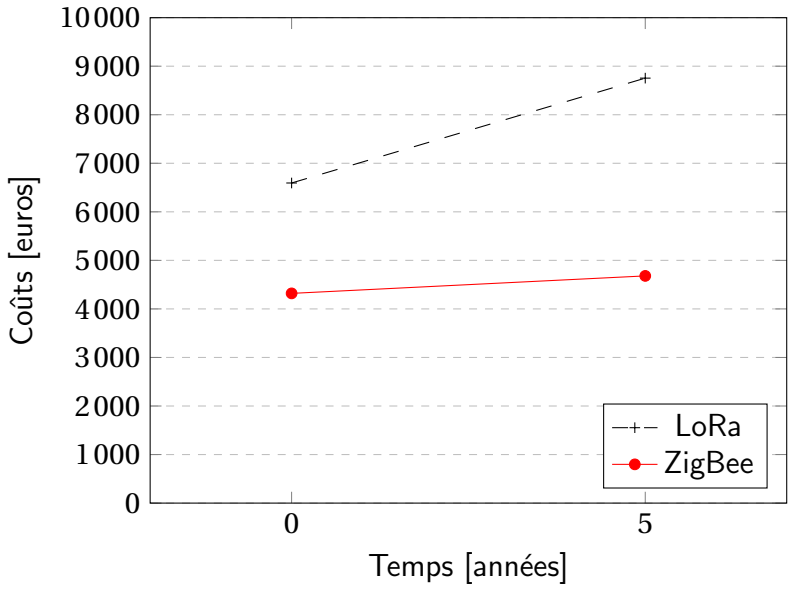


FIGURE 5.18 – Les coûts de réseau avec les deux solutions.

5.4.3 Perte de données

Dans cette section, nous présentons la simulation de perte de paquets en utilisant l'outil Omnet++ comme un exemple de simulateur cible. Omnet++ est un logiciel basé sur un environnement de simulation par événements discrets. Il dispose d'un framework *INET* qui est un outil de simulation de réseau de communication open source. Il convient aux simulations de différents types de réseaux filaires et sans fil. Un certain nombre de standards sont déjà disponibles dans ce framework (IEEE 802.11 et IEEE 802.15.4.). Il comprend également des modules de mobilité des nœuds et de mesure de la consommation d'énergie.

Nous avons réalisé des simulations avec le simulateur Omnet++ en générant un script de simulations. La Figure 5.19 illustre ce script. Il contient des informations liée à la facette d'environnement physique (les dimensions de la zone), la facette réseau (le nombre de

noeuds), la facette flux de donnée (la taille du message à envoyer) et à la facette matérielle (le débit de données).

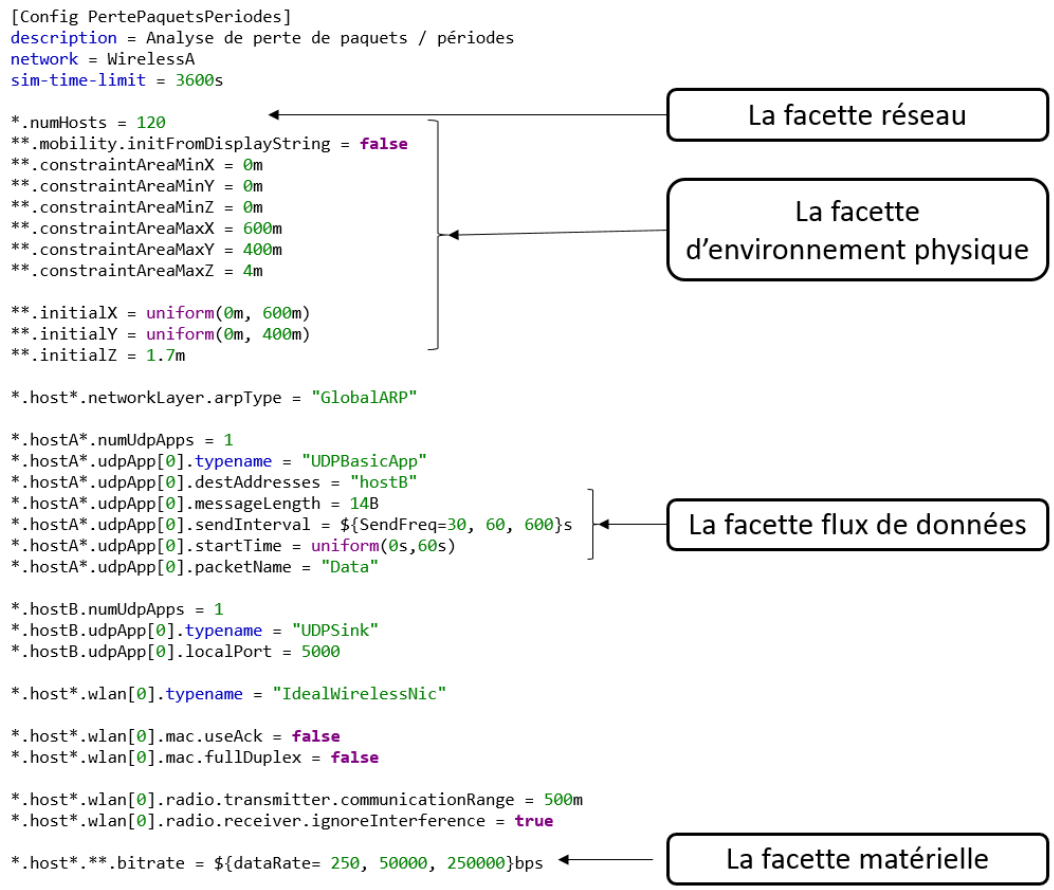


FIGURE 5.19 – Script de simulation pour le simulateur Omnet++.

La technologie LoRa offre la possibilité de sélectionner le mode de transmission en paramétrant la largeur de bande, le taux de codage et le facteur d'étalement. Il existe dix modes (mode1-10) qui définissent 10 débit. Le débit binaire maximal est de 50kbps pour le mode 10 [152] (qui permet d'avoir la plus courte portée), et 250bps pour le mode 1[152] (la plus longue portée) et huit autres modes intermédiaires.

La Figure 5.20 montre le taux des paquets reçus selon les débits utilisés : 250kbps, 50kbps et 250bps (ZigBee, LoRa mode 10, LoRa mode 1). Nous avons remarqué que le nombre de paquets perdus en utilisant la technologie ZigBee et LoRa mode 10 est négligeable comparé au mode 1 de LoRa. La technologie la plus rapide (mode 10) a moins de perte de paquets par rapport à la plus lente; ceci est lié au fait que la plus rapide envoie des données dans des courts intervalles de temps et donc moins de probabilité de collision par rapport à la plus lente (mode 1). En augmentant le temps d'envoi, les pertes de paquets diminuent en raison de la diminution de la probabilité de collision.

Dans cette section, nous avons étudié la différence entre LoRa et ZigBee en termes de débit. Nous sommes conscients que cette comparaison présente certaines limites. Ceci est principalement dû à l'absence d'un modèle complet dans l'outil Omnet++ qui prend

en compte toutes les couches LoRa et ZigBee (physique, MAC et réseau). A travers ces simulations, notre objectif est de vérifier principalement la faisabilité du développement de passerelles vers des outils de simulation.

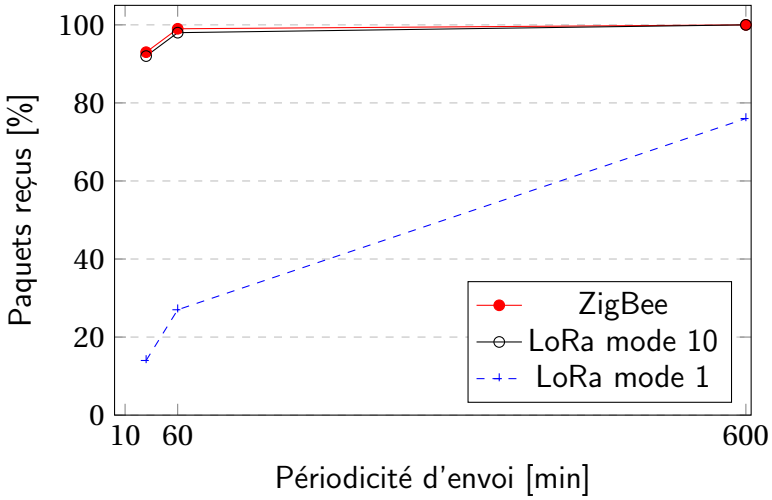


FIGURE 5.20 – Les paquets reçus en fonction de la vitesse de transmission des données.

