Année : 2020 Thèse N° 185 /ST2I

École Nationale Supérieure d'Informatique et d'Analyse des systèmes
Centre d'Études Doctorales en Sciences des Technologies de l'Information et de l'Ingénieur

THÈSE DE DOCTORAT

Proposition et validation formelle d'une architecture ReDy fiable et dynamique destinée aux systèmes IoT
- Application aux Smart Grids -

Présentée par
Kaoutar Hafdi

Le 27 Novembre 2020

Formation doctorale : Informatique

Structure de recherche : Laboratoire Advanced Digital enterprise Modelling and Information Retrival (ADMIR), Equipe IT Architecture and Model Driven Systems Development (IMS)

Jury

Pr. Bouchra EL ASRI PES, Ensias, Université Mohamed V de Rabat	**Président**
Pr. Mahmoud NASSAR PES, Ensias, Université Mohamed V de Rabat	**Directeur de thèse**
Pr. Ahmed EL KHADIMI PES, INPT, Rabat	**Rapporteur**
Pr. Abderrahim HASBI PES, EMI, Université Mohamed V de Rabat	**Rapporteur**
Pr. Driss BOUZIDI PES, Ensias, Université Mohamed V de Rabat	**Rapporteur**
Pr. Rahal ROMADI PES, Ensias, Université Mohamed V de Rabat	**Examinateur**

Remerciements

Je tiens tout d'abord à demander à Dieu d'accepter et d'englober dans sa miséricorde mon cher Directeur de thèse **Mr Abdelaziz KRIOUILE** qui a continué à m'encadrer dans ma thèse jusqu'aux derniers jours de sa vie malgré son départ en retraite. Que Dieu accepte tout son travail avec des centaines voir des milliers d'ingénieurs et des dizaines de docteurs qui ont bénéficié de son enseignement et de son encadrement.

Je tiens à remercier **Mr Mahmoud NASSAR**, Professeur de l'Enseignement Supérieur à l'ENSIAS, Université Mohammed V de Rabat d'avoir accepté de diriger et de clôturer cette thèse malgré les conditions très particulières par lesquelles nous passons en ces moments ainsi que notre pays et le monde entier.

J'adresse ma gratitude également pour les honorables professeurs, **Mr Ahmed ELKHADIMI**, PES à l'Institut National des Postes et Télécommunications de Rabat, **Mr Abderrahim HASBI**, PES à l'Ecole Mohammadia des Ingénieurs de Rabat, **Mr Driss BOUZIDI**, PES à l'ENSIAS, Université Mohammed V de Rabat, qui ont accepté d'évaluer ce travail et d'en être les rapporteurs.

J'exprime mes vifs remerciements à **Mme Bouchra EL ASRI**, PES à l'ENSIAS, Université Mohammed V de Rabat, pour avoir accepté d'être la présidente du jury.

Je tiens à remercier l'examinateur et membre du jury, **Mr Rahal ROMADI**, PES à l'ENSIAS, Université Mohammed V de Rabat.

Je tiens également à remercier **Mr Abderahman KRIOUILE,** PA à l'ENSIAS, pour son rôle dans le co-encadrement et la collaboration dans cette thèse qui a été concrétisé par plusieurs publications communes.

Mes profonds remerciements trouveront destination chez mes chers parents, ma sœur et mes frères, ainsi que toute ma belle-famille, pour leur encouragement et leur patience. C'est grâce à votre amour et à votre présence que je suis arrivée là aujourd'hui.

A la fin, je ne peux trouver les mots pour remercier mes deux petites perles Aya et Nouha, ainsi que mon mari Abderahman, pour être à mes cotés aux moments les plus difficiles. Je ne peux sans doute oublier mon petit bébé qui m'a pleinement accompagné durant les derniers mois de cette thèse en priant Dieu qu'il arrive en bonne santé dans ce monde.

Résumé

L'internet des objets (Internet of Things - IoT) est la rencontre de plusieurs domaines technologiques ayant pour objectif de rendre la technologie de plus en plus omniprésente. L'IoT a pu envahir les applications domestiques et celles déstinées aux entreprises et a commencé à accéder aux applications à grande échelle comme le transport et les services publics. Ceci dit, l'IoT doit répondre à de nouveaux défis concernant la fiabilité, le dynamisme et l'extensibilité pour pouvoir réussir ce passage technologique de l'internet des personnes à l'internet des objets, où des objets communiquent avec des objets ou des personnes pour répondre au mieux aux différents besoins. Le système IoT doit être *Fiable* pour tolérer des erreurs si certains composants ne sont plus fonctionnels. Il doit être *Dynamique* dans le sens ajout et suppression de composants au cours du fonctionnement du système. Le système IoT doit être *Extensible* pour pouvoir être déployé dans des applications à grande échelle. Les réseaux électriques intelligents (Smart Grids) présentent un exemple de ces nouvelles applications IoT où les défis de fiabilité, de dynamisme et d'extensibilité sont essentiels à la bonne intégration de l'IoT.

Dans notre travail, nous commençons par analyser l'architecture type de bout en bout des systèmes IoT et nous l'appliquons en particulier dans notre cas d'étude des Smart Grids. Ensuite, nous proposons une architecture dite ReDy (Reliable and Dynamic) destinée aux systèmes IoT distribués, fiables, dynamiques et extensibles. Notre architecture ReDy est hybride vue qu'elle combine des solutions centralisées et décentralisées. Afin de construire ce type d'architecture, nous utilisons un algorithme de brassage permettant la gestion de l'adhésion des noeuds au système. Nous proposons une formalisation de cet algorithme.

Afin de valider formellement les critères de fiabilité, de dynamisme et d'extensibilité de l'architecture ReDy, nous modélisons, en langage formel LNT, l'architecture ReDy dans différentes configurations. Nous proposons aussi un modèle formel pour l'analyse de l'algorithme de brassage. Nous avons pu simuler et générer les systèmes à transitions étiquetées (LTS) correspondant aux différentes configurations. Ensuite, nous exprimons des propriétés de logique temporelle en utilisant le langage MCL. La validation formelle

est menée en utilisant la boîte à outil CADP.
Notre approche est appliquée dans le cas d'étude d'un micro Smart Grid. En effet, nous avons proposé une architecture du micro Smart Grid étudié selon l'architecture ReDy et nous l'avons modélisée et validée formellement en suivant notre processus de validation formelle. Comme résultat, notre proposition permet de mieux gérer dynamiquement l'optimisation de la production et de la consommation de l'énergie électrique.

Mots clés :
Systèmes Distribués, Algorithmes Distribués, Internet des Objets (IoT), Architecture IoT, Architecture ReDy, Fiabilité, Dynamisme, Modélisation formelle, Vérification Formelle, Model-Checking, CADP, Systèmes Asynchrones, Algorithme de brassage, Processus incrémental et itératif, Smart Grids.

Abstract

The Internet of Things (IoT) is the meeting of several technological fields with the aim of making technology more and more omnipresent. IoT was able to invade home and business applications and began to access large-scale applications such as transportation and utilities. Therefore, the IoT must respond to new challenges related to reliability, dynamism and scalability to be able to succeed in this technological transition from the internet of people to the internet of things, where things communicate with things or people to best meet different needs. The IoT system must be *reliable* to tolerate errors if certain components are no longer functional. It must be *dynamic* in the sense of adding and removing components during the operation of the system. The IoT system must be *scalable* in order to be deployed in large-scale applications. Smart Grids present an example of these new IoT applications where the challenges of reliability, dynamism and scalability are essential to the proper integration of IoT.

In this work, we start by analyzing a typical end-to-end IoT architecture and we apply it in particular in our case study of Smart Grids. Then, we present the proposed ReDy (Reliable and Dynamic) architecture intended for distributed, reliable, dynamic and scalable IoT systems. Our ReDy architecture is hybrid since it combines centralized and decentralized solutions. In order to build this type of architecture, we use the shuffling algorithm responsible of the membership management of the system nodes. We propose a formalization of this algorithm.

In order to formally validate the criteria of reliability, dynamism and scalability of the ReDy architecture, we modelize, in formal language LNT, the ReDy architecture in different configurations. We also propose a formal model for the analysis of the shuffling algorithm. We were able to simulate and generate the labeled transition systems (LTS) corresponding to the different configurations. Next, we express temporal logic properties using the MCL language. Formal validation is carried out using the CADP toolbox.

Our approach is applied in the case of a micro Smart Grid. Indeed, we propose an architecture of the micro Smart Grid studied according to the ReDy architecture and we

formally modelize and validate it following our formal validation process. As a result, our proposal makes it possible to better manage dynamically the optimization of the production and consumption of energy.

Keywords : Distributed Systems, Distributed Algorithms, Internet of Things (IoT), IoT architectures, Redy Architecture, Reliability, Dynamism, formal Modeling, Formal Verification, Model Checking, CADP, Asynchronous Systems, Shuffling algorithm, Incremental and iterative process, Smart Grids.

Table des matières

Résumé (Français/English) **i**

Table des matières **v**

Liste des figures **ix**

Liste des tableaux **xi**

Liste des abréviations **xiii**

1 Introduction **1**

1.1 Contexte général 1

1.2 Motivation et problématique 2

1.3 Contribution 3

1.4 Organisation du document 5

2 Etat de l'art **7**

2.1 Introduction 7

2.2 Internet des objets (IoT) : généralités, architectures et technologies de déploiement 7

2.2.1 Définitions de l'internet des objets (IoT) 10

2.2.2 Tendances 12

2.2.3 Domaines d'application de l'internet des objets 13

2.2.4 Caractéristiques de l'internet des objets 18

2.2.5 Architectures de l'internet des objets (IoT) 21

2.2.6 Technologies utilisées dans les systèmes IoT 31

2.2.7 Modèles de communication dans une architecture IoT 32

2.3 Défis des systèmes IoT 34

2.4 Problématique de la thèse 38

2.5 Validation formelle des architectures IoT 38
2.5.1 Analyse formelle des architectures IoT dans la littérature 39
2.5.2 Approche adoptée de validation formelle 40
2.5.3 Boite à outils CADP 45
2.6 Cas d'illustration : Les Smart Grids 47
2.6.1 Présentation générale des Smart Grids 47
2.6.2 Exigences générales des Smart Grids 49
2.6.3 Présentation des systèmes AMI (Advanced Metering Infrastructure) 51
2.6.4 Application de notre approche au Smart Grids 53
2.7 Conclusion 54

3 Architecture de type ReDy déstinée aux systèmes IoT 55
3.1 Introduction 55
3.2 Architecture générale de bout en bout (E2E) de l'IoT 55
3.2.1 Présentation de l'architecture générale de bout en bout 55
3.2.2 Défis des systèmes IoT selon l'architecture IoT E2E 57
3.3 Modélisation des Smart Grids selon l'architecture IoT de bout en bout . . 57
3.4 Spécification d'un système IoT de type ReDy 60
3.4.1 Exigences générales des systèmes ReDy 61
3.4.2 Architecture de type ReDy 62
3.4.3 Etude fonctionnelle 64
3.4.4 Représentation simplifiée de l'architecture ReDy 67
3.5 Conception des systèmes ReDy 68
3.5.1 Algorithme de brassage pour la gestion de l'adhésion des composants au système 68
3.5.2 Modèle de communication 69
3.6 Conception d'un composant du système : architecture logicielle 74
3.7 Processus incrémental et itératif de la conception de l'architecture ReDy basé sur la validation formelle 76
3.8 Conclusion 78

4 Modélisation et validation formelle de l'architecture ReDy proposée 79
4.1 Introduction 79
4.2 Présentation du processus de validation formelle 79
4.3 Présentation du modèle de l'architecture ReDy 80
4.4 Modélisation des types de données 82

4.5 Modélisation formelle d'un sous-système de l'architecture ReDy (mode local) 83
4.5.1 Modélisation d'une unité de détection DU (Detection Unit) 83
4.5.2 Modélisation d'une unité d'action AU (Action Unit) 84
4.5.3 Modélisation d'une unité de gouvernance GU (Governance Unit) . 85
4.5.4 Modélisation d'un sous-système unitaire 91
4.5.5 Génération d'un LTS d'un sous-système 95
4.5.6 Exemples de génération du modèle du sous-système et leurs caractéristiques 95
4.6 Modélisation formelle de l'architecture ReDy (mode distant) 98
4.6.1 Présentation du modèle formel de l'architecture ReDy 98
4.6.2 Exemples de génération du modèle de l'architecture ReDy et leurs caractéristiques 100
4.7 Validation formelle du modèle de l'architecture ReDy 100
4.7.1 Propriété ordre Detection Action 101
4.7.2 Absence d'inter-blocages (deadlocks) 102
4.7.3 Absence de blocage operationnel (livelocks) 103
4.8 Conclusion 103

5 Algorithme de brassage proposé dans l'architecture et sa formalisation 105
5.1 Introduction 105
5.2 Présentation de l'Algorithme de Brassage 106
5.3 Formalisation de l'algorithme de brassage 108
5.3.1 Définition des variables et des types de données 108
5.3.2 Formalisation d'une communication de brassage 109
5.4 Conclusion 113

6 Modélisation et analyse formelle de l'algorithme de brassage proposé 115
6.1 Introduction 115
6.2 Modélisation formelle de l'agorithme de brassage 116
6.2.1 Modélisation des types de données relatifs à l'algorithme de brassage 116
6.2.2 Modélisation de l'architecture globale du système utilisé pour la validation de l'algorithme de brassage 118
6.2.3 Modélisation d'une unité de gouvernance opérant dans l'algorithme de brassage 120
6.3 Modélisation du scénario d'ajout d'un noeud au système 129
6.3.1 Modèle formel du scénario d'ajout 130