Nous avons utilisé ces résultats pour guider nos choix de conception entre les deux solutions (LoRa, ZigBee). Nos exigences incluent une longue durée de vie, de faibles coûts financiers et de faibles taux de perte de paquets. Selon les résultats de l'ensemble des simulations, la solution ZigBee répond mieux aux exigences fixés.

5.5 Déploiement par génération automatique du code source

Nous avons intégré un plugin dans MODESENE qui permet de lire les facettes et de récupérer les informations nécessaires (les unités d'acquisition, les périodes d'acquisition et de transmission, l'unité de traitement (pic18f25k42), l'unité de communication (ZigBee). Ces informations sont ensuite utiliser pour générer un macrocode structuré en plusieurs sections.

La Figure 5.21 illustre le macrocode composé des bibliothèques spécifiques aux technologies, *ID* du nœud, la trame à envoyer, des variables et les fonctions excutées suivant un ordre défini. Nous trouvons des fonctions d'initialisation (comme init_sht31()) et aussi des fonctions cycliques (les fonctions entre les deux balises *do* et *while*).

Les fonctions représentées dans le macrocode sont combinées avec celles qui sont représentées dans des fichiers de définitions. La Figure 5.22 illustre un exemple d'un fichier de définition; dans cet exemple, chaque composant matériel possède une définition comme la définition du module de communication *ZigBee*). Chaque définition a un ensemble de fonctions; par exemple, la définition *ZigBee* possède quatre fonctions : *init_ZigBee*, *enable_ZigBee*, *disable_ZigBee*, *send_ZigBee*. Chaque fonction a des propriétés comme le nom de la fonction et le nom de la fonction implémenté dans des bibliothèques à fournir.

FIGURE 5.21 – Le macrocode pour la génération de code de notre nœud capteur.

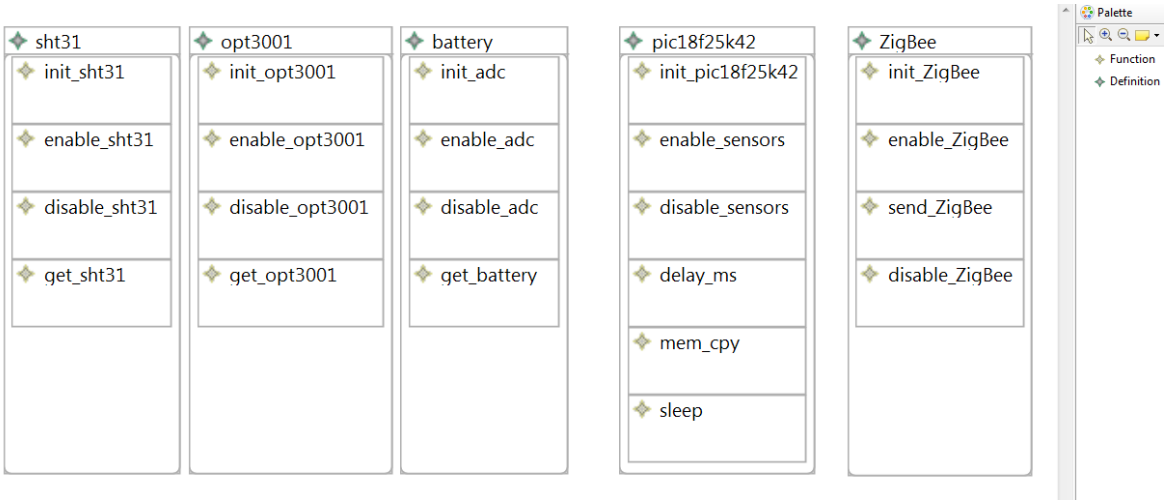


FIGURE 5.22 – Le fichier de définition pour être combiné avec le macrocode.

5.6 Exploitation de données et maintenance

Le cycle de vie d'un RdC comprend l'étape d'exploitation de données. Nous avons choisi Node-RED pour la mise en œuvre de cette étape [6].

Node-RED est un outil de programmation qui permet de connecter des périphériques matériels, des API et offre des services IoT. Il fournit une interface d'édition basée sur un navigateur qui facilite le câblage des flux en utilisant un ensemble de noeuds présents dans une palette. Il peut s'exécuter localement dans un ordinateur, Raspberry Pi ou à travers le cloud comme le propose IBM Cloud, SenseTecnic FRED, Amazon Web Services et Microsoft Azure.

Dans Node-RED, on décrit un modèle de flux basé sur des modèles de nœuds. Comme l'illustre la Figure 5.23, les nœuds sont enregistrés dans une palette qui est divisée en groupes tels que des entrées, des fonctions et des sorties (*dashboard*). Les nœuds d'entrée et de sortie varient entre des déclencheurs de flux et des types de connexion tels que TCP, WebSockets ou MQTT. Les nœuds de fonction permettent d'exécuter des opérations définies par l'utilisateur en Java-Script. il possède également des nœuds qui permettent de recevoir ou d'envoyer des messages par email ou par Twitter. Il dispose d'une bibliothèque incorporée permettant d'enregistrer des fonctions, modèles ou flux pour réutilisation. Les flux créés dans Node-RED sont stockés dans un fichier JSON qui peut être facilement importé et exporté.

Cet outil permet de réduire le temps de développement d'interface graphique en fournissant des composants graphiques de base comme des jauges et des histogrammes.

Nous avons développé des templates pour l'exploitation des données sous Node-RED. Grâce à la fonctionnalité d'importer et d'exporter des modèles implémentée dans Node-RED, nous pouvons modifier ces fichiers json suivant les informations des facettes pour générer des interfaces personnalisées.

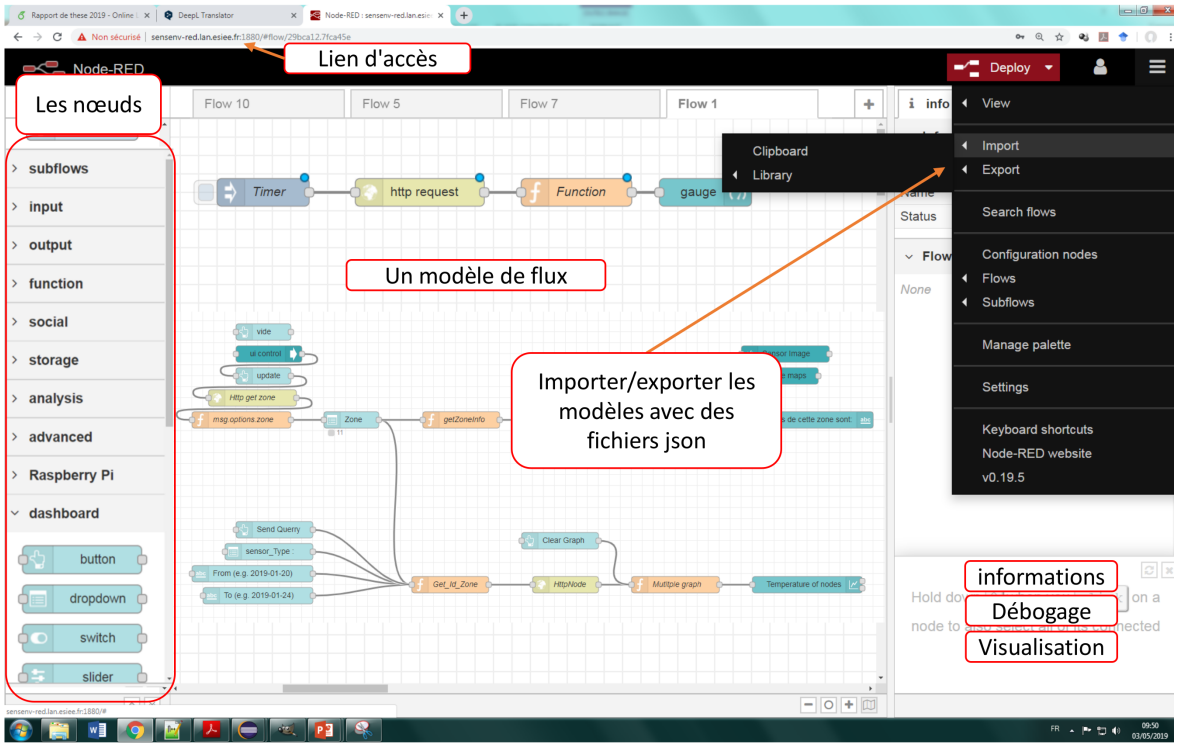


FIGURE 5.23 – Outil choisi pour l'exploitation de données : NodeRED.

5.6.1 Interface de visualisation

Nous avons développé une interface de visualisation sous Node-RED. Elle est illustrés dans la Figure 5.24 où nous nous focalisons sur la zone 33 qui regroupe deux capteurs 33.1 et 33.2. Dans cette interface, nous pouvons voir l'emplacement des capteurs dans la zone. Elle permet également un historique des mesures sur un intervalle de temps, ainsi que les dernières mesures sur un seul capteur (vue capteur). Dans cette interface nous pouvons changer la zone et aussi l'*ID* du nœud capteurs pour la visualisation de d'autres zones.

La vue capteur illustrée dans la partie droite de la Figure 5.24 est basée sur le modèle présenté dans la Figure 5.25. Nous avons généré ce modèle en utilisant des informations extraites à partir des facettes. Ces informations sont spécifiées dans la facette réseau (les informations liées à l'accès à la base de données) et dans la facette flux de données (l'ensemble des types de capteurs présents dans le noeud). Par exemple, si nous ajoutons un capteur de Co2 dans la facette flux de données, une transformations automatique permettra d'ajouter un élément graphique dans le modèle de la Figure 5.25.

FIGURE 5.24 – Interface sous Node-RED pour visualiser les données des capteurs par rapport à une zone.

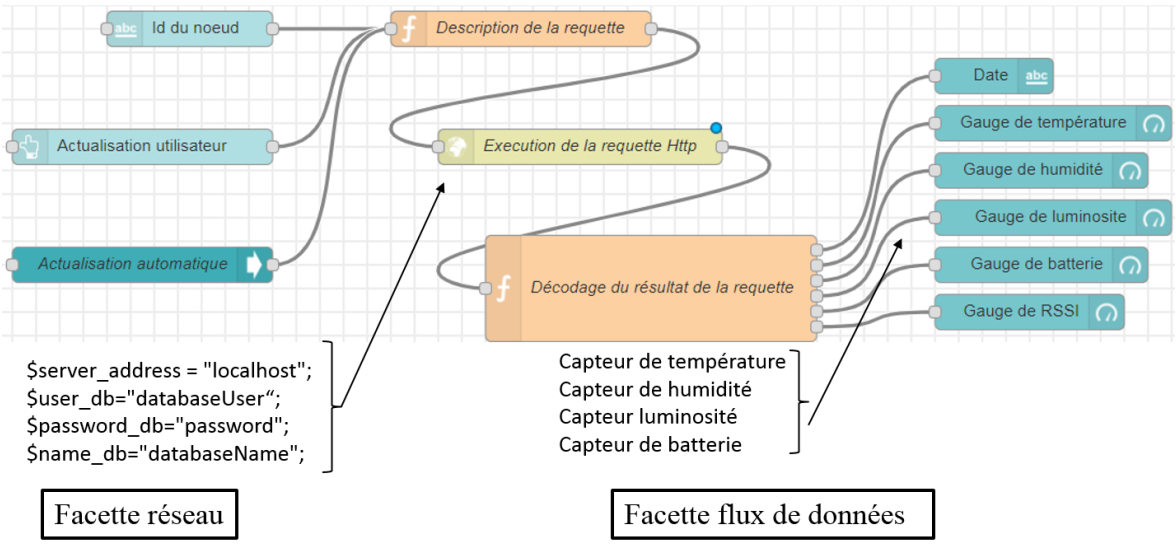


FIGURE 5.25 – Modèle de flux sous sous Node-RED qui permet de décrire la vue capteur.

5.6.2 Interface de notification

Pour assurer les fonctionnalités du réseau de capteurs, il faut développer aussi des interfaces de notification. La Figure 5.26 montre un exemple d'interface qui permet de notifier l'utilisateur si un nœud ou plusieurs n'ont pas envoyé des données après un intervalle de temps paramétrable. Dans ce contexte, nous avons aussi développé un service de notification si le niveau de la batterie est en dessous d'un certain seuil fixé par l'utilisateur. Ceci permet de remplacer les batteries et d'éviter des pertes de données.

Ces notifications peuvent être envoyées à distance par e-mail. L'adresse e-mail peut être indiquée par l'élément utilisateur dans la facette réseau. En utilisant les transformations de modèle, il est possible de configurer cette interface automatiquement pour envoyer ces notifications.

FIGURE 5.26 – Interface de notifications sous Node-RED.

5.7 Conclusion

Dans ce chapitre, nous avons mis en œuvre notre méthodologie sous forme d'un framework nommé MODESENE. Pour le valider, une application de bâtiment intelligent est utilisée comme étude de cas. Dans cette optique, nous avons conçu de nouvelles plateformes de nœuds capteurs et développé le logiciel associé.

A travers le cas d'étude, les facettes sont développées. Nous avons interfacé MODESENE avec un exemple de simulateur (Omnet++). Les résultats d'analyse/simulation sont

utilisées pour mieux dimensionner le réseau. Par rapport à notre cas d'étude, les résultats de simulations ont montré que la solution ZigBee est mieux adaptée à notre application par rapport à la solution LoRa.

Les transformations de modèle génèrent ensuite des codes sources pour les nœuds capteurs. Les transformations de modèle vers texte permettent de fournir des interfaces pour l'exploitation de données sous Node-RED.

Conclusion générale

Synthèse

Ce manuscrit présente les travaux réalisés au cours de ma thèse intitulée : *Ingénierie dirigée par les modèles pour la conception et la mise en œuvre des réseaux de capteurs.*

Nous avons d'abord présenté les concepts de base des réseaux de capteurs en introduisant les définitions nécessaires, les applications dans divers domaines, les caractéristiques des technologies de communication utilisées ainsi que les contraintes de conception des réseaux de capteurs. Nous avons constaté que le cycle de vie d'un réseau de capteur implique plusieurs étapes : le dimensionnement du réseau, la conception matérielle, la spécification du logiciel, l'analyse de performances, le déploiement du réseau, l'exploitation de données et la maintenance. Chaque étape est souvent décrite dans un outil dédié avec un langage métier. Les outils sont rarement interconnectés. Ceci implique des descriptions multiples tout au long des étapes du cycle de vie. Le temps passé sur ces descriptions devient significatif quand le nombre de noeuds devient important. Un changement dans le réseau doit être traduit dans chaque outil. Il est donc difficile d'assurer la cohérence entre les différentes descriptions. Il est également difficile de gérer et d'assurer la maintenance d'un réseau à grande échelle. Ceci entraîne souvent des erreurs de conception.

Nous avons dressé un état de l'art des travaux qui sont en lien avec cette problématique. Nous avons montré que les travaux existants se limitent seulement à une partie du cycle de vie du réseau de capteurs. De plus, ils présentent de nombreuses limites au niveau de l'expressivité, de la réutilisabilité et de la flexibilité des modèles.

Nous avons proposé une méthodologie basée sur l'ingénierie dirigée par les modèles pour faciliter la conception et la mise en œuvre des réseaux de capteurs. Elle vise à couvrir le cycle de vie d'un RdC en formalisant l'ensemble des concepts et des caractéristiques d'un RdC à l'aide d'un *métamodèle*. A partir de ce métamodèle, nous avons identifié un ensemble de facettes qui donne lieu à une approche *multi-facettes* couvrant l'ensemble des étapes du cycle de vie d'un RdC. Nous avons défini six facettes :

— La facette architecture réseau se focalise sur la composition et la topologie du réseau ;

— La facette architecture matérielle modélise les détails de bas niveau des différents

types de nœuds ;

— La facette logiciel décrit le comportement logiciel des nœuds ;

— La facette flux de données modélise les échanges de données entre les différents types de nœuds ;

— La facette environnement physique spécifie la zone de déploiement du réseau ;

— La facette projection établit des liens entre les cinq précédentes facettes pour formuler une solution possible d'un RdC à concevoir.