6.3.2 Génération de scénarios d'exécution 132
6.4 Modélisation du scénario de suppression d'un noeud du système 132
6.5 Conclusion 135

7 Application de notre architecture ReDy au Smart Grids 137
7.1 Introduction 137
7.2 Modélisation d'un Smart Grid par l'architecture ReDy 138
7.3 Etude fonctionnelle des Smart Grids 139
7.3.1 Mode local 140
7.3.2 Mode distant 140
7.4 Présentation du micro Smart Grid à modéliser formellement 141
7.5 Modélisation formelle du Micro Smart Grid 143
7.5.1 Modélisation formelle d'un noeud du micro Smart grid 144
7.5.2 Modélisation formelle du micro Smart Grid 153
7.6 Validation formelle 154
7.7 Conclusion 156

8 Conclusion et perspectives 159
8.1 Résumé des contributions 159
8.2 Perspectives 162
8.3 Publications scientifiques 163

Bibliographie 175

Table des figures

2.1 Domaines d'application de l'internet des objets (IOT) [GBMP13] 9
2.2 Visions de l'IoT [ČH18] 11
2.3 Courbe des technologies émergentes en 2018 (Hype Cycle) 13
2.4 Tendances de recherche depuis 2004 des termes Wireless Sensor Networks, Ubiquitous Computing, Internet of Things 14
2.5 Architecture IoT basée sur le style SOA [GTK+10] 24
2.6 Architecture IoT basée sur le cloud [GBMP13] 26
2.7 Architecture IoT basée sur le fog computing [AFGM+15] 27
2.8 Architecture IoT générale multi-couches [TM17] 28
2.9 Architecture IoT multi-couches [CXL+14] 29
2.10 Technologies de communication sans fil pour l'internet des objets [ČH18] . 33
2.11 (a) Modèle de communication Device-to-Device (b) Modèle de communication Device-to-Cloud [TATM15] 33
2.12 (a) Modèle de communication Device-to-Gateway (b) Modèle de communication Back-End Data Sharing [TATM15] 34
2.13 Défis des systèmes IoT 35
2.14 Etude des domaines d'application des systèmes IoT [GBMP13] 39
2.15 Deux LTS représentant le fonctionnement des machines (M1) et (M2) . . 45
2.16 Exemple de composition d'un Smart Grid 50
2.17 Description des systèmes AMI (Advanced Metering Infrastructure) 52

3.1 Architecture IoT multi-couches générale de bout en bout 56
3.2 Défis des systèmes IoT répartis par couches de l'architecture IoT E2E . . 58
3.3 Smart Grid selon l'architecture IoT de bout en bout 59
3.4 Composants du Smart Grid organisés par niveau 60
3.5 Architecture ReDy en couches 65
3.6 Modes de communication dans l'architecture ReDy 66

3.7 Exemple d'une représentation simplifiée de l'architecture ReDy 67
3.8 `Événement d'initialisation` 72
3.9 `Événement de récupération (Recovery)` 72
3.10 `Événement de broadcast` 72
3.11 `Événement de livraison (delivery)` 73
3.12 Architecture logicielle d'une unité de gouvernance 74
3.13 Processus incrémental et itératif basé sur la validation formelle 77

4.1 Processus de la validation formelle 81
4.2 Modules LNT de l'architecture ReDy 82
4.3 Portes de communication dans un sous-système 83
4.4 Processus du module Governance_unit 86
4.5 Composition parallèle d'un sous-système unitaire 92
4.6 LTS d'un sous-système 96
4.7 LTS minimisé d'un sous-système 97
4.8 Composition parallèle d'une architecture ReDy 98
4.9 Portes de communication dans un système ReDy 99
4.10 Propriété MCL ordre Detection Action 102
4.11 Détection d'un inter-blocage-Problème de famine 104

5.1 Exemple d'une itération de l'algorithme de brassage 107

6.1 Initialisation des noeuds du système 120
6.2 Initialisation des noeuds du système lors du scénario d'ajout 131
6.3 LTS de quelques scénarios d'exécution du modèle d'ajout d'un noeud au système 133
6.4 Scénario de suppression d'un noeud du système 134

7.1 Modélisation d'un Smart Grid par l'architecture ReDy 140
7.2 Modélisation du micro Smart Grid étudié selon l'architecture ReDy 142
7.3 Portes de communication en mode local 151
7.4 Portes de communication dans le mode distant 153
7.5 Propriété MCL ordre ConsS ProdA 156
7.6 Propriété MCL ordre ProdS ConsA 156

Liste des tableaux

2.1 Résumé des architectures IoT 31
2.2 Technologies utilisées dans les systèmes IoT 32

4.1 Exemples de génération du modèle du sous-système 95
4.2 Exemples de génération du modèle de l'architecture ReDy 100

7.1 Eléments de modélisation des Smart Grids par l'architecture ReDy 138

Liste des abréviations

AMI	Advanced Metering Infrastructure
API	Application Programming Interface
AUPS	AUthenticated Publish subscribe
AU	Action Unit
BCG	Binary Coded Graph
BLE	Bluetooth Low Energy
CADP	Construction and Analysis of Distributed Processes
CIU	Customer Interface Unit
CTL	Computation Tree Logic
CoAP	Constrained Application Protocol
CONVECS	Construction of Verified Concurrent Systems
CORBA	Common Object Request Broker Architecture
CPS	Cyber Physical System
DCP	Data Collection Platform
DCU	Data Concentrator Unit
DHT	Distributed Hash Table
DMS	Data Management System
DTLS	Datagram Transport Layer Security
DU	Detection Unit
EMS	Energy Management System
GU	Gouvernance Unit
HAN	Home Area Network
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
IoT	Internet of Things
IoV	Internet of Vehicles
ISoc	Internet Society
LAN	Local Area Network

LTL	Linear Temporal Logic
LTS	Labeled Transition System
LNT	LOTOS New Technology
LOTOS	Language of Temporal Ordering Specification
MAS	Multi Agent Systems
MBT	Model Based Testing
MCL	Model Checking Language
MDCS	Meter Data Collection System
MDMS	Meter Data Management System
MEMS	MicroElectroMechanical Systems
MGU	Management Governance Unit
MQTT	Message Queue Telemetry Transport
M2M	Machine To Machine
NAN	Neighborhood Area Network
NFC	Near Field Communication
OS	Operating System
OCIS	Open/Caesar Interactive Simulator
OTA	Online Trust Alliance
P2P	Peer To Peer
ReDy	Reliable and Dynamic
REST	Representational State Transfer
RFID	Radio Frequency Identification
SG	Smart Grid
SOA	Service Oriented Architecture
SVL	Script Verification Language
TCP	Transmission Control Protocol
TIC	Technologies de l'Information et de Communication
UbiComp	Ubiquitous Computing
UDP	User Datagram Protocol
UGU	Unit Gouvernance Unit
USS	Unit Sub-System
URIs	Universal Resource Indicators
VANETs	Vehicular Adhoc Networks
WSNs	Wireless Sensor Networks
XML	eXtensible Markup Language

Chapitre 1

Introduction

1.1 Contexte général

L'internet des objets (IoT) [Ash09] est en pleine voie de recherche et d'investigation afin d'acquérir de plus en plus de maturité pour être intégré dans plusieurs domaines d'applications. L'IoT présente un axe de développement de l'humanité qui va changer plus ou moins notre façon de vivre en permettant une intégration plus forte des technologies de l'information et de communication (TIC) dans notre quotidien.

Technologiquement, l'IoT est la rencontre de plusieurs domaines technologiques. Chaque communauté scientifique l'aborde de son point de vue et essaie de donner des orientations et des standards. Le standard IoT, la carte embarquée IoT standard, la solution IoT standard, l'architecture d'un système IoT standard, le protocole de communication IoT standard : ce sont tous des objectifs que plusieurs communautés scientifiques, mais surtout industrielles, ont essayé d'atteindre. Mais au final, l'ensemble des efforts a créé tant de divergences que de convergences.

En effet, au cours des dernières années, les systèmes IoT sont devenus de plus en plus complexes et omniprésents dans notre vie. Leurs applications sont devenues intégrées dans notre vie quotidienne, créant un nouveau style de vie amélioré. Les applications récentes des systèmes IoT requièrent une intégration étroite des technologies de calcul, de communication et de contrôle pour assurer l'évolutivité, les performances, la fiabilité, la robustesse et l'efficacité de ces systèmes [RLSS10]. Ces systèmes sont en contact avec leur environnement au moyen de capteurs et d'actionneurs permettant une grande interactivité

entre le système et son environnement.

L'IoT est la prochaine phase de transformation numérique : de nombreux objets qui nous entourent seront connectés. Des capteurs, des actionneurs et des unités de calcul vont former des réseaux pour différentes applications. Ces applications peuvent être déstinées à des utilisations domestiques et personnelles, en entreprise, en services publics, en transports ou autres. L'IoT s'est déja intégré dans quelques domaines spécifiques comme les applications déstinées à l'utilisation personnelle ainsi que certaines utilisations en entreprise.

1.2 Motivation et problématique

L'IoT est en train de relever plusieurs défis pour répondre aux besoins continus des applications personnelles et domestiques, ainsi que des applications déstinées aux entreprises. Aujourd'hui l'IoT est entrain de relever les défis relatifs aux applications au niveau des villes et services publics et du transport. L'objectif serait d'assurer le passage à des objets intelligents "plug and play" [GBMP13], c'est à dire à des systèmes IoT extensibles, dynamiques et fiables. Le système IoT doit être *Extensible* pour pouvoir être déployé dans de grandes installations à la taille d'une ville ou d'un réseau électrique intelligent au niveau d'un pays. Le système IoT doit être *Dynamique* dans le sens ajout et suppression de composants au cours du fonctionnement du système. Le système IoT doit être *Fiable* pour tolérer des erreurs et doit continuer à fonctionner avec les ressources disponibles même si certains composants ne sont plus fonctionnels.

Les réseaux électriques intelligents (Smart Grids) présentent un exemple de ces nouvelles applications IoT où les défis de fiabilité, de dynamisme et d'extensibilité sont essentiels à la bonne intégration de l'IoT. En effet, un réseau électrique intelligent nécessite d'être étendu de façon récurrente. Il ne doit pas être limité par une certaine taille, d'où le besoin en extensibilité. L'ajout d'une nouvelle entité ou la déconnexion d'une entité ne doit pas impacter le fonctionnement du réseau d'où le besoin en fiabilité et en dynamisme.

De nombreux systèmes IoT ont plusieurs exigences communes, indépendamment des caractéristiques relatives au champ d'application du système en question. Les exigences communes peuvent être rassemblées et analysées afin de proposer des solutions réutilisables pour chaque famille de systèmes IoT similaires. Comment peut on alors proposer une solution réutilisable pour une famille de systèmes IoT ? et comment peut on répondre

aux défis de fiabilité, de dynamisme et d'extensibilité de ce type de système permettant des applications IoT à grande échelle ? Ce sont ce genre de questions qui représentent l'essentiel de la problématique que nous avons développé dans ce travail de thèse.

1.3 Contribution

Notre travail a commencé par une étude de l'état de l'art des architectures IoT proposées dans la littérature. Ceci nous a permis de dresser une architecture type de bout en bout en cinq couches d'un système IoT : dispositifs, réseaux, middleware, application et business. Pour chacune de ces couches, nous décrivons les principales technologies utilisées, ainsi que les principaux défis rencontrés. Cette proposition peut être utilisée dans de nombreux domaines d'application des systèmes IoT tels que les maisons intelligentes, la surveillance de la santé, le transport intelligent, l'agriculture intelligente, les Smart Grids, etc. Parmi les défis des systèmes IoT, nous nous sommes penchés sur les défis de dynamisme, de fiabilité et d'extensibilité vu que ce sont des défis essentiels pour permettre les applications à grande échelle des systèmes IoT.

Nous proposons ensuite une architecture que nous avons appelée ReDy (Reliable and Dynamic), fiable et dynamique qui est une solution réutilisable pour un large spectre de systèmes IoT [HK15]. L'architecture ReDy est une architecture hybride vu qu'elle propose une composition en un ensemble de sous-systèmes. Ces derniers communiquent dans une logique décentralisée peer-to-peer, alors que chaque sous-système est régi par une logique centralisée client/serveur entre son unité de gouvernance et l'ensemble de ses autres éléments. Notre solution intègre principalement trois exigences importantes communes aux systèmes concernés. (i) La première consiste à concevoir le système dans un environnement très dynamique, c'est-à-dire que les composants peuvent en permanence rejoindre et quitter le réseau du système. (ii) La deuxième exigence concerne la tolérance aux pannes afin d'assurer la fiabilité du système. Le système conçu doit présenter une résistance élevée aux pannes, ce qui permet de préserver le comportement général du système, même en présence de composants défectueux. (iii) La troisième exigence (qui peut être déduite de l'aspect dynamique) consiste à assurer l'extensibilité de notre solution, c'est à dire qu'il faut être capable d'ajouter de nouveaux composants au système sans altérer son fonctionnement global. Ces trois exigences sont communes à une grande famille d'applications IoT. Ainsi, en plus de proposer une architecture commune pour ces systèmes, notre solution fournit des règles générales qui doivent être respectées et mises en œuvre pendant la phase de conception afin de construire des systèmes IoT

fiables, dynamiques et extensibles. Nous devons également être en mesure d'exprimer le comportement du système de manière non ambigue afin de valider son fonctionnement global.