L'approche multi-facettes permet de faire face à la complexité en se basant sur la séparation des préoccupations (réseau, matériel, logiciel, flux de données, environnement physique). Nous avons porté une attention particulière aux modèles proposés afin que les modèles décrits dans chaque facette soient génériques et abstraits de toute technologies de mise en oeuvre. Ils sont ainsi réutilisables et permettent de décrire différentes architectures sans se limiter à des plateformes ou configurations particulières. Les facettes sont interconnectées grâce au métamodèle proposé. Moyennant des règles de transformations, tout changement dans une facette est répercuté automatiquement sur les autres facettes. Grâce aux transformations de modèles, nous générons automatiquement le macrocode des nœuds capteurs. Ce code intermédiaire est indépendant de toute technologie, il permet de générer le code source pour des cibles différentes. Cela permet de réduire le temps de développement et d'éviter les erreurs causées par les modifications manuelles.

Nous avons implémenté la méthodologie pour fournir un Framework d'aide à la conception des RdC. Ce Framework, dénommé MODESENE, est développé sous Eclipse GMF à l'aide des techniques de métamodélisation et de transformation de modèles. MODESENE offre un environnement d'édition des facettes qui implémente ainsi une séparation claire entre l'architecture réseau, l'architecture matérielle, l'architecture logicielle. Nous avons expérimenté et validé ce Framework à l'aide d'une application de mesure climatique de bâtiment. Ce cas d'étude déployé dans notre école *ESIEE Paris* a été décrit à l'aide de nos facettes. Nous avons développé une passerelle vers le simulateur Omnet++ pour estimer les performances du RdC. Par ailleurs, les transformations de modèle ont permis de générer le macrocode puis le code source des noeuds. Nous avons aussi utilisé les transformations de modèles pour créer des interfaces de visualisation et de notification sous l'outil Node-RED.

Travaux futurs

Les travaux proposés dans cette thèse ont posé les bases de la méthodologie et ont permis d'obtenir le socle d'un Framework d'aide à la conception des RdC. Des travaux futurs

peuvent être envisagés afin d'étendre et d'améliorer cette méthodologie et ce Framework. Les perspectives sont multiples et consistent principalement à :

— Étendre les langages de modélisation pour fournir des concepts et des éléments de descriptions supplémentaires.

  • La facette logicielle pourrait intégrer ou faire le lien avec des formalismes tels que le langage Statechart ou les composants logiciels ;

  • Lors de la génération de code, en fonction des cibles matérielles, il est nécessaire d'établir un lien avec les librairies spécifiques. Pour cela, il serait intéressant d'ajouter une description de l'environnement logiciel (librairies, système d'exploitation, chaîne de compilation, etc.) à la description de l'architecture matérielle ;

  • La modélisation du réseau pourrait être étendue avec de nouveaux éléments pour supporter une plus large gamme de topologie réseau (maillage complet, partiel, etc.) [153] ;

  • La modélisation de l'environnement physique pourrait être enrichie en prenant en charge des modèles standards de données du bâtiment comme par exemple la technologie Building Information Modeling (BIM) [154].

— Ajouter dans le Framework de nouveaux moteurs d'analyse ou de génération de code. En s'appuyant sur les outils existants, il serait possible de proposer des algorithmes d'aide au dimensionnement optimal du RdC en prenant en compte le positionnement des capteurs et leurs environnement physique afin de fournir le positionnement optimal des passerelles.

— Fournir de nouvelles facettes pour la modélisation des bases de données, pour la modélisation des interfaces de visualisation, et pour la spécification des contraintes de qualité de service, et pour de sécurité (cryptage, fiabilité, authentification, intégrité, confidentialité, etc.), ou encore, la modélisation des aspects liés à la maintenance (re-calibration ou reconfiguration de capteurs, chargement à distance du code embarqué, etc.).

— Enrichir la méthodologie à travers d'autres cas d'étude et d'expériences utilisateurs, cela permettrait de confronter l'approche à d'autres situations telles que la géolocalisation de capteurs, les réseaux IoT, etc [155].

Bibliographie

[1] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks : a survey. *Computer networks*, 38(4) :393–422, 2002.

[2] Dag Grini. Rf basics : Rf for non-rf engineers. Technical report, Texas Instruments Inc, 2006. `http://www.ti.com/lit/ml/slap127/slap127.pdf`. Accessed May 13, 2019.

[3] Marco Centenaro, Lorenzo Vangelista, Andrea Zanella, and Michele Zorzi. Long-range communications in unlicensed bands : The rising stars in the iot and smart city scenarios. *IEEE Wireless Communications*, 23(5) :60–67, 2016.

[4] Alberto Rodrigues da Silva. Model-driven engineering : A survey supported by the unified conceptual model. *Computer Languages, Systems and Structures*, 43 :139 – 155, 2015. ISSN 1477-8424. doi : https://doi.org/10.1016/j.cl.2015.06.001. URL `http://www.sciencedirect.com/science/article/pii/S1477842415000408`.

[5] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and . . . , 2008.

[6] Michael Blackstock and Rodger Lea. Toward a distributed data flow platform for the web of things (distributed node-red). In *Proceedings of the 5th International Workshop on Web of Things*, pages 34–39. ACM, 2014.

[7] Srisakdi Charmonman, Pornphisud Mongkhonvanit, V Ngoc Dieu, and N Linden. Applications of internet of things in e-learning. *International Journal of the Computer, the Internet and Management*, 23(3) :1–4, 2015.

[8] Syst Cisco. Fog computing and the internet of things : Extend the cloud to where the things are, 2016. `https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf`. Accessed June 13, 2018.

[9] Ian F. Akyildiz and Kasimoglu Ismail H. Wireless sensor and actor networks : research challenges. *Ad Hoc Networks*, 2(4) :351–367, 2004. ISSN 1570-8705. doi :

https:doi.org/10.1016/j.adhoc.2004.04.003. URL http:www.sciencedirect.com/science/article/pii/S1570870504000319.

[10] Ian F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks : a survey. *Computer Networks*, 38(4) :393 – 422, 2002. ISSN 1389-1286. doi : https://doi.org/10.1016/S1389-1286(01)00302-4. URL http://www.sciencedirect.com/science/article/pii/S1389128601003024.

[11] Geoffrey Werner-Allen, Konrad Lorincz, Mario Ruiz, Omar Marcillo, Jeff Johnson, Jonathan Lees, and Matt Welsh. Deploying a wireless sensor network on an active volcano. *IEEE internet computing*, 10(2) :18–25, 2006.

[12] Ian Johnstone, James Nicholson, Babar Shehzad, and Jeff Slipp. Experiences from a wireless sensor network deployment in a petroleum environment. In *Proceedings of the 2007 international conference on Wireless communications and mobile computing*, pages 382–387. ACM, 2007.

[13] Iuliu Vasilescu, Keith Kotay, Daniela Rus, Matthew Dunbabin, and Peter Corke. Data collection, storage, and retrieval with an underwater sensor network. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 154–165. ACM, 2005.

[14] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12) :2292 – 2330, 2008. ISSN 1389-1286. doi : https://doi.org/10.1016/j.comnet.2008.04.002. URL http://www.sciencedirect.com/science/article/pii/S1389128608001254.

[15] Wan Du. *Modeling and simulation of wireless sensor networks*. PhD thesis, Ecole Centrale de Lyon, 2011.

[16] Saber Amri, Fekher Khelifi, Abbas Bradai, Abderrezak Rachedi, Med Lassaad Kaddachi, and Mohamed Atri. A new fuzzy logic based node localization mechanism for wireless sensor networks. *Future Generation Computer Systems*, 2017.

[17] Safa Hamdoun, Abderrezak Rachedi, and Abderrahim Benslimane. Rssi-based localization algorithms using spatial diversity in wireless sensor networks. 2015.