Dans notre travail, nous utilisons les méthodes formelles pour modéliser l'architecture ReDy et pour valider des comportements compliqués des systèmes correspondants. Les méthodes formelles constituent un type particulier de techniques basées sur les mathématiques pour la spécification et la vérification de ces systèmes. L'utilisation des méthodes formelles nous permet d'assurer un bon niveau de fiabilité et de robustesse de la conception que nous proposons. La modélisation formelle de notre système nous permet d'exprimer le comportement du système de manière non ambiguë : la spécification formelle exprime une sémantique unique. De plus, notre modélisation formelle élaborée peut être validée à l'aide de méthodes formelles automatiques et exhaustives. Ce qui va nous permettre la modélisation et la validation de notre architecture proposée. Dans notre travail, notre modèle formel est exprimé en utilisant le langage LNT (LOTOS New Technology) [CCG+14] et les propriétés de logique temporelle sont exprimées en utilisant le langage MCL (Model Checking Language) [MT08a]. Pour exploiter le modèle formel pour des fins de spécification et de validation formelle, nous utilisons la boite à outils CADP (Construction and Analysis of Distributed Processes) [GLMS13].

Notre contribution dans ce travail de thèse peut se résumer par les points suivants :

- En se basant sur l'étude de l'état de l'art, nous avons pu établir une architecture IoT de bout en bout [HKK19] suite à l'étude des différentes architectures IoT proposées dans la littérature et nous avons pu réaliser une application de cette architecture au cas d'étude des Smart Grids.

- Proposer une architecture ReDy pour la conception des systèmes IoT fiables, dynamiques et extensibles [HK15]. Cette architecture est composée de plusieurs sous-systèmes. Chaque sous-système est composé d'une unité de gouvernance et d'un ensemble d'unités de détection et d'action.

- Proposer un algorithme de brassage pour assurer l'adhésion des composants au système IoT conçu selon l'architecture ReDy proposée. En partant des éléments décrivant l'algorithme de brassage dans la littérature en langue naturelle [VGVS05], nous avons pu le formaliser afin de permettre son implémentation et sa vérification.

- Modélisation formelle de l'architecture ReDy en utilisant le langage LNT [HKK17]. Le

modèle formel obtenu est un modèle valide ce qui permet de valider la structure modulaire et extensible de notre architecture ReDy. Nous utilisons également les techniques de model checking afin de vérifier des propriétés assurant le bon fonctionnement du système. Cette validation formelle nous permet d'assurer la fiabilité et l'extensibilité de la structure de notre architecture ReDy.

- Modélisation formelle de l'algorithme de brassage [HKK17]. Le but de cette modélisation est de valider le bon fonctionnement de notre algorithme ainsi que les comportements d'ajout et de suppression de composants au cours du fonctionnement du système. Ce travail de validation formelle nous permet de garantir l'aspect dynamique de notre architecture mais aussi de renforcer sa fiabilité.

- Application de notre proposition au cas pratique d'un micro Smart Grid. Ce cas d'étude nous permet de mettre en illustration notre proposition en mettant en pratique toutes les propositions théoriques précédemment citées [HKK18].

1.4 Organisation du document

Le reste de ce rapport est organisé comme suit.

Dans le chapitre II, nous présentons la synthèse de l'état de l'art de l'IoT en examinant les domaines d'application des systèmes IoT et les défis rencontrés lors de la mise en place de ces systèmes. Nous étalons une étude des architectures proposées dans la littérature pour répondre à ces défis et nous relevons notre problématique et positionnons notre travail par rapport au travaux existants. Nous présentons également la boite à outils CADP utilisée pour valider formellement notre travail ainsi que tous les concepts et méthodes relatifs à la validation formelle de notre approche.

Dans le chapitre III, nous présentons notre architecture ReDy proposée en étalant les différentes phases suivies pour aboutir à ce résultat. Nous présentons principalement le processus de notre approche incrémentale et itérative basée sur la validation formelle.

Dans le chapitre IV, nous présentons le processus de notre approche de validation formelle. Ensuite nous étalons le modèle formel de l'architecture ReDy proposée et nous procédons à la phase de validation formelle de ce modèle en suivant le processus proposé et en appliquant principalement les outils de model checking adéquats.

Dans le chapitre V, nous élaborons l'algorithme de brassage qui intervient dans la phase

de construction d'un réseau de composants dans l'architecture ReDy. Nous proposons, ensuite, sa formalisation en algorithmique afin de préparer sa modélisation formelle.

Dans le chapitre VI, nous proposons la modélisation formelle de l'algorithme de brassage proposé afin de valider les scénarios d'ajout et de suppression d'un composant du système ReDy pour renforcer les exigences de notre architecture.

Dans le chapitre VII, nous présentons un cas d'étude pratique de notre architecture ReDy. Il s'agit d'un micro Smart Grid auquel nous appliquons également notre processus de validation formelle.

Dans le chapitre VIII, nous présentons une conclusion générale de notre travail ainsi que les principales perspectives et directives de recherches futures.

Chapitre 2

Etat de l'art

2.1 Introduction

Dans ce chapitre, nous commençons par étudier le domaine de l'IoT en analysant plusieurs aspects relatifs à cette thématique. Nous nous focalisons sur les architectures IoT existantes ainsi que les principaux défis relevés par celles-ci. Nous allons ensuite présenter notre problématique pour enfin proposer la méthodologie de travail choisie qui repose sur l'utilisation des méthodes formelles comme outil principal. Enfin nous présentons les principaux concepts et outils de méthodes formelles utilisés dans notre travail.

2.2 Internet des objets (IoT) : généralités, architectures et technologies de déploiement

La détection ubiquitaire (Ubiquitous sensing) [GBMP13] réalisée par les technologies de réseau de capteurs sans fil (WSN) est utilisée dans plusieurs domaines de notre vie quotidienne moderne. Ceci permet de mesurer et de comprendre des indicateurs environementaux, allant de ressources naturelles aux environnements urbains. La prolifération des capteurs et des actionneurs dans un réseau communiquant et interagissant crée la notion de *l'Internet des objets (IoT)* [GBMP13, AIM10]. Dans cette optique, les capteurs et les actionneurs sont parfaitement intégrés dans l'environnement qui nous entoure.

Motivé par l'adaptation récente d'un ensemble de technologies sans fil adaptées telles que

les tags RFID et les capteurs et les actionneurs embarqués, l'internet des objets n'est plus dans sa phase initiale et représente actuellement la prochaine technologie revolutionnaire permettant de passer à l'internet futur complétement intégré dans l'environnement.

Dans le paradigme de l'internet des objets, plusieurs objets qui nous entourent feront partie du réseau d'une manière ou d'une autre. La technologie de l'identification par radio fréquence (RFID) et la technologie des réseaux de capteurs ainsi que d'autres technologies seront developpées afin de répondre à ce nouveau défi, dans lequel les systèmes d'information et de communication sont invisiblement embarqués au sein de l'environnement qui nous entoure. Ainsi, il en résulte la génération d'énormes quantités de données qui doivent être stockées, traitées et présentées sous une forme efficace et facilement interprétable. Ceci dit, il faut prévoir des dispositifs de monitoring et de stockage, des outils d'analyse et de traitement et enfin des platformes de visualisation.

Une partie indispensable de l'internet des objets réside dans la connectivité inteligente avec les réseaux existants et le calcul sensible au contexte en utilisant les resources du réseau. Avec la présence croissante du Wifi et du 4G-LTE accès internet sans fil, voire même la 5G [CAMD16], l'évolution se fait vers des réseaux d'informations et de communication ubiquitaires. Cependant, afin d'assurer l'emergence de l'internet des objets, il faut être capable de connecter, non seulement les smartphones et les portables, mais aussi les objets existants qui nous entourent, afin d'intégrer l'inteligence au sein de notre environnement. Afin que la technologie soit complètement intégrée dans la vie de l'utilisateur, l'internet des objets demande : (1) une compréhension partagée de la situation de ses utilisateurs et de leurs appareils, (2) des architectures logiciel et des réseaux de communication omniprésents afin de calculer et de transmettre l'information au bon endroit, (3) des outils d'analyse visant à réaliser un comportement autonome et intelligent. Avec ces trois fondements en place, la connectivité intelligente et le calcul sensible au contexte peuvent être accomplis.

Le terme internet des objets (Internet of Things) est inventé en premier par Kevin Ashton en 1999 dans le contexte de la gestion de la chaine d'approvisionnement [Ash09]. Cependant, la définition est devenue plus inclusive couvrant un large champs d'applications comme les soins de la santé, les services publics, le transport, etc [SGFW10, MSDPC12, WADX15]. Malgré que la définition des "objets" a changé avec l'évolution de la technologie, le but principal de permettre à l'ordinateur de détecter l'information sans aucune intervention humaine reste le même. Actuellement, on commence à parler d'un réseau d'objets interconnectés qui est responsable non seulement de collecter l'information

de l'environnement (détection) et d'intéragir avec le monde physique (actions/ commandes/ contrôle), mais aussi il utilise les standards d'internet existants afin de fournir des services pour le transfert de l'information, l'analyse, les applications et la communication [WADX15, ČH18]. Motivé par la dominance de dispositifs faisant usage de technologies sans fil ouvertes telles que le Bluetooth, la RFID, le Wi-Fi, les services téléphoniques, ainsi que les capteurs et actionneurs embarqués [WADX15, ČH18, GBMP13, CAMD16], l'internet des objets n'est plus dans ses tous débuts, et il est sur le point de transformer l'internet statique actuelle à l'Internet Futur complétement intégré. La révolution de l'internet a donné lieu à un niveau d'interconnexion entre les gens sans précédent. La révolution à venir sera l'interconnexion entre les objets afin de créer un environnement intelligent.

La figure 2.1 illustre les domaines d'application de l'internet des objets ainsi que ses utilisateurs finaux.

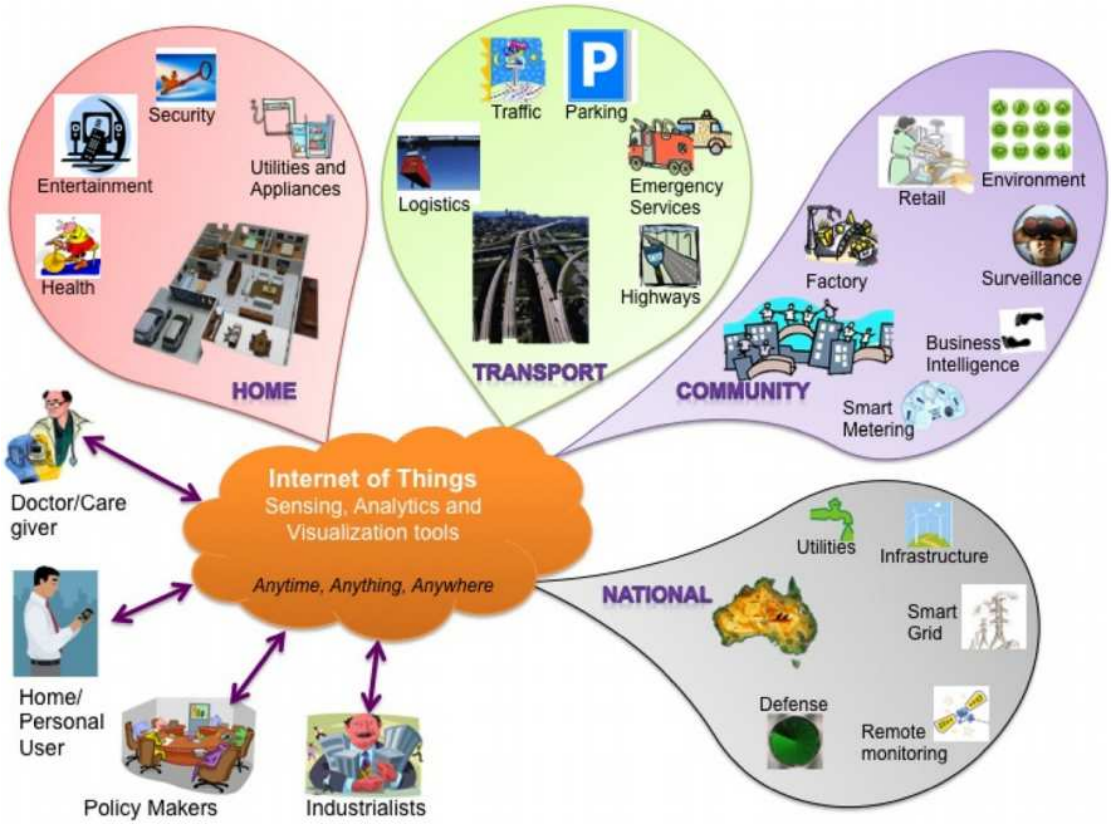


FIGURE 2.1 – Domaines d'application de l'internet des objets (IOT) [GBMP13]

L'omniprésence croissante des technologies de communication et de l'intelligence dans les objets de tous les jours a donné naissance à l'informatique ubiquitaire (Ubiquitous

computing) qui a pour but d'intégrer la technologie dans tous les détails de notre vie quotidienne. Actuellement, nous sommes dans l'ère de l'informatique ubiquitaire qui est la troisième ère de l'histoire de l'informatique, succèdant ainsi à l'ère des ordinateurs personnels et celle des ordinateurs centraux [GBMP13]. Les smartphones, les appareils portables ainsi que les objets intelligents de façon générale changent notre environnement qui devient plus interactif et plus informatif. Mark Weiser définit un environnement intéligent comme étant "le monde physique qui est entrelacé de manière riche et invisible par les capteurs, les actionneurs, les afficheurs, et les éléments de calcul, embarqués en toute transparence dans les objets utilisés quotidiennement dans notre vie" [WGB99].

La création de l'internet représente une étape très importante pour réaliser la vision de l'informatique ubiquitaire (ubicomp) qui permet aux appareils individuels de communiquer avec tout autre appareil dans le monde. L'interconnexion entre les appareils démontre l'existence d'une grande quantité de ressources de calcul distribué et de stockage appartenant à de différents propriétaires.