[18] Ibtissem Boulanouar, Stéphane Lohier, Abderrezak Rachedi, and Gilles Roussel. Dta : Deployment and tracking algorithm in wireless multimedia sensor networks. *Ad hoc & sensor wireless networks*, 28(1-2) :115–135, 2015.

[19] Mario Di Francesco, Sajal K. Das, and Giuseppe Anastasi. Data collection in wireless sensor networks with mobile elements : A survey. *ACM Trans. Sen. Netw.*, 8(1) :7 :1–7 :31, August 2011. ISSN 1550-4859. doi : 10.1145/1993042.1993049. URL http://doi.acm.org/10.1145/1993042.1993049.

[20] Geoffrey Werner-Allen, Konrad Lorincz, Mario Ruiz, Omar Marcillo, Jeff Johnson, Jonathan Lees, and Matt Welsh. Deploying a wireless sensor network on an active volcano. *IEEE internet computing*, 10(2) :18–25, 2006.

[21] Moslem Amiri. *Measurements of energy consumption and execution time of different operations on Tmote Sky sensor nodes*. PhD thesis, Masarykova univerzita, Fakulta informatiky, 2010.

[22] Pei Zhang, Christopher M Sadler, Stephen A Lyon, and Margaret Martonosi. Hardware design experiences in zebranet. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 227–238. ACM, 2004.

[23] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *IEEE Communications magazine*, 40(8) :102–114, 2002.

[24] Ram Prasadh Narayanan, Thazath Veedu Sarath, and Vellora Veetil Vineeth. Survey on motes used in wireless sensor networks : Performance and parametric analysis. *Wireless Sensor Network*, 8(04) :51, 2016.

[25] Michael Johnson, Michael Healy, Pepijn van de Ven, Martin J Hayes, John Nelson, Thomas Newe, and Elfed Lewis. A comparative review of wireless sensor network mote technologies. In *SENSORS, 2009 IEEE*, pages 1439–1442. IEEE, 2009.

[26] Muhammad Omer Farooq and Thomas Kunz. Operating systems for wireless sensor networks : A survey. *Sensors*, 11(6) :5900–5930, 2011.

[27] Adi Mallikarjuna V Reddy, AVU Kumar, D Janakiram, and G Ashok Kumar. Wireless sensor network operating systems : a survey. *International Journal of Sensor Networks*, 5(4) :236–255, 2009.

[28] N Kurata, B Spencer Jr, M Ruiz-Sandoval, Y Miyamoto, and Y Sako. A study on building risk monitoring using wireless sensor network mica mote. *Strain*, 1 :35, 2003.

[29] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, et al. Tinyos : An operating system for sensor networks. In *Ambient intelligence*, pages 115–148. Springer, 2005.

[30] International Organization for Standardization/International Electrotechnical Commission et al. Information technology—open systems interconnection—basic reference model : The basic model. *ISO/IEC*, pages 7498–1, 1994.

[31] Noman Shabbir and Syed Rizwan Hassan. Routing protocols for wireless sensor networks (wsns). In *Wireless Sensor Networks-Insights and Innovations*. IntechOpen, 2017.

[32] Sandra M Hedetniemi, Stephen T Hedetniemi, and Arthur L Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4) :319–349, 1988.

[33] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd annual Hawaii international conference on system sciences*, pages 10–pp. IEEE, 2000.

[34] Kemal Akkaya and Mohamed Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3(3) :325 – 349, 2005. ISSN 1570-8705. doi : https://doi.org/10.1016/j.adhoc.2003.09.010. URL `http://www.sciencedirect.com/science/article/pii/S1570870503000738`.

[35] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1567–1576. IEEE, 2002.

[36] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107. ACM, 2004.

[37] Mubashir Husain Rehmani, Abderrezak Rachedi, Stéphane Lohier, Thierry Alves, and Benoit Poussot. Intelligent antenna selection decision in ieee 802.15. 4 wireless sensor networks : An experimental analysis. *Computers & Electrical Engineering*, 40 (2) :443–455, 2014.

[38] IPW Group et al. Part 15.4 : Low-rate wireless personal area networks (lr-wpans). *IEEE, IEEE Standard for Local and metropolitan area networks IEEE Std*, 802 : 4–2011, 2011. `http://ecee.colorado.edu/~liue/teaching/comm_standards/2015S_zigbee/802.15.4-2011.pdf`. Accessed June 12, 2019.

[39] Djamila Bendouda, Abderrezak Rachedi, and Hafid Haffaf. Programmable architecture based on software defined network for internet of things : connected dominated sets approach. *Future Generation Computer Systems*, 80 :188–197, 2018.

[40] Athar Ali Khan, Mubashir Husain Rehmani, and Abderrezak Rachedi. Cognitive-radio-based internet of things : Applications, architectures, spectrum related functionalities, and future research directions. *IEEE wireless communications*, 24(3) : 17–25, 2017.

[41] Abderrezak Rachedi, Mubashir Husain Rehmani, Soumaya Cherkaoui, and Joel JPC Rodrigues. Ieee access special section editorial : The plethora of research in internet of things (iot). *IEEE Access*, 4 :9575–9579, 2016.

[42] Geoff Mulligan. The 6lowpan architecture. In *Proceedings of the 4th workshop on Embedded networked sensors*, pages 78–82. ACM, 2007.

[43] Maha Bouaziz and Abderrezak Rachedi. A survey on mobility management protocols in wireless sensor networks based on 6lowpan technology. *Computer Communications*, 74 :3–15, 2016.

[44] Nurul Halimatul Asmak Ismail, Rosilah Hassan, and Khadijah WM Ghazali. A study on protocol stack in 6lowpan model. *Journal of Theoretical and Applied Information Technology*, 41(2) :220–229, 2012.

[45] Aminul Haque Chowdhury, Muhammad Ikram, Hyon-Soo Cha, Hassen Redwan, Shahiera Shams, Ki-Hyung Kim, and Seung-Wha Yoo. Route-over vs mesh-under routing in 6lowpan. In *Proceedings of the 2009 ACM International Wireless Communications and Mobile Computing Conference, IWCMC 2009*, pages 1208–1212, 01 2009. doi : 10.1145/1582379.1582643.

[46] Philippe Fabian, Abderrezak Rachedi, Cedric Gueguen, and Stephane Lohier. Fuzzy-based objective function for routing protocol in the internet of things. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[47] Ghada Glissa, Abderrezak Rachedi, and Aref Meddeb. A secure routing protocol based on rpl for internet of things. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2016.

[48] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. Low power wide area networks : An overview. *IEEE Communications Surveys & Tutorials*, 19(2) :855–873, 2017.

[49] Shahin Farahani. *ZigBee wireless networks and transceivers*. Newnes, 2011.

[50] Mourad Ouadou, Ouadoudi Zytoune, Yassin El Hillali, Atika Menhaj-Rivenq, and Driss Aboutajdine. Energy efficient hardware and improved cluster-tree topology for lifetime prolongation in zigbee sensor networks. *Journal of Sensor and Actuator Networks*, 6(4) :22, 2017.

[51] Bluetooth Special Interest Group. Specification of the bluetooth system, 2003. `https://inf.ethz.ch/personal/hvogt/proj/btmp3/Datasheets/Bluetooth_11_Specifications_Book.pdf`. Accessed June 12, 2019.

[52] Nick Baker. Zigbee and bluetooth : Strengths and weaknesses for industrial applications. *Computing and Control Engineering*, 16(2) :20–25, 2005.

[53] Vikethozo Tsira and Gypsy Nandi. Bluetooth technology : Security issues and its prevention. *Int. J. Computer Technology & Applications*, 5(5) :1833–1837, 2014.

[54] Theodoros Salonidis, Pravin Bhagwat, Leandros Tassiulas, and Richard LaMaire. Distributed topology construction of bluetooth personal area networks. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 3, pages 1577–1586. IEEE, 2001.

[55] Elke Mackensen, Matthias Lai, and Thomas M Wendt. Bluetooth low energy (ble) based wireless sensors. In *SENSORS, 2012 IEEE*, pages 1–4. IEEE, 2012.

[56] Seyed Darroudi and Carles Gomez. Bluetooth low energy mesh networks : A survey. *Sensors*, 17(7) :1467, 2017.

[57] Lorenzo Vangelista, Andrea Zanella, and Michele Zorzi. Long-range iot technologies : The dawn of lora™. In *Future access enablers of ubiquitous and intelligent infrastructures*, pages 51–58. Springer, 2015.

[58] Libelium Comunicaciones Distribuidas S.L. Waspmote lorawan networking guide. Technical report, Libelium, 2018. `http://www.libelium.com/development/waspmote/documentation/waspmote-lorawan-networking-guide/`. Accessed Feb 22, 2019.

[59] Pascal Stephany. Smockeo : Le détecteur de fumée connecté sigfox, 2019. `http://www.domotique-info.fr/2016/04/smockeo-detecteur-de-fumee-connecte-sigfox/`. Accessed Feb 22, 2019.

[60] Maha Bouaziz, Abderrezak Rachedi, Abdelfettah Belghith, Marion Berbineau, and Saad Al-Ahmadi. Ema-rpl : Energy and mobility aware routing for the internet of mobile things. *Future Generation Computer Systems*, 97 :247–258, 2019.

[61] Maha Bouaziz, Abderrezak Rachedi, and Abdelfettah Belghith. Ekf-mrpl : Advanced mobility support routing protocol for internet of mobile things : Movement prediction approach. *Future Generation Computer Systems*, 2017.

[62] Donggang Liu and Peng Ning. *Security for wireless sensor networks*, volume 28. Springer Science and Business Media, 2007.

[63] Abderrezak Rachedi and Amina Hasnaoui. Advanced quality of services with security integration in wireless sensor networks. *Wireless Communications and Mobile Computing*, 15(6) :1106–1116, 2015.

[64] H. Moungla, K. Haddadi, and S. Boudjit. Distributed interference management in medical wireless sensor networks. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 151–155, Jan 2016. doi : 10.1109/CCNC.2016.7444748.

[65] Naresh Kumar, AS Zadgaonkar, and Abhinav Shukla. Evolving a new software development life cycle model sdlc-2013 with client satisfaction. *International Journal of Soft Computing and Engineering (IJSCE)*, 3(1) :2231–2307, 2013.

[66] Abbas Bradai, Kamal Singh, Abderrezak Rachedi, and Toufik Ahmed. Emcos : Energy-efficient mechanism for multimedia streaming over cognitive radio sensor networks. *Pervasive and Mobile Computing*, 22 :16–32, 2015.

[67] Kaveh Pahlavan and Prashant Krishnamurthy. *Networking fundamentals : Wide, local and personal area communications*. John Wiley and Sons, 2009.

[68] Vibha Prasad and Sang H Son. Classification of analysis techniques for wireless sensor networks. In *2007 Fourth International Conference on Networked Sensing Systems*, pages 93–97. IEEE, 2007.

[69] Vlado Handziski, Andreas Köpke, Andreas Willig, and Adam Wolisz. Twist : a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks : from theory to reality*, pages 63–70. ACM, 2006.

[70] Esteban Egea-Lopez, Javier Vales-Alonso, Alejandro S Martinez-Sala, Pablo Pavon-Marino, and Joang García-Haro. Simulation tools for wireless sensor networks. In *proceedings of the international symposium on performance evaluation of computer and telecommunication systems (SPECTS05)*, page 24, 2005.

[71] Gilbert Chen, Joel Branch, Michael Pflug, Lijuan Zhu, and Boleslaw Szymanski. Sense : a wireless sensor network simulator. In *Advances in pervasive computing and networking*, pages 249–267. Springer, 2005.

[72] Ludovic Samper. *Modélisations et analyses de réseaux de capteurs*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2008.

[73] Hossam Mahmoud Ahmad Fahmy. *Simulators and Emulators for WSNs*, pages 381–491. Springer Singapore, Singapore, 2016. ISBN 978-981-10-0412-4. doi : 10.1007/978-981-10-0412-4_6. URL `https://doi.org/10.1007/978-981-10-0412-4_6`.

[74] Muhammad Imran, Abas Md Said, and Halabi Hasbullah. A survey of simulators, emulators and testbeds for wireless sensor networks. In *2010 International Symposium on Information Technology*, volume 2, pages 897–902. IEEE, 2010.

[75] Ben L Titzer, Daniel K Lee, and Jens Palsberg. Avrora : Scalable sensor network simulation with precise timing. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 477–482. IEEE, 2005.

[76] Teerawat Issariyakul and Ekram Hossain. Introduction to network simulator 2 (ns2). In *Introduction to network simulator NS2*, pages 1–18. Springer, 2009.

[77] Antoine Fraboulet, Guillaume Chelius, and Eric Fleury. Worldsens : development and prototyping tools for application specific wireless sensors networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 176–185. ACM, 2007.

[78] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim : a library for parallel simulation of large-scale wireless networks. In *Proceedings. Twelfth Workshop on Parallel and Distributed Simulation PADS'98 (Cat. No. 98TB100233)*, pages 154–161. IEEE, 1998.

[79] Gyula Simon, Peter Volgyesi, Miklós Maróti, and Akos Ledeczi. Simulation-based optimization of communication protocols for large-scale wireless sensor networks. In *IEEE aerospace conference*, volume 3, pages 31339–31346, 2003.

[80] TK Tan, A Raghunathan, and Niraj Kumar Jha. Emsim : An energy simulation framework for an embedded operating system. In *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No. 02CH37353)*, volume 2, pages II–II. IEEE, 2002.

[81] Maneesh Varshney, Defeng Xu, M Srivastava, and R Bagrodia. squalnet : An accurate and scalable evaluation framework for sensor networks. In *Information Processing in Sensor Networks*, 2007.

[82] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim : Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137. ACM, 2003.

[83] Jonathan Polley, Dionysus Blazakis, Jonathan McGee, Daniel Rusk, and John S Baras. Atemu : a fine-grained sensor network simulator. In *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.*, pages 145–152. IEEE, 2004.

[84] Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with cooja. In *First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, 2006.

[85] Mauri Kuorilehto, Marko Hännikäinen, and Timo D Hämäläinen. Rapid design and evaluation framework for wireless sensor networks. *Ad Hoc Networks*, 6(6) :909–935, 2008.

[86] Jeff Hiner, Ashish Shenoy, Roman Lysecky, Susan Lysecky, and Ann Gordon Ross. Transaction-level modeling for sensor networks using systemc. In *2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, pages 197–204. IEEE, 2010.

[87] Franco Fummi, Davide Quaglia, and Francesco Stefanni. A systemc-based framework for modeling and simulation of networked embedded systems. In *2008 Forum on Specification, Verification and Design Languages*, pages 49–54. IEEE, 2008.

[88] Stephen J Mellor, Tony Clark, and Takao Futagami. Model-driven development : guest editors' introduction. *IEEE software*, 20(5) :14–18, 2003.

[89] Object Management Group. Industrial internet reference architecture faq, 2019. `https://www.omg.org/index.htm`. Accessed June 15, 2019.

[90] Pierre-André Caron. *Ingénierie dirigée par les modèles pour la construction de dispositifs pédagogiques sur des plateformes de formation.* PhD thesis, Université des Sciences et Technologie de Lille-Lille I, 2007.

[91] Jean Bézivin and Olivier Gerbé. Towards a precise definition of the omg/mda framework. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pages 273–280. IEEE, 2001.

[92] Edwin Seidewitz. What models mean. *IEEE software*, 20(5) :26–32, 2003.

[93] Benoît Combemale. Ingénierie dirigée par les modèles (idm)–état de l'art, 2008. `https://hal.archives-ouvertes.fr/hal-00371565/file/mde-stateoftheart.pdf`. Accessed June 14, 2019.

[94] Raphaël Marvie. *Séparation des préoccupations et méta-modélisation pour environnements de manipulation d'architectures logicielles à base de composants.* PhD thesis, Université des Sciences et Technologie de Lille-Lille I, 2002.

[95] Object Management Group. Omg meta object facility (mof) specification v1. 4, 2002.

[96] Jean Bézivin. On the unification power of models. *Software & Systems Modeling*, 4 (2) :171–188, 2005.