Les avancées de la technologie des systèmes micro-electro-mechaniques (MEMS), des communications sans fil, et de l'électronique digitale a donné lieu au développement d'appareils miniaturisés disposant de fonctionnalités de détection, de calcul et de communication sans fil sur de courtes distances. Ces appareils miniaturisés, appelés noeuds, se connectent entre eux afin de construire des réseaux de capteurs sans fil qui sont utilisés dans beaucoup de domaines d'application [ASSC02]. Les réseaux de capteurs sans fil (WSN) assurent la capacité de détection ubiquitaire qui est une fonctionnalité principale pour la réalisation de la vision de l'informatique ubiquitaire et par conséquent l'implémentation des systèmes IoT. Pour la réalisation d'une vision complète de l'IoT, il est indispensable d'assurer des ressources de stockage et de calcul efficaces, fiables et évolutives.

2.2.1 Définitions de l'internet des objets (IoT)

Dans la littérature, l'IoT a été traité par des écoles différentes, résultant ainsi à différentes visions et points de vues. Ces visions ont influé les applications. Les auteurs définnissent l'IoT selon des aspects particuliers et des intérêts spécifiques. Ainsi, l'IoT peut être défini selon plusieurs visions : IoT orienté internet, IoT orienté objets, IoT orienté sémantique, et IoT orienté service (voir figure 2.2) [AIM10, ČH18]. Les approches existantes abordent l'IoT selon une ou plusieurs visions parmi les quatre citées en haut.

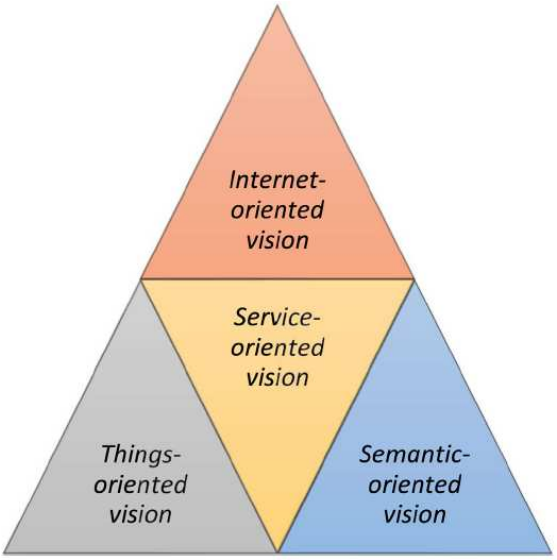


FIGURE 2.2 – Visions de l'IoT [ČH18]

- Selon la vision *orientée internet*, l'IoT est considérée comme une infrastructure globale qui assure la connectivité entre des objets virtuels et physiques. La communauté ISOC (Internet SOCiety) définit ainsi l'IoT comme étant *"des scénarios ou la connectivité du réseau et le calcul s'étend aux objets, aux capteurs, et aux dispositifs de tous les jours qui ne sont pas normalement considérés comme ordinateurs. Ces dispositifs peuvent ainsi générer, échanger et consommer les données avec le minimum d'intervention humaine"* [REC15] :

- Il existe des définitions qui se focalisent sur l'aspect physique (vision orientée objets). Les objets peuvent être des entités du monde réel ou bien des entités virtuelles. Dans cette vision, on retrouve Al-Fuqaha et al. [AFGM+15] qui considèrent l'IoT comme étant *"une technologie qui permet aux objets physiques de voir, d'entendre, de partager l'information, de coordiner les décisions et de réaliser des tâches"*. Dans la même vision, on retrouve la communauté européenne de projets de recherche sur l'internet des objets [SGFW10] qui considère que *"les objets sont des participants actifs dans les processus d'affaires, d'informations et de société, ayant la possibilité d'intéragir et de communiquer entre eux et avec l'environnement en échangeant les données et les informations détéctées dans l'environnement. En même temps, ces objets réagissent de façon autonome aux événements du monde réel/physique et l'influencent en exécutant des processus qui déclenchent des actions et créent des services avec ou sans intervention humaine directe"*.

- Gubbi et al [GBMP13] proposent une définition centrée utilisateur (orientée service), appliquée dans le cloud computing, n'utilisant aucun protocole de communication spécifique.

Leur définition de l'internet des objets pour les environnements inteligents est comme suit : *"Interconnexion d'appareils de détection et d'actionnement offrant la possibilité de partager l'information à travers des platformes en utilisant un framework unifié. Ceci est réalisé par une détection ubiquitaire, une analyse de données et une représentation de l'information avec le cloud computing comme framwork unifié"*.

-Il existe également des définitions incluant plusieurs visions à la fois. On retrouve la définition de Atzori et al. [AIM10] qui stipule que l'internet des objets peut être réalisé en trois paradigmes : IoT orienté internet (middlware), IoT orienté objets (capteurs), et Iot orienté sémantique (connaissance). L'utilité de l'internet des objets est prouvée lorsqu'on l'utilise dans un domaine d'application où les trois paradigmes se retrouvent.

Colakovic et al. [ČH18] proposent une approche qui inclut quatre visions : internet, objets, sémantique et services. Ils définissent ainsi l'IoT comme *"un paradigme d'interconnexion assuré par un ensemble de technologies qui fournissent une connectivité transparente entre les objets physiques et virtuels afin de faciliter le développement de services et d'applications intelligentes avec des capacités d'auto-configuration. L'ensemble de technologies est une combinaison de divers technologies qui assurent une connectivité transparente à tout moment et n'importe où par n'importe qui et n'importe quoi"*.

2.2.2 Tendances

L'internet des objets fait partie des technologies IT émergentes identifiées dans la courbe de Gartner (voir figure 2.3). Cette courbe[Ref URL] est un moyen de représenter l'émergence, l'adoption, la maturité et l'impact sur les applications de technologies spécifiques. Il a été prévu que l'internet des objets prendra 5 à 10 ans pour être adopté par le marché.

La popularité de différents paradigmes varie avec le temps. La popularité de la recherche sur le Web, telle que mesurée par les tendances de la recherche Google au cours des dix dernières années pour les termes Internet des objets, réseaux de capteurs sans fil et informatique ubiquitaire est présentée dans la figure 2.4. Comme on peut le constater, depuis la création de l'internet des objets, le volume de recherche augmente constamment avec la tendance à la baisse des réseaux de capteurs sans fil.

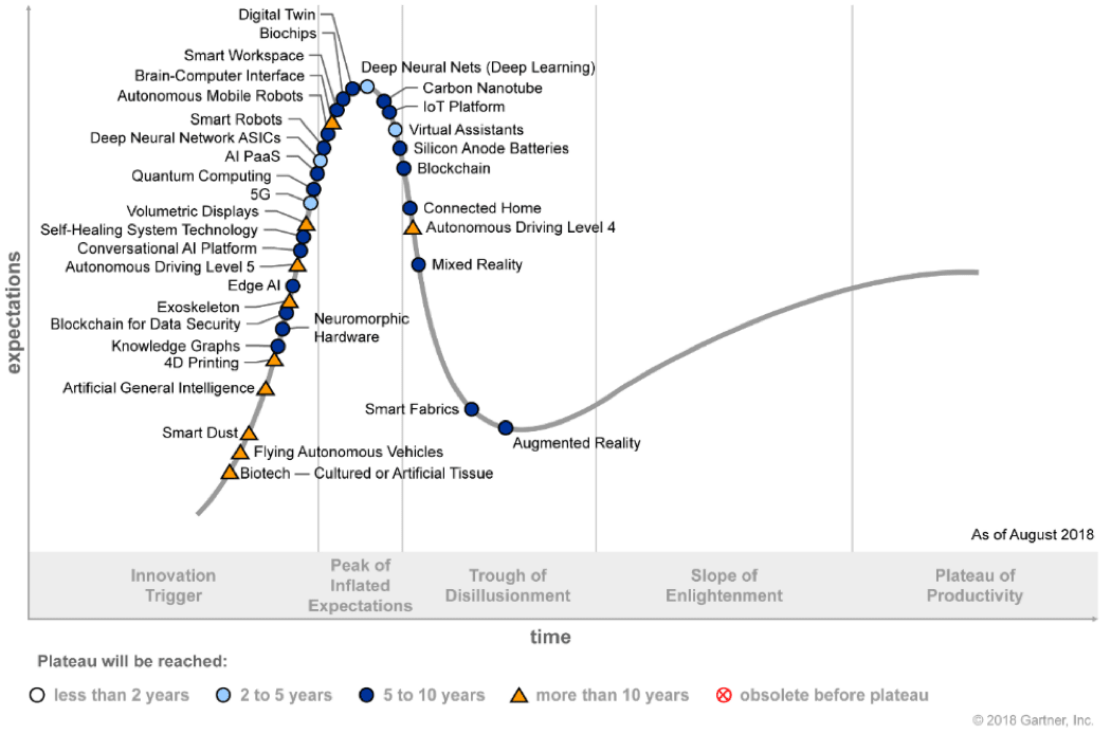


FIGURE 2.3 – Courbe des technologies émergentes en 2018 (Hype Cycle)

2.2.3 Domaines d'application de l'internet des objets

Il existe plusieurs classifications des domaines d'application de l'internet des objets [GBMP13, AIM10, WADX15, MSDPC12, SGFW10].

Dans la suite, nous proposons une classification des applications de l'internet des objets selon sept domaines : (1) Personnel et domestique ; (2) Applications sociales ; (3) Soins de la santé ; (4) L'industrie et l'entreprise ; (5) Transport et logistique ; (6) Services publics et (7) Monitoring de l'environnement.

Personnel et domestique

L'information collectée par les capteurs est utilisée uniquement par des individus qui sont des propriétaires directs du réseau. En général, le WiFi est la technologie utilisée permettant ainsi une large bande passante lors du transfert de données.

Le contrôle des équipements de la maison comme les climatiseurs, les réfrigérateurs, les machines à laver etc., va permettre une meilleure gestion de la maison et de l'énergie. Comme résultat, les consomateurs vont devenir impliqués dans la révolution de l'internet

FIGURE 2.4 – Tendances de recherche depuis 2004 des termes Wireless Sensor Networks, Ubiquitous Computing, Internet of Things

des objets [AB05, DM08].

Les capteurs et les actionneurs distribués dans les maisons et les entreprises peuvent rendre notre vie plus confortable : le chauffage des pièces de la maison peut être adapté selon nos préférences et selon le temps qu'il fait, l'éclairage des pièces peut changer au cours de la journée en fonction des horaires, les incidents domestiques peuvent être évités en utilisant un contrôle adapté et des systèmes d'alarme, et pour finir l'énergie peut être économisée en éteignant les équipements électriques lorsqu'on n'en a pas besoin[ST17].

- Perte ou vol d'objets : ces applications consistent à utiliser un outil permettant de trouver les objets dont on ne se rappelle pas l'endroit. De la même manière, une telle application permet à l'utilisateur d'être averti si jamais des objets sont déplacés hors d'une zone restreinte (la maison ou le bureau), ce qui indique qu'il y a une tentative de vol de l'objet. Dans ce cas, l'événement déclenché doit être directement notifié au propriétaire ou bien aux agents de sécurité. Par exemple, l'application peut envoyer un SMS aux utilisateurs lorsque l'objet volé quitte le bâtiment sans autorisation. L'objet concerné peut être un ordinateur portable, un portefeuille ou autre.

Applications sociales

Les applications dans ce domaine permettent à l'utilisateur d'interagir avec d'autres personnes afin de maintenir et construire des relations sociales. En effet, les objets peuvent déclencher automatiquement la transmission de messages aux amis afin de leur permettre de connaitre ce qu'on est en train de faire ou bien ce qu'on avait fait. On pourrait ainsi informer nos amis de nos déplacements, de nos voyages ainsi que de nos activités [WBC+09]. Les principales applications dans ce domaine sont les suivantes :

- Les réseaux sociaux : ces applications relatives à la mise à jour automatique concernant nos activités sociales sur les réseaux sociaux (Twitter à titre d'exemple). On peut penser à utiliser des RFID qui génèrent des événements à propos des personnes et des endroits afin de donner aux utilisateurs la possibilité d'effectuer des mises à jour en temps réel sur leurs réseaux sociaux. Les interfaces utilisateur de l'application affichent un ensemble d'événements que les amis ont préalablement définis, et les utilisateurs peuvent contrôler leurs listes d'amis ainsi que les événements à afficher pour chaque ami[WADX15, AUSA19].

Soins de la santé

Les technologies de l'IoT ont plusieurs applications dans le domaine médical. Elles peuvent être utilisées afin de renforcer les solutions de vie assistée actuelles. Les patients vont porter sur eux des capteurs médicaux afin de contrôler des paramètres de santé comme la température, la tension artérielle et l'activité respiratoire. D'autres capteurs, portables (comme les accéléromètres et les gyroscopes) ou fixes, sont utilisés afin de rassembler les données servant à contrôler les activités du patient dans le milieu où il vit. Les informations sont rassemblées et transmises à des centres médicaux distants qui vont effectuer du contrôle distant avancé et seront capables de donner des réponses rapides quand il y a besoin. L'interconnexion entre ces capteurs hétérogènes peut fournir une image complète des paramètres de la santé, permettant ainsi de faire appel à une intervention médicale faisant objet d'un soin préventif [DMOD$^+$10, CDDM$^+$15].

Une autre application relative à ce domaine concerne les solutions de soins de la santé et de bien-être personalisées. L'utilisation de capteurs portables combinés avec des applications convenables installées sur des dispositifs personnels permet au gens de contrôler leurs activitées quotidiennes (nombre de pas, calories brûlées, exercices effectués, etc.) tout en donnant des suggestions afin d'améliorer leur style de vie et de prévenir des problèmes de santé[SRS$^+$16].

Industrie et entreprise

Dans un environnement professionel, le réseau des objets est assimilé à une application destinée à être utilisée au sein de l'entreprise. Les informations collectées par ce type de réseau sont utilisées uniquement par les propriétaires et les données sont affichées de façon selective. L'application commune à ce type de réseau est le contrôle de l'environnement de travail. Cette application est implémentée afin de garder la traçabilité des fonctionnaires

ainsi que la gestion de différents services au sein du bâtiment (Chauffage, ventilation, climatisation, éclairage)[LL15].

Au sein d'une usine, les capteurs ont toujours été une partie intégrante de sa configuration afin d'assurer la sécurité, l'automatisation, le contrôle du climat, etc. Ces capteurs sont remplacés par un système sans fil qui va permettre d'effectuer des modifications dans la configuration quand il y a besoin. Il s'agit dans ce cas d'un sous réseau d'IoT dédié à la maintenance de l'usine. L'intégration des nouvelles technologies de façon générale et de l'IoT en particulier dans l'industrie a donné lieu à une nouvelle génération nommée Industrie 4.0[XXL18].

Transport et logistique

Le transport et la logistique sont placés dans un domaine séparé vue la nature de partage de données et d'implémentation requise. Le trafic urbain est le contributeur principal dans la pollution sonore ainsi que dans la dégradation de la qualité de l'air de la ville à cause de l'émission de gaz à effet de serre. L'embouteillage induit directement une augmentation dans le coût des activités économiques et sociales dans plusieurs villes. L'efficacité et la productivité des chaines d'approvisionnement sont sévèrement influencées par cette congestion provoquant ainsi du retard dans le chargement de la marchandise et par la suite des problèmes dans le temps de livraison. Le transport intelligent doté d'application IoT va permettre l'utilisation de réseaux de capteurs sans fil à grande échelle afin de pouvoir contrôler la durée des trajets, le choix de la route à prendre, le temps d'attente estimé, ainsi que les emissions polluant l'air. L'internet des objets est susceptible de remplacer les informations sur le trafic fournies par les réseaux de capteurs existants qui implémentent des détecteurs de véhicules à boucle inductive employés dans les intersections dans les systèmes de contrôle de trafic [WBC+09, HL84]. Plusieurs travaux de recherches s'interèssent au contrôle du trafic routier afin de répondre au problème de congestion des voitures[CN16]. Nous trouvons aussi des applications particulières comme celles proposées par Vardhana et al.[VAAV18] qui proposent une solution pour faciliter le trajet des ambulances pendant les embouteillages.

Services publics

Dans ce domaine d'application, l'information collectée du réseau est généralement destinée à l'optimisation des services plutôt qu'à la consommation du client. Ce type d'information est utilisé par les entreprises exerçant dans le domaine des services publics pour gérer

les ressources afin de réduire le coût et d'optimiser le profit. Ce domaine est caractérisé par des réseaux très étendus, à niveau régional ou national, ayant pour objectif la surveillance de services publics critiques et la gestion efficace des ressources. Dans ce cas, la communication se fait par WiFi, satellite ou cellulaire [GBMP13].

Réseau électrique intelligent : La réduction de la consommation de l'éléctricité, et par la suite du coût qui en résulte, peut être réalisée par la surveillance des points d'éléctricité. cette information peut être utilisée pour modifier la manière de consommation de l'éléctricité. Ceci est utilisé au niveau de la ville afin d'assurer une haute qualité de service [LLC+11].

L'internet des objets basé sur les vidéos : ce type d'application intègre le traitement d'image, la vision par ordinateur anisi que les frameworks de réseau. La surveillance par caméra connectée à un réseau permet de suivre une cible donnée, d'identifier des activités suspectes, de détecter un bagage abondonné et de contrôler l'accès non autorisé. L'analyse automatique de comportement et la détection d'événement sont des aspects avancés dans l'analyse et le traitement de la vidéo [AMC07, PYC+18].

Le contrôle du réseau de distribution d'eau et la qualité de l'eau potable est une application critique dans ce domaine. Des capteurs mesurant des paramètres critiques sont installés à des endroits spécifiques afin d'assurer une bonne qualité d'approvisionnement. Ceci empêche une contamination accidentelle de l'eau potable. Le même réseau peut être étendu afin de pouvoir contrôler l'irrigation de terrains agricoles, ainsi que de contrôler des paramètres de sol afin de pouvoir prendre des décisions concernant l'agriculture [LSL+11, GBMP13].

Le monitoring de l'environnement

L'internet des objets peut être utilisé également dans des applications de contrôle et de surveillance des paramètres de l'environnement. La fonctionnalité de détection assurée par les capteurs faisant partie d'un système distribué assure le suivi et la surveillance de phénomènes naturels (mesure de la température, des précipitations, de la hauteur de l'eau dans une rivière, de la vitesse du vent). L'ensemble de ces données hétérogènes sont intégrées dans des applications globales afin de déduire des résultats pouvant sauver des vies humaines et minimiser les dégâts. L'utilisation de dispositifs miniaturisés permet l'accès à des zones critiques dont les conditions ne permettent pas à l'être humain de s'y rendre (zones volcaniques, profondeurs de l'océan). L'information collectée est

communiquée à des centres de décision afin de détecter des anomalies. Une application possible de l'internet des objets assurant la sûreté de l'environnement réside dans la détection d'incendie, principalement dans les forêts, par moyen de capteurs mesurant la température. La détection précoce d'incendie permet de prendre les mesures adéquates afin de sauver les vies humaines et de limiter les dégats et les pertes permettant ainsi de réduire l'impact du désastre. Il existe plusieurs autres scénarios relatifs à la protection civile pouvant tirer profit des technologies de l'internet des objet, à savoir les tremblements de terre, les tsunamis, etc. Dans ces cas, l'avertissement des équipes de sauvetage et des différentes équipes concernées permet de prendre les mesures nécessaires et d'établir des stratégies de coordinations entre les différentes entités concernées [MSDPC12, SKS17].

Il existe également des applications plus simples relatives au domaine de l'environnement comme l'évaluation de la qualité de l'air ou bien la programmation de notification lorsque les poubelles sont pleines [SKP+11, VSB12].

2.2.4 Caractéristiques de l'internet des objets

Dans la suite, nous identifions des caractéristiques que l'internet des objets doit pouvoir supporter [MSDPC12].

Hétérogénéité

L'internet des objets est caractérisé par une grande hétérogénéité au niveau des appareils faisant partie du système. Ces appareils doivent remplir des fonctionnalités très différentes du point de vue calcul et communication. La gestion d'un tel haut niveau d'hétérogénéité doit être supporté au niveau de l'architecture et du protocole utilisés.

Passage à l'échelle (scalability)

Vu que les objets que l'on utilise quotidiennement deviennent connectés à une infrastructure d'information globale, le passage à l'échelle s'impose sous différents niveaux y compris : (i) Le nommage et l'adressage (naming and addressing), vue la taille du système résultant dans l'internet des objets, (ii) la communication des données et le réseautage, en raison du haut niveau d'interconnexion entre un grand nombre d'entités, (iii) la gestion de l'information et de la connaissance, en raison de la possibilité de construire une représentation digitale de toute entité et/ou phénomène dans le monde physique, (iv) l'approvisionnement du service et la gestion, en raison du nombre massif des options

d'exécution des services qui doivent être disponibles ainsi que le besoin de manipuler des ressources hétérogènes.

Échange de données à travers les technologies sans fil

Dans le domaine de l'IoT, les technologies de communication sans fil vont jouer un rôle très important, car elles permettent aux objets intelligents d'être connectés. L'adoption ubiquitaire de moyens de communication sans fil pour l'échange de données peut relever des problèmes au niveau du spectre de disponibilité poussant ainsi vers l'adoption de systèmes radio cognitifs et dynamiques [Hay05].

Optimisation de l'énergie

Pour un bon nombre d'entités de l'IoT, la minimisation de l'énergie dépensée dans la communication et le calcul représente une contrainte primordiale. Alors que les techniques en relation avec la récupération de l'énergie (par exemple les micro panneaux solaires, le matériel piézoélectrique) vont libérer les appareils des contraintes imposées par l'utilisation des batteries, l'énergie sera toujours une ressource rare qu'il faut manipuler avec précaution. Ainsi, le besoin de concevoir des solutions qui ont tendance à optimiser l'utilisation de l'énergie (même au dépend de la performance) sera de plus en plus présent [MSDPC12].

Localisation et suivi

Etant donné que les entités dans l'IoT peuvent être identifiées et sont munies de moyens de communication sans fil de courte portée, il devient possible de suivre la position et le mouvement de ces objets intelligents dans le monde physique. Ceci est particulièrement important pour les applications de la logistique et les applications concernant la gestion du cycle de vie des produits. Dans ces domaines, les technologies RFID sont déjà largement utilisées [MSDPC12].

Auto-organisation

La complexité et le dynamisme des systèmes IoT poussent à distribuer de l'intelligence sur les entités du système, afin que des objets intelligents soient capables de réagir de façon autonome dans différentes situations dans le but de minimiser l'intervention humaine. Selon les requêtes des utilisateurs, les noeuds dans l'IoT vont s'organiser de façon autonome dans des réseaux ad hoc transitoires, assurant ainsi les moyens basics pour le partage des données et l'exécution de tâches de coordination. Ceci inclut la

possibilité d'effectuer la découverte d'appareils et de services sans avoir besoin d'un déclenchement externe, afin de construire un réseau et de réaliser le comportement des protocoles pour l'adaptation au contexte actuel [MSDPC12].

Gestion des données

L'internet des objets va mettre le point sur l'échange et l'analyse de quantités massives de données. Afin de transformer les données en informations utiles, et afin d'assurer l'interopérabilité entre différentes applications, il est nécessaire d'organiser les données dans des formats adéquats standardisés, et dans des modèles et des sémantiques de description de leur contenu (meta-data), en utilisant des langages et des formats bien définis. Ceci va permettre aux applications IoT de supporter le raisonnement automatisé qui est une caractéristique très importante pour assurer une adoption réussie d'une telle technologie sur une grande échelle [MSDPC12].

Sécurité et protection de la vie privée

En raison de l'implication très importante dans le monde physique, la technologie de l'IoT doit être sécurisée et doit préserver la vie privée des individus. Ceci signifie que la sécurité représente une propriété très importante au niveau système, et doit être prise en considération lors de la conception d'architectures et de méthodes pour les solutions IoT. Cet aspect représente une exigence très importante qui va assurer l'acceptation des utilisateurs et par conséquent une large adoption de la technologie.

D'après l'alliance OTA (Online Trust Alliance) de ISoc (Internet Society), les défis relatifs à la sécurité de l'internet des objets peuvent se résumer dans les points suivants [REC15] : la sécurité faible est favorisée par l'économie, difficultés d'instaurer la sécurité surtout pour les nouvelles entreprises, la complexité des systèmes IoT qui impose de sécuriser chaque partie à part, le support de la sécurité n'est pas toujours maintenu, la connaissance en sécurité est faible et limitée, les incidents de sécurité sont difficilement détectables, les mécanismes de responsabilité légale ne sont pas clairs [REC15].

Exemples de risques de la sécurité pour la personne : manipulation de caméras de surveillance, lectures malveillantes du compteur d'électricité intelligent, manipulation d'un aspirateur robot intelligent.

2.2.5 Architectures de l'internet des objets (IoT)

Nous avons besoin de traiter les architectures afin de représenter, organiser et structurer l'internet de objets de manière lui permettant un bon fonctionnement [WADX15]. L'artchitecture IoT peut être considérée comme un système composé d'objets physiques (e.g : détecteurs, actionneurs), d'objets virtuels (e.g : services du cloud, les couches et les protocoles de communication), ou bien une combinaison de ces deux perspectives [Ray18]. Ces architectures doivent être capables de supporter les dispositifs de l'IoT et leurs services, ainsi que le workflow que ces dispositifs vont affecter. Dans la suite, on classifie les architectures comme suit : architecture physique (hardware/network), architecture logicielle, architecture du processus, et architecture générale [WADX15, ČH18].

Architecture physique de l'IoT (Hardware/network)

Une multitude d'architectures physiques ont été proposées afin de supporter l'environnement de calcul distribué requis par l'internet des objets. Parmi ces architectures on retrouve principalement l'architecture point-à-point (peer-to-peer), l'architecture à base du code EPC et l'architecture des réseaux de capteurs sans fil(WSNs).

- *L'architecture peer-to-peer (P2P)* : dans ce modèle, l'interaction entre les entités est symétrique, c'est à dire que toute entité se comporte comme client et serveur à la fois. L'architecture peer-to-peer peut être implémentée en utilisant un protocole de distribution comme le protocole de routage appelé table de hashage distribué (Distributed Hash Table-DHT). Il est possible de faire la conception d'une architecture point-à-point déstinée à être utilisée dans les applications Web des objets (Web of Things-WoT). Ces applications utilisent une communication M2M impliquant ainsi les dispositifs embarqués [ACCM10].

- *Architecture à base du code EPC* : EPC (Electronic Product Code) est un identifiant universel qui donne une identité unique à un object auquel nous avons affecté une étiquette RFID. L'identité de chaque objet doit être unique afin que chaque objet soit identifiable au sein d'un certain champs d'objets [SDS19]. Le nombre EPC permet l'échange des informations des données entre les sociétés et leurs partenaires. Pour des objectifs de standardisation, une architecture nommée EPCglobal network a été proposée [GC13]. Dans cette architecture, des rôles, des interfaces et un vocabulaire commun sont spécifiés. Les détails d'implémentation restent à la charge des utilisateurs finaux en fonction du domaine d'application. Une telle architecture assure la traçabilité du mouvement des objets dans la chaîne d'approvisionnement ainsi que la collecte d'informations relatives à

chaque objet [GC13].

Plusieurs architectures IoT sont basées sur l'approche EPC. Une telle architecture peut être implémentée pour supporter un réseau d'accès hétérogène, en particulier en utilisant un réseau ZigBee puisqu'il est capable de collecter les informations les plus récentes des objets [HM11]. L'architecture proposée assure deux fonctions. La première fonction consiste à montrer comment procéder pour enregistrer de nouveaux objets ou dispositifs dans le HAN (Home Area Network). La deuxième fonction est comment permettre aux différents objects de communiquer par internet en utilisant les protocoles génériques. L'architecture EPC proposée utilise une combinaison des réseaux de capteurs et des réseaux EPC. Ces derniers sont responsables de fournir les informations du produit à travers les services web des fabricants [HM11].

Une autre proposition d'architecture IoT basée sur l'approche EPC est le projet Bridge [A+10]. BRIDGE (Building Radio Frequency Identification Solutions for the Global Environment) est un projet européen qui a pour objectifs la recherche, le développement et l'implémentation d'outils permettant le déploiement d'applications à base de la technologie RFID et du réseau EPC. Ce projet est implémenté selon une architecture complétement décentralisée basée sur l'architecture EPCglobal. Cette architecture est déstinée à la chaine d'approvisionnement des produits et peut être appliquée dans sept domaines d'application. Le problème principal rencontré dans une telle architecture concerne la sécurité et la vie privée. Une utilisation illicite du code EPC doit être empêchée et une transmission sécurisée des données entre les lecteurs et les étiquettes doit être assurée. Il n'existe pas d'extension du réseau standard EPC pouvant inclure les données des capteurs.

- *Architecture à base de capteurs et de réseaux de capteurs sans fil (Wireless sensor networks-WSN)* : les réseaux de capteurs sans fil (WSNs) permettent aux dispositifs embarqués d'être utilisés en toute transparence. L'utilisation des WSNs lors de la conception des architectures IoT est très prometteuse car elle assure la distributivité et la sensibilité au contexte qui sont deux caractéristiques principales des architectures IoT. Il est possible d'implémenter un framework intégré pour l'interconnexion des réseaux de capteurs sans fil et des actionneurs avec les réseaux standards en utilisant par exemple les services web [CBC+10]. Lors de la conception d'une architecture à base de WSNs, il est possible d'utiliser une méthode d'adaptation complète de l'IP, comme est le cas dans l'architecture SNAIL (Sensor Networks for an All-IP World) [HKP+10]. Cette architecture inclut quatre protocoles réseau : mobilité, web enablement, synchronisation du temps et sécurité. Une

autre approche consiste à utiliser des points d'accès M2M dans les architectures IoT basées sur les WSNs. L'idée principale consiste à connecter les différents capteurs au point d'accès M2M pour assurer la communication avec les utilisateurs finaux ou bien avec les différent services fournis. Cette solution peut être appliquée dans les applications des bâtiments intelligents utilisant les WSNs. L'hétérogénéité et la sécurité sont assurées en utilisant cette approche [FPVC14]. Il existe une autre approche permettant la conception d'architectures IoT à base de WSNs. Il s'agit de l'architecture autonomique qui consiste à implémenter un protocole de communication autonomic qui prend en consideration les contraintes du terminal sans fil disponible dans l'internet des objets, à savoir les capteurs et les RFID [Puj06].

Architecture logicielle de l'IoT

Les architectures logicielles sont nécessaires pour assurer l'accès et le partage des services offerts par les dispositifs de l'IoT. Il existe plusieurs approches qui offrent un framework pour l'IoT comme l'architecture orientée service (SOA), l'architecture RESTful et les architectures basées sur le fog et le cloud computing. Ces architectures se focalisent sur les services et la flexibilité et couvrent les systèmes d'exploitation, le middleware de l'IoT, les API, la gestion des données, le big data, etc.

- *Les architectures IoT basées sur l'architecture orientée service (Service Oriented Architecture-SOA)* : l'architecture orienté service (SOA) est un style d'architecture logicielle qui repose sur l'utilisation des standards des services web. Il est possible également d'implémenter le style SOA en utilisant toute autre technologie basée sur les services, comme Jini [ASW+99], CORBA [SF00] ou REST [FT00]. Dans une architecture IoT basée sur le style SOA, tout dispositif est un consommateur et/ou fournisseur de services, ou bien partageant les ressources et interagissant avec les consommateurs des services via des APIs (Application Programming Interfaces) compatibles. Les technologies SOA permettent la publication, la découverte, la sélection, et la composition des services offerts par les dispositifs IoT [GTK+10]. Contrairement aux services et aux applications entreprise traditionnels, qui sont principalement des entités virtuelles, les services du monde réel sont fournis par des systèmes embarqués qui sont directement reliés au monde physique [GTK+10]. Dans les architecture IoT, on peut trouver les deux types de services selon l'application en question (Figure 2.5).

Plusieurs travaux de recherche ont été menés pour assurer une intégration effective de

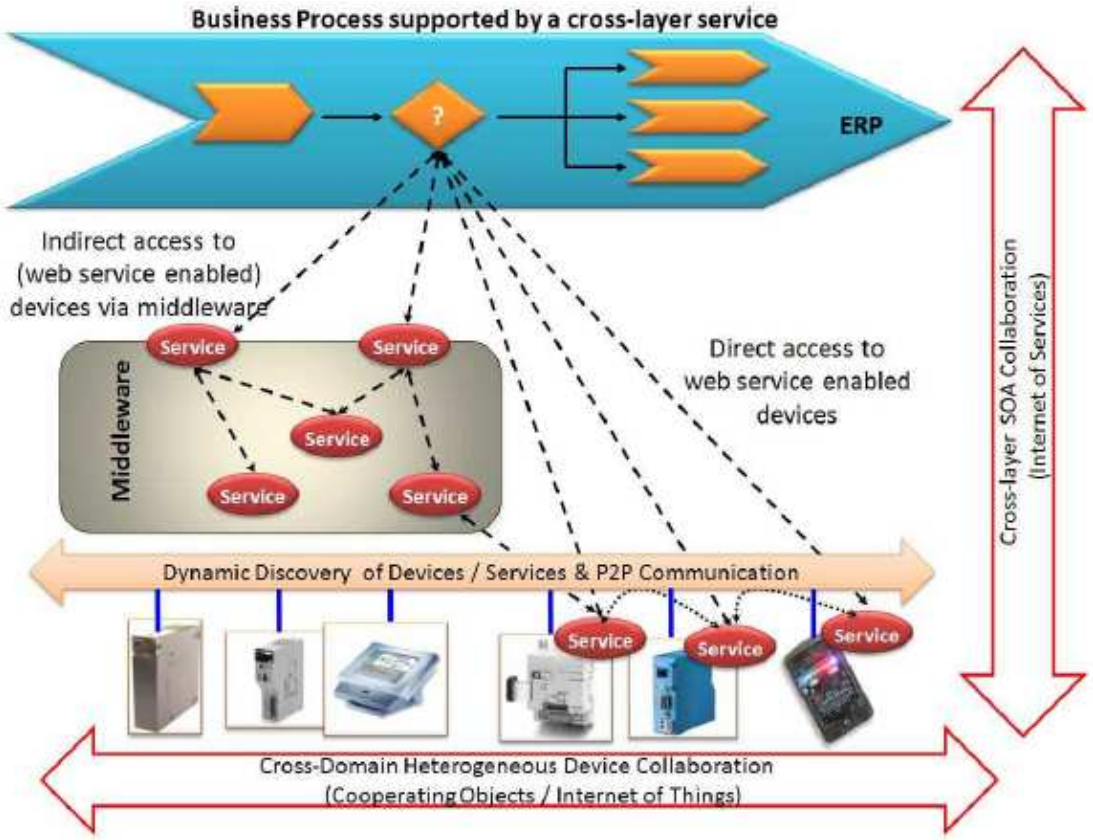


FIGURE 2.5 – Architecture IoT basée sur le style SOA [GTK+10]

l'internet des objets dans les services entreprise en utilisant le paradigme SOA [Grø08, SKG+09, CGB16]. Comme architecture IoT basée sur le style SOA, il existe l'architecture d'intégration SOCRADES (SOCRADES Integration Architecture-SIA) [SKG+09]. Cette architecture est composée de six couches : Interface d'application, gestion de service, gestion de dispositif, sécurité, platforme d'abstraction, et dispositifs. Une communication standardisée au niveau de toutes les couches est assurée en utilisant les services web. En plus, une architecture système est proposée afin d'assurer un usage dynamique des services implémentés au niveau des dispositifs physiques. Des travaux de recherche ont également proposé un ensemble d'exigences qui facilitent la recherche et la découverte des services du monde réel [GTK+10].

Les systèmes IoT sont hétérogènes et très dynamiques. Ceci encourage la communauté scientifique à proposer des protocoles de gestion de la confiance (trust management protocol) pour les architectures IoT basées sur le style SOA [CGB16].

- *Les architectures IoT basées sur le transfert d'état représentationnel (The Represen-*

tational State Transfer-REST) : REST est un style d'architecture logicielle qui définit un ensemble de contraintes déstinées à être utilisées pour la création de services web. L'architecture REST se base principalement sur une communication client-serveur avec des contraintes définies. Le style REST est implémenté en utilisant les URIs (Universal Resource Identifier) pour l'identification des services dans le web, HTTP (Hypertext Transfer Protocol), ainsi que des types média standardisés (comme HTML et Extensible Markup Language-XML) pour l'échange des données [FT00]. L'architecture RESTful est une architecture faiblement couplée, simple et modulaire, ce qui motive de l'utiliser avec les standards du web afin d'intéragir avec les objets intelligents. Comme résultat, on a introduit le concept WoT (Web of Things) plutôt que le concept IoT (Internet of Things) [GTMW11]. Dans le concept WoT, les objets intelligents et leurs services sont complétement integrés dans le web en réutilisant et en adaptant les technologies et les patterns utilisés dans le web traditionnel. Il est à noter que l'utilisation du protocole HTTP fait augmenter la moyenne de la latence de communication. Ce protocole doit être utilisé dans des scénarios où les délais relativement long n'affectent pas les exigences du système.

Il est possible de construire des services web pour les applications IoT en utilisant le protocole CoAP (Constrained Application Protocol) [SHB14] qui est défini dans la charte de l'environnement RESTful (Constrained RESTful Environment-CoRE) [She12]. Le protocole CoAP permet des communications à base du style REST entre les applications figurant dans les systèmes distribués embarqués [CBC$^+$10, CGB$^+$11]. Le but du protocole CoAP est de pouvoir supporter les paquets de taille limitée, les dispositifs à faible énergie, ainsi que les canaux de communication non fiables. Ces aspects sont principalement présents dans les architectures IoT. Par conséquent, l'utilisation du protocole CoAP améliore la moyenne de latence de communication dans les systèmes IoT.

Il existe dans la littérature un autre protocole utilisé pour l'implémentation de l'architecture REST. Il s'agit dy protocole MQTT (Message Queue Telemetry Transport MQTT) [RSMCP16]. MQTT est un protocole léger orienté événement et message, ce qui permet aux dispositifs de communiquer de façon asynchrone à travers des réseaux afin d'atteindre des systèmes éloignés. Le protocole MQTT est basé sur le modèle d'interaction publish/subscribe. En particulier, MQTT a été implémenté afin de faciliter la connexion des objets avec le web, et afin de supporter les réseaux non fiables caractérisés par une petite bande passante et une grande latence. Comme résultat, le protocole MQTT est adéquat pour la conception d'architecture IoT basée sur le style REST. Le problème prin-

cipal du protocole MQTT réside dans le faible niveau de sécurité qu'il fournit. Il existe des travaux de recherche qui visent à renforcer la sécurité du protocole MQTT en proposant, par exemple, le système open source appelé AUPS (AUthenticated Publish/Subscribe) qui est déstiné à être utilisé dans l'IoT [RSMCP16].

- *Architectures IoT basées sur le cloud* : les systèmes IoT génèrent une très grande quantité de données qui doit être stockée, analysée et présentée de manière transparente, efficace et facilement interprétable. Le cloud computing assure une grande fiabilité, modularité et autonomie pour les systèmes IoT. En effet, une plateforme basée sur le cloud reçoit les données envoyées par les capteurs ubiquitaires, analyse et interprète les données, et assure la visualisation des résultats en utilisant des outils du web [ATR16]. Il est possible de concevoir une architecture IoT basée sur une vision centrée cloud [GBMP13]. Selon cette vision, un framework conceptuel intégrant les dispositifs ubiquitaires de détection et d'action ainsi que les applications est proposé (Figure 2.6).

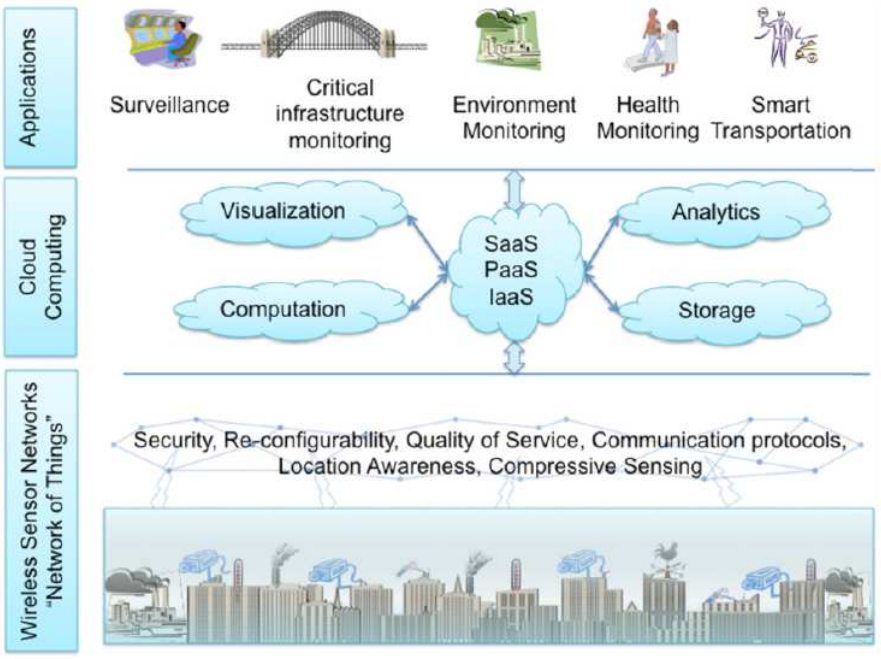


FIGURE 2.6 – Architecture IoT basée sur le cloud [GBMP13]

On reproche à la vision centrée cloud d'être très centralisée. En effet, les dispositifs physiques doivent être capables de communiquer avec les services du cloud qui sont généralement géographiquement loins et éparpillés. Un tel aspect peut être très contraignant vu que les dispositifs utilisés dans les systèmes IoT disposent généralement de resources très limitées. Pour faciliter l'accès au services du cloud, il existe des architectures

qui proposent l'utilisation de points d'accès comme intermédiaires entre les dispositifs physiques et les services du cloud. C'est ce qu'on appelle le fog computing [AFGM+15]. La figure 2.7 illustre une architecture à base du fog computing.

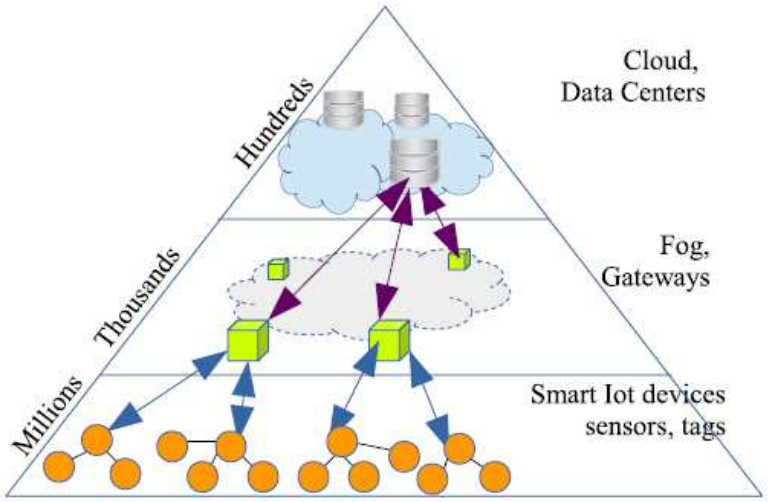


FIGURE 2.7 – Architecture IoT basée sur le fog computing [AFGM+15]

Architecture processus de l'IoT

L'internet des objets va certainement affecter les processus business. Les architectures de processus sont nécessaires afin de pouvoir structurer les processus business qui vont constituer l'IoT. En particulier, les chercheurs ont travaillé sur comment structurer les workflows afin de supporter les environnements dotés de l'informatique omniprésente (pervasive computing) [GCFP10, KKA10].

Architecture générale de l'IoT

Il n'existe pas d'accord sur une architecture unique qui convient le mieux à l'internet des objets. Un nombre d'articles proposent des critères pour l'évaluation des architectures proposées [FN08], tandis que d'autres proposent des modèles d'architectures conceptuelles afin de réaliser les exigences des objets intelligents [KKSF10]. Dans la suite nous allons présenter des architectures générales qui proposent des solutions de bout en bout (End-to-End architecture) couvrant plusieurs aspects de l'internet des objets. Ces architectures sont conçues selon un modèle en couches. Ces couches diffèrent en terme de nombre et de solutions technologiques déployées dépendemment du domaine d'application et des exigences spécifiques relatives à chaque cas. La figure 2.8 illustre une architecture générale

de bout en bout. L'adaptation de cette architecture selon le domaine d'application et les solutions technologiques utilisées sera explicité dans la suite.

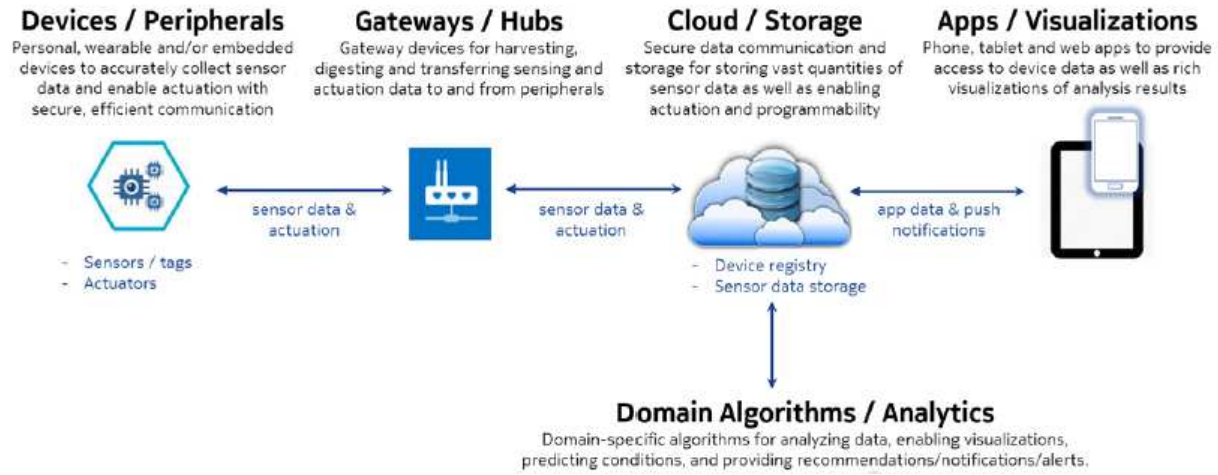


FIGURE 2.8 – Architecture IoT générale multi-couches [TM17]

Les solutions IoT peuvent être utilisées dans la conception d'une architecture déstinée à être appliquée dans un cas d'étude composé, qui consiste en plusieurs applications IoT : maison intelligente, transport intelligent, soins de la santé intelligents, etc. C'est l'objectif principal des travaux de recherche réalisés par Sarkar et al., [SSP+15] qui proposent une architecture distribuée en couches, appelée DIAT (Distributed Internet-like Architecture for Things). Cette architecture est composée de plusieurs niveaux d'abstraction afin d'aborder les problèmes relatifs à la scalabilité, l'hétérogénéité, la sécurité et l'interopérabilité. Les fonctionnalités du système IoT sont regroupées en trois couches : couche objet virtuel (VOL), couche objet virtuel composé (CVOL), et couche service (SL). Les trois couches sont responsables, respectivement, de la virtualisation de l'objet, de la composition de services, de l'exécution, de la création et de la gestion de services. Il existe un module appelé Daemon qui encapsule les trois couches citées en haut en plus d'un module transversal responsable de la gestion de la sécurité.

L'application des principes de l'IoT dans les véhicules intelligents donne naissance au nouveau paradigme Internet de Véhicules (Internet of Vehicles-IoV). Kaiwartya et al,. [KAC+16] proposent une architecture en cinq couches déstinée à l'IoV. Ces couches sont : couche de perception, couche de coordination, couche de l'intelligence artificielle, couche application, et couche business. Les auteurs explicitent les protocoles qui peuvent être utilisés afin d'accomplir les exigences fonctionnelles au niveau de chaque couche de l'architecture. En plus, un modèle de réseau de l'internet de véhicules (IoV) est proposé

dans lequel les éléments principaux du réseau sont identifiés. Des recommendations générale et technologiques sont données afin d'assurer les exigences de l'IoV en utilisant l'architecture proposée, en particulier les aspects relatifs à la sûreté, la connectivité, et les services de localisation. Une comparaison des exigences fonctionnelles et non fonctionnelles entre l'IoV et les VANETs (Vehicular Adhoc Networks) a été effectuée dans ce travail.

Il existe plusieurs travaux de recherche qui proposent des architectures conceptuelles qui ne sont pas appliquées à un cas d'étude pratique. Les auteurs [CXL+14] proposent une architecture IoT générale avec trois couches (platformes fonctionnelles) : Détection et point d'accès, Ressource et administration, Open Application. Il s'agit d'une architecture open et générique avec des interfaces et des ressources open, qui prend en considération des scénarios business différents, des technologies différentes et des exigences relatives aux applications (figure 2.9). Cette architecture IoT peut être appliquée dans neuf domaines : industrie, agriculture intelligente, logistique intelligente, transport intelligent, réseau électrique intelligent, protection intelligente de l'environnement, sûreté intelligente, soin médical intelligent, et maison intelligente. Les auteurs donnent des recommendations et des exigences générales afin de pouvoir déployer cette architecture IoT dans chaque domaine d'application, sans expliciter un cas d'étude pratique.

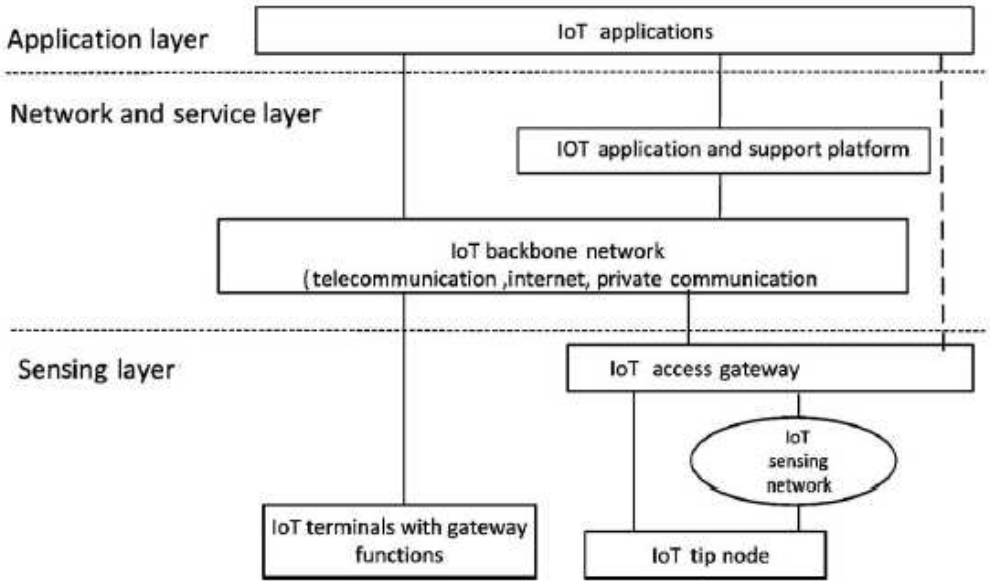


FIGURE 2.9 – Architecture IoT multi-couches [CXL+14]

Les auteurs [KAK16] proposent une architecture conceptuelle pour organiser les communautés et les sociétés pervasives basées sur les principes de l'IoT. Le but de cette architecture conceptuelle est de pouvoir intégrer le concept de communauté collaborative

sensible au processus (process-aware collaborative communities) au sein d'un framework standardisé de l'IoT. Ceci est réalisé par l'implication du concept d'automatisation de processus. Cette architecture comporte quatre couches : couche application IoT, couche support de la communauté virtuelle, couche réseau, et couche dispositif. Aucun cas d'étude pratique n'a été explicité pour cette architecture conceptuelle [KAK16]. C'est le cas également de Wu et al. [WLL+10] qui proposent une architecture conceptuelle pour les applications IoT en se basant sur le modèle TCP/IP. Cette architecture est composée de cinq couches : perception, transport, traitement, application et business.

Les solutions IoT peuvent être utilisées dans la gestion de la chaine d'approvisionnement. Une architecture IoT a été proposée pour le monitoring des conditions dans lequelles les marchandises sont transportées et stockées [ČH]. Cette architecture assure le suivi de flotte de véhicules, le contrôle et le monitoring des marchandises, ainsi que les services de localisation. L'objectif est d'améliorer la performance, la sûreté et la fiabilité du processus de transport. Dans cette proposition, les auteurs donnent des directions et des exigences générales pour l'implémentation de l'architecture IoT dans les systèmes de monitoring du transport. Une étude comparative des choix technologiques possibles est réalisée. Aucune implémentation pratique n'a été réalisée.

Il existe des propositions pour une architecture générique déstinée à être appliquée dans plusieurs domaines d'application. Ceci est réalisé par la proposition de choix technologiques et de solutions pratiques pour chaque domaine. Les auteurs [YAA17] proposent une architecture modulaire divisée en plusieurs couches, avec un ensemble de choix pour chaque couche. Les choix effectués au niveau de chaque couche peuvent être remplacés ou combinés sans affecter le fonctionnement général du système. Les différentes couches de l'architecture proposée sont : détecteurs et actionneurs physiques, processeur embarqué à basse énergie, émetteur-récepteur sans fil, point d'accès à internet, et serveur cloud pour la gestion d'application. La conception modulaire de cette architecture lui permet d'être appliquée dans plusieurs domaines comme les soins de santé, maison intelligente, système d'agriculture ainsi que le suivi d'objets. Pour chaque domaine d'application, des choix technologiques et des résultats expérimentaux sont donnés .

Résumé des architectures IoT

Le tableau 2.1 résume les types d'architectures étudiées.

TABLE 2.1 – Résumé des architectures IoT

Architecture physique	- Architecture Peer-to-Peer [ACCM10] - Architecture à base du code EPC [GC13, HM11, A+10] - Architecture à base de WSN [CBC+10, HKP+10, FPVC14, Puj06]
Architecture logicielle	- Architecture basée sur le SOA [GTK+10, Grø08, SKG+09, CGB16] - Architecture basée sur le REST [GTMW11, CBC+10, CGB+11, RSMCP16] - Architecture basée sur le cloud et le fog [ATR16, GBMP13, AFGM+15]
Architecture processus	- Processus business [GCFP10, KKA10]
Architecture de bout en bout	- Cas d'étude composé [SSP+15] - Domaine d'application de l'IoV [KAC+16] - Logistique et gestion de la chaine d'approvisionnement [ČH] - Architectures conceptuelles [CXL+14, KAK16, WLL+10] - Architecture générique[YAA17]

2.2.6 Technologies utilisées dans les systèmes IoT

Dans cette partie, nous discutons les différentes technologies utilisées dans un système IoT. En effet, un système IoT est composé de plusieurs éléments, à savoir la perception du monde physique assurée par des objets physiques en contact avec le monde externe, la communication des données à travers des moyens adéquats, le stockage et le traitement des données collectées ainsi que la présentation et la visualisation des résultats par les clients finaux. Pour réaliser tous les éléments d'un système IoT, il existe plusieurs technologies qui doivent être combinées à différents niveaux afin de pouvoir implémenter les fonctionnalités requises par le système. Ainsi, nous présentons dans la suite les technologies à déployer pour les systèmes IoT. Nous nous basons sur une présentation par couches en explicitant les technologies qui doivent être déployées à chaque couche d'un système IoT. La tableau 2.2 résume les technologies déployées à chaque couche. Ce tableau a été établi en se basant sur plusieurs travaux de recherche portant sur les technologies utilisées dans les systèmes IoT [ČH, ČH18, YAA17, AFGM+15].

La figure 2.10 illustre les technologies de communication sans fil pour l'internet des objets. Ces technologies sont classées par leurs portées qui commencent par les technologies de

TABLE 2.2 – Technologies utilisées dans les systèmes IoT

IoT architecture layers	Enabling technologies
Business	- Semantic : RDF, OWL, EXI, Data mining, Dataware house.
Application	- Smart Grid, Smart home, Smart city, Smart agriculture, Shipping, etc.
Middleware	- Fog and cloud platforms : OpenIoT, Google Cloud, Arkessa, ThinkWorks, Oracle IoT, Nimbits, Hadoop, etc. - Data processing mechanisms : Data mining, Big Query, Apache Hadoop, SPARQL, SciDB, Semantic technologies (RDF, OWL, EXI, W3C,...), etc. - Data storage : MongoDB, Cassandra, Hadoop, Hbase, CouchDB, etc. - OS : contiki, TinyOS, LiteOS, Riot OS, Android, etc.
Network	- Wireless communication technologies : RFID, NFC,UWB, Bluetooth, BLE (Bluetooth Smart), Z-wave, WiFi (IEEE 802.11x), WiFiDirect, LTE (4G), 5G Zigbee, 6LoWPAN, WiMAX, GPRS, EDGE, LPWAN (Sigfox, LoRA), Satellite communication, Cognitive radio, etc.
Devices	- Low power embedded processors and microcontrollers : Raspberry Pi, Arduino, Adafruit, Phidgets, intel galileo, etc. - Sensors : smart sensors, wearable sensing devices, embedded sensors, actuators, RFID tag, etc.

proximité comme le NFC et la RFID, jusqu'aux technologies utilisées dans un réseau sans fil très étendu comme la 4G, la 5G ou la communication par satellite.

2.2.7 Modèles de communication dans une architecture IoT

Quelque soit le type d'architecture choisi pour la construction du réseau IoT, il existe différents modèles de communication entre les éléments du réseau IoT. En particulier, nous présentons quatre modèles possibles [TATM15]. Dans une seule architecture IoT, plusieurs modèles peuvent être utilisés à la fois.

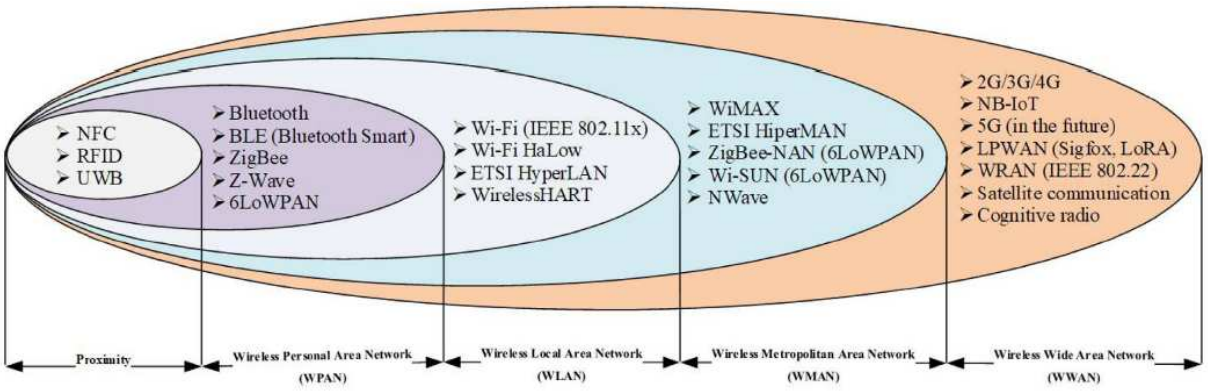


FIGURE 2.10 – Technologies de communication sans fil pour l'internet des objets [ČH18]

Communication Device-to-Device

Le premier modèle concerne la communication directe entre deux dispositifs physiques. La figure 2.11(a) illustre un exemple de communication entre deux dispositifs développés par deux différents fabricants. Il s'agit d'un interrupteur qui communique avec une ampoule. Afin de réaliser cette communication, les deux dispositifs doivent se mettre d'accord sur les protocoles de communication à chaque couche, l'architecture à adopter (point-à-point, client/serveur,etc.), les aspect relatifs à la sécurité et la gestion des données,etc.

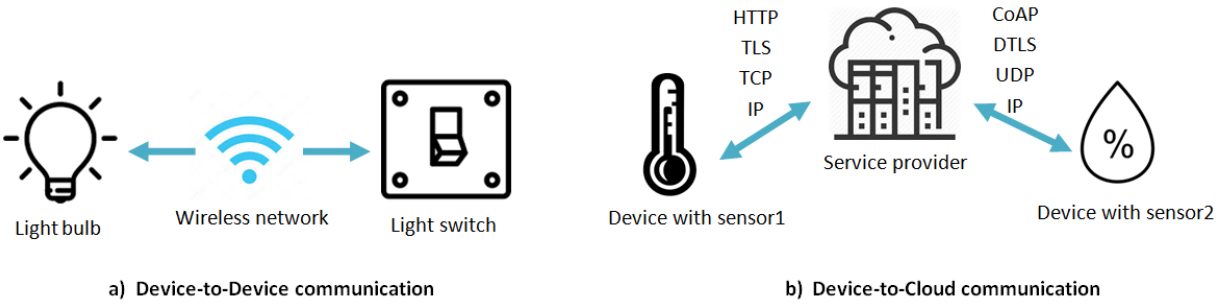


FIGURE 2.11 – (a) Modèle de communication Device-to-Device (b) Modèle de communication Device-to-Cloud [TATM15]

Communication Device-to-Cloud

Le deuxième modèle concerne la communication entre un dispositif physique et un fournisseur de service (service provider). La figure 2.11(b) illustre un exemple de communication de détecteurs physiques avec le serveur cloud qui fournit les services dans ce cas. Dans ce modèle, les deux entités qui communiquent doivent supporter les mêmes protocoles de communication (IP based, UDP, CoAP).

Communication Device-to-Gateway

Le troisième modèle de communication concerne la communication entre un dispositif physique et un point d'accès (gateway-passerelle). Ce modèle de communication est utilisé lorsque les dispositifs physiques du système IoT ne supportent pas tous les mêmes protocoles de communication. Dans ce cas, le point d'accès (gateway) est considéré comme un intermédiaire assurant la communication entre les dispositifs physiques et les services fournis par le cloud (Figure 2.12(a)). En utilisant ce modèle de communication, on assure l'interopérabilité du système.

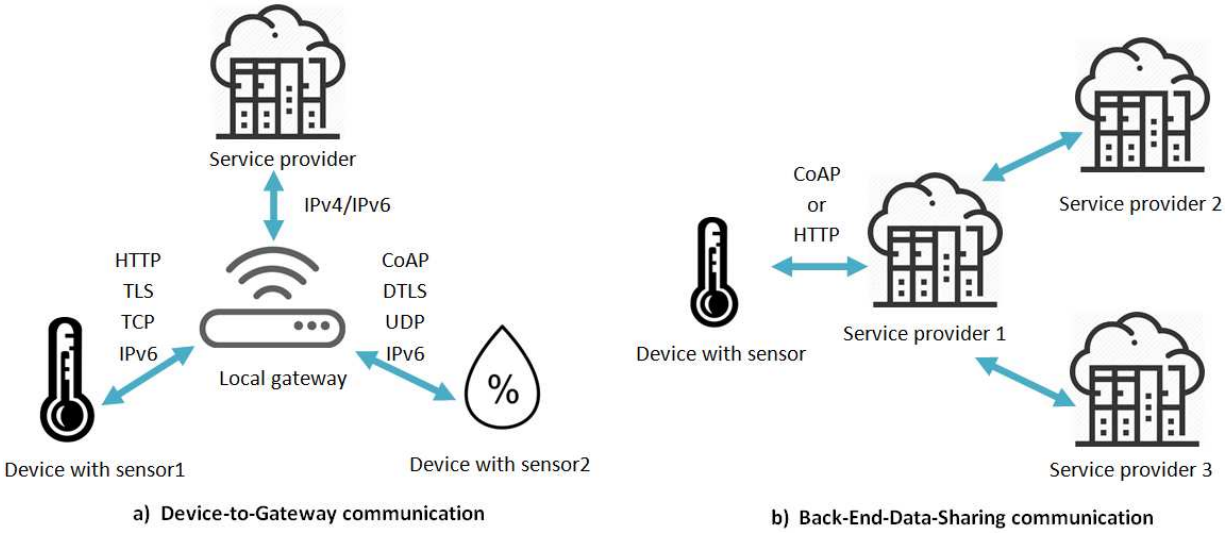


FIGURE 2.12 – (a) Modèle de communication Device-to-Gateway (b) Modèle de communication Back-End Data Sharing [TATM15]

Communication Back-End Data Sharing

Dans un système IoT, le fournisseur de service peut ne pas être muni de toutes les opérations nécessaires pour fournir le service demandé par les dispositifs physiques. Ainsi, ce fournisseur de service va faire appel à d'autres entités pour effectuer des opérations de stockage, de calcul, de récupération de données, etc,. C'est ce qui est explicité par ce dernier modèle de communication dans une architecture IoT (Figure 2.12(b)).

2.3 Défis des systèmes IoT

Il n'existe pas encore d'unanimité pour une architecture IoT valable pour tous les domaines d'application. Chaque communauté scientifique donne des définitions et des

directions relatives à ses domaines d'intérêt. Par conséquent, les défis des systèmes IoT diffèrent également selon les perspectives adoptées. Dans notre travail de recherche, nous allons établir les différents défis rencontrés lors de l'implémentation des systèmes IoT (figure 2.13).

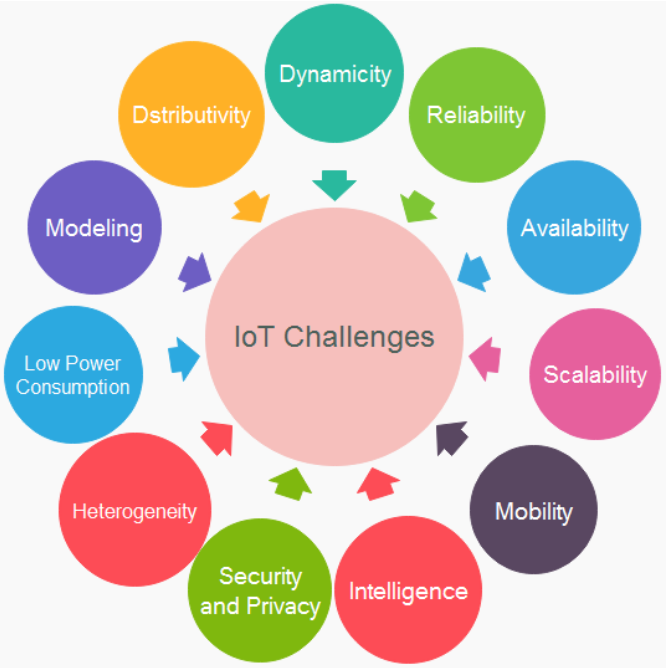


FIGURE 2.13 – Défis des systèmes IoT

- Architecture : La réalisation d'un système IoT selon une architecture donnée est primordiale. En effet, cette architecture permet d'acquérir un ensemble d'exigences requises par le système. Dans ce sens, la proposistion d'une architecture pour les systèmes IoT est un défi essensiel à relever. En effet, il existe dans la littérature plusieurs propositions d'architectures IoT. Certaines propositions concernent des architectures physiques, d'autres proposent des architectures logicielles, tandis qu'il y a une autre école qui propose des architectures de bout en bout (End-to-End). Une architecture doit satisfaire les besoins fonctionnels et non fonctionnels du système en question. Il faut donc être en mesure de choisir l'architecture adéquate selon les objectifs et les intérêts spécifiques au domaine d'application en question [AFGM+15, ČH18, Sta14].

- Modélisation des systèmes IoT : Les systèmes IoT sont des systèmes complexes car ils intègrent une grande variété de périphériques et de services qui