[97] Deniz Cetinkaya, Alexander Verbraeck, and Mamadou D Seck. Mdd4ms : a model driven development framework for modeling and simulation. In *Proceedings of the 2011 summer computer simulation conference*, pages 113–121. Society for Modeling & Simulation International, 2011.

[98] Frank Truyen. The fast guide to model driven architecture the basics of model driven architecture. *Cephas Consulting Corp*, 2006.

[99] Sébastien Demathieu, Frédéric Thomas, Charles André, Sébastien Gérard, and François Terrier. First experiments using the uml profile for marte. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 50–57. IEEE, 2008.

[100] Peter Swithinbank, Mandy Chessell, Tracy Gardner, Catherine Griffin, Jessica Man, Helen Wylie, and Larry Yusuf. *Patterns : Model-Driven Development Using IBM Rational Software Architect*. IBM, International Technical Support Organization, 2005.

[101] Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. The generic modeling environment. In *Workshop on Intelligent Signal Processing, Budapest, Hungary*, volume 17, page 1, 2001.

[102] Eran Gery, David Harel, and Eldad Palachi. Rhapsody : A complete life-cycle model-based development system. In *International Conference on Integrated Formal Methods*, pages 1–10. Springer, 2002.

[103] Kari Smolander, Kalle Lyytinen, Veli-Pekka Tahvanainen, and Pentti Marttiin. Metaedit—a flexible graphical environment for methodology modelling. In *International Conference on Advanced Information Systems Engineering*, pages 168–193. Springer, 1991.

[104] Etienne Juliot and Jérôme Benois. Viewpoints creation using obeo designer or how to build eclipse dsm without being an expert developer. *Obeo Designer Whitepaper*, 2010.

[105] Ed Merks, R Eliersick, T Grose, F Budinsky, and D Steinberg. The eclipse modeling framework. *retrieved from, total*, page 37, 2003.

[106] Martin Fowler. Language workbenches : The killer-app for domain specific languages, 2005. `http://www.cime.cl/archivos/ILI253/8870_cl2-MartinFowler-Language-Workbench-DSL.pdf`. Accessed May 01, 2019.

[107] Zoé Drey Marie Gouyette François Tanguy, Didier Vojtisek. Chapter 2. getting started with gmf, 2010. `http://www.kermeta.org/docs/fr.irisa.triskell.kermeta.samples.fsm.documentation/build/html.chunked/KerMeta-Create-FSM-Graphical-Editor-With-GMF/ch02.html`. Accessed May 01, 2019.

[108] Eclipse Foundation Inc. Atl - a model transformation technology, 2019. `https://www.eclipse.org/atl/`. Accessed May 01, 2019.

[109] Eclipse Foundation Inc. Generate anything from any emf model, 2019. `https://www.eclipse.org/acceleo/`. Accessed May 01, 2019.

[110] Philip Baldwin, Sanjeev Kohli, Edward A Lee, Xiaojun Liu, and Yang Zhao. Modeling of sensor nets in ptolemy ii. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 359–368. ACM, 2004.

[111] Cristina Vicente-Chicote, Fernando Losilla, Bárbara Álvarez, Andrés Iborra, and Pedro Sánchez. Applying mde to the development of flexible and reusable wireless sensor networks. *International Journal of Cooperative Information Systems*, 16 (03n04) :393–412, 2007.

[112] Mohammad Mostafizur Rahman Mozumdar, Francesco Gregoretti, Luciano Lavagno, Laura Vanzago, and Stefano Olivieri. A framework for modeling, simulation and automatic code generation of sensor network application. In *2008 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 515–522. IEEE, 2008.

[113] Bahar Akbal-Delibas, Pruet Boonma, and Junichi Suzuki. Extensible and precise modeling for wireless sensor networks. In *International United Information Systems Conference*, pages 551–562. Springer, 2009.

[114] Gerhard Fuchs and Reinhard German. Uml2 activity diagram based programming of wireless sensor networks. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications*, pages 8–13. ACM, 2010.

[115] Pruet Boonma and Junichi Suzuki. Model-driven performance engineering for wireless sensor networks with feature modeling and event calculus. In *Proceedings of the 3rd workshop on Biologically inspired algorithms for distributed systems*, pages 17–24. ACM, 2011.

[116] Taniro Rodrigues, Priscilla Dantas, Paulo F Pires, Luci Pirmez, Thais Batista, Claudio Miceli, Albert Zomaya, et al. Model-driven development of wireless sensor network applications. In *2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing*, pages 11–18. IEEE, 2011.

[117] Yann Ben Maissa, Fabrice Kordon, Salma Mouline, and Yann Thierry-Mieg. Modeling and analyzing wireless sensor networks with verisensor : An integrated workflow. In *Transactions on Petri Nets and Other Models of Concurrency VIII*, pages 24–47. Springer, 2013.

[118] Krishna Doddapaneni, Enver Ever, Orhan Gemikonakli, Ivano Malavolta, Leonardo Mostarda, and Henry Muccini. A model-driven engineering framework for architecting and analysing wireless sensor networks. In *Proceedings of the Third International Workshop on Software Engineering for Sensor Network Applications*, pages 1–7. IEEE Press, 2012.

[119] Baobing Wang and John S Baras. Integrated modeling and simulation framework for wireless sensor networks. In *2012 IEEE 21st International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises*, pages 268–273. IEEE, 2012.

[120] Priscilla Dantas, Taniro Rodrigues, Thais Batista, Flavia C Delicato, Paulo F Pires, Wei Li, and Albert Y Zomaya. Lwissy : A domain specific language to model wireless sensor and actuators network systems. In *2013 4th International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pages 7–12. IEEE, 2013.

[121] John Khalil Jacoub. *Designing power aware wireless sensor networks leveraging software modeling techniques*. PhD thesis, University of Ontario : Institute of Technology, 2014.

[122] Kent I. F. Simonsen A. Ajith Kumar S. Towards a model-based development approach for wireless sensor-actuator network protocols. In *Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*, pages 35–39. ACM, 2014.

[123] Leandro Buss Becker, Fabio Paulo Basso, Antônio Augusto Fröhlich, and AR Paulon. Model-driven development of wsn applications. In *2013 III Brazilian Symposium on Computing Systems Engineering*, pages 161–166. IEEE, 2013.

[124] Emad Ebeid, Franco Fummi, and Davide Quaglia. Model-driven design of network aspects of distributed embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(4) :603–614, 2015.

[125] Nguyen Xuan Thang. *Model-driven development of sensor network applications with optimization of non-functional constraints*. PhD thesis, University of Kassel, Germany, 2016.

[126] Kenji Tei, Ryo Shimizu, Yoshiaki Fukazawa, and Shinichi Honiden. Model-driven-development-based stepwise software development process for wireless sensor networks. *IEEE Transactions on Systems, Man, and Cybernetics : Systems*, 45(4) : 675–687, 2015.

[127] Carlos Vidal, Carlos Fernández-Sánchez, Jessica Díaz, and Jennifer Pérez. A model-driven engineering process for autonomic sensor-actuator networks. *International Journal of Distributed Sensor Networks*, 11(3) :684892, 2015.

[128] Aymen J Salman and Adil Al-Yasiri. Sennet : a programming toolkit to develop wireless sensor network applications. In *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–7. IEEE, 2016.

[129] Elaine Cheong, Edward A Lee, and Yang Zhao. Viptos : a graphical development and simulation environment for tinyos-based wireless sensor networks. In *SenSys*, volume 5, pages 302–302, 2005.

[130] Mubashir Husain Rehmani, Martin Reisslein, Abderrezak Rachedi, Melike Erol-Kantarci, and Milena Radenkovic. Guest editorial special section on smart grid and renewable energy resources : Information and communication technologies with industry perspective. *IEEE Transactions on Industrial Informatics*, 13(6) :3119–3123, 2017.

[131] Jörg Kienzle, Wisam Al Abed, and Jacques Klein. Aspect-oriented multi-view modeling. In *Proceedings of the 8th ACM international conference on Aspect-oriented software development*, pages 87–98. ACM, 2009.

[132] Aditya A Shah, Aleksandr A Kerzhner, Dirk Schaefer, and Christiaan JJ Paredis. Multi-view modeling to support embedded systems engineering in sysml. In *Graph transformations and model-driven engineering*, pages 580–601. Springer, 2010.

[133] Mubashir Husain Rehmani, Martin Reisslein, Abderrezak Rachedi, Melike Erol-Kantarci, and Milena Radenkovic. Integrating renewable energy resources into the smart grid : Recent developments in information and communication technologies. *IEEE Transactions on Industrial Informatics*, 14(7) :2814–2825, 2018.

[134] Mubashir Husain Rehmani, Ayaz Ahmad, Abderrezak Rachedi, Soumaya Cherkaoui, and Kok-Lim Alvin Yau. Ieee access special section editorial : Energy harvesting and scavenging : Technologies, algorithms, and communication protocols. *IEEE Access*, 6 :13461–13465, 2018.

[135] Minhaj Ahmad Khan, Tariq Umer, Samee U Khan, Shui Yu, and Abderrezak Rachedi. Ieee access special section editorial : green cloud and fog computing : energy efficiency and sustainability aware infrastructures, protocols, and applications. *IEEE Access*, 6 :12280–12283, 2018.

[136] Caleb Phillips, Douglas Sicker, and Dirk Grunwald. A survey of wireless path loss prediction and coverage mapping methods. *IEEE Communications Surveys & Tutorials*, 15(1) :255–270, 2012.

[137] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1) : 22–32, 2014.

[138] Deborah Snoonian. Smart buildings. *IEEE spectrum*, 40(8) :18–23, 2003.

[139] Priyanka Rawat, Kamal Deep Singh, Hakima Chaouchi, and Jean Marie Bonnin. Wireless sensor networks : a survey on recent developments and potential synergies. *The Journal of supercomputing*, 68(1) :1–48, 2014.

[140] Microchip Technology Inc. Microchip technology (2009) pic18f2480/2580/4480/4580 data sheet, 28/40/44-pin enhanced flash microcontrollers with ecan™ technology. Technical report, Microchip., 2009. http://ww1.microchip.com/downloads/en/DeviceDoc/39637d.pdf. Accessed 30 January 2018.

[141] Digi International Singapore. Digi (2017) digi xbee s2c 802.15.4 rf modules. Technical report, Digi., 2017. https://www.digi.com/pdf/ds_xbee-s2c-802-15-4.pdf. Accessed 30 January 2018.

[142] Semtech Corporation. Wireless and sensing products : Sx1272/73 – 860 mhz to 1020 mhz low power long range transceiver. Technical report, Semtech Corporation ., 2017. https://www.semtech.com/uploads/documents/sx1272.pdf. Accessed 30 January 2018.

[143] Congduc Pham. Low-power, long-range wan for iot : A technology overview. Technical report, Université de Pau et des Pays de l'Adour (UPPA)., 2015. http://cpham.perso.univ-pau.fr/Paper/Talk-Rescom-16-LPWAN-review.pdf. Accessed 30 January 2018.

[144] Sharmad Pasha. Thingspeak based sensing and monitoring system for iot with matlab analysis. *International Journal of New Technology and Research*, 2(6), 2016.

[145] Sensirion AG Switzerland. Datasheet sht3x-dis. Technical Report 3, Sensirion AG Switzerland, 2016. http://www.mouser.com/ds/2/682/Sensirion_Humidity_Sensors_SHT3x_Datasheet_digital-971521.pdf. Accessed May 25, 2019.

[146] Texas instrument inc. Opt3001 ambient light sensor (als). Technical report, Texas Instrument Inc, 2017. http://www.ti.com/lit/ds/symlink/opt3001.pdf. Accessed May 25, 2019.

[147] Microchip Technology Inc. Pic18(l)f24/25k42 : 28-pin, low-power, high-performance microcontrollers with xlp technology. Technical report, Microchip Technology Inc., 2016. http://ww1.microchip.com/downloads/en/devicedoc/40001869b.pdf. Accessed May 25, 2019.

[148] Eclipse Foundation Inc. Eclipse documentation archived release, eclipse kepler, 2019. https://help.eclipse.org/kepler/index.jsp. Accessed May 11, 2019.

[149] Libelium. Waspmote-lora-868mhz 915mhz-sx1272 : Networking guide. Technical Report v7.0, Libelium Comunicaciones Distribuidas S.L., 2017.

[150] International Digi. Digi xbee s2c 802.15.4 rf modules. Technical report, Digi international singapore, 2018. https://www.digi.com/pdf/ds_xbee-s2c-802-15-4.pdf. Accessed June 13, 2018.

[151] Technology Crossbow. micaz : wireless measurement system, 2007. `http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf`. Accessed June 13, 2018.

[152] Jonas Skog Andersen and Joakim Eriksson. *Investigating the practical performance of the LoRaWAN technology*. PhD thesis, Linköping University, 2017.

[153] Mubashir Husain Rehmani, Abderrezak Rachedi, Melike Erol-Kantarci, Milena Radenkovic, and Martin Reisslein. Cognitive radio based smart grid : The future of the traditional electrical grid. *Ad Hoc Networks*, 100(41) :1–4, 2016.

[154] Salman Azhar. Building information modeling (bim) : Trends, benefits, risks, and challenges for the aec industry. *Leadership and management in engineering*, 11(3) : 241–252, 2011.

[155] Mubashir Husain Rehmani, Melike Erol Kantarci, Abderrezak Rachedi, Milena Radenkovic, and Martin Reisslein. Ieee access special section editorial smart grids : A hub of interdisciplinary research. *IEEE Access*, 3 :3114–3118, 2015.

Résumé : *Ingénierie dirigée par les modèles pour la conception et la mise en oeuvre des réseaux de capteurs*

Le cycle de vie d'un réseau de capteurs implique plusieurs étapes, telles que le dimensionnement du réseau, le développement de logiciels embarqués, la réalisation de matériels, des analyses et des simulations, le déploiement physique, l'exploitation des données et la maintenance du réseau. Tout au long de ce cycle, plusieurs outils spécifiques sont utilisés. Il est donc nécessaire de décrire manuellement les caractéristiques du réseau de capteurs dans chacun de ces outils en utilisant leurs propres langages. Il en résulte des coûts de développement importants et éventuellement des incohérences entre les différentes descriptions.

Pour répondre à cette problématique, nous proposons une méthodologie basée sur l'approche MDE (Model Driven Engineering). Elle vise à couvrir toutes les étapes du cycle de vie d'un réseau de capteurs. Tous les concepts et les caractéristiques d'un réseau de capteurs sont décrits à l'aide d'un métamodèle. A partir de ce référentiel, il est possible d'extraire, à chaque étape du cycle de vie, une ou plusieurs facettes. Chaque facette est présentée et éditée avec l'outil le plus adéquat en créant des passerelles vers des outils tiers.

Afin de mettre en œuvre la méthodologie proposée, un Framework est développé pour offrir un environnement de description multi-facettes : architecture réseau, architecture matérielle, architecture logicielle, flux de données et environnement physique. Un exemple de passerelle entre le Framework et un simulateur existant (Omnet++) a été développé pour estimer les performances des réseaux de capteurs. Le Framework permet la génération automatique des scripts de simulation ainsi que des codes sources pour les noeuds du réseau.

Abstract : *Model-driven engineering for the design and implementation of sensor networks*

The life cycle of a sensor network involves several steps, including network sizing, embedded software development, hardware design, analysis and simulation, physical deployment, data exploitation, and network maintenance. Throughout this cycle, several specific tools are used. It is therefore necessary to manually describe the characteristics of the sensor network in each of these tools using their own languages. This results in significant development costs and potential inconsistencies between the different descriptions.

To address this problem, we propose a methodology based on model driven engineering approach. It aims to cover all life cycle steps of a sensor network. All concepts and characteristics of a sensor network are described using a metamodel. From this referential, it is possible to extract, at each step of the life cycle, one or more facets. Each facet is presented and edited with the most appropriate tool by creating gateways to external tools.

In order to implement the proposed methodology, a Framework is developed to provide a multi-facets environment : network architecture, hardware architecture, software architecture, data flow and physical environment. An example of a gateway between the Framework and an existing simulator (Omnet++) has been developed to estimate sensor network performances. The Framework allows automatic generation of simulation scripts as well as source codes for network nodes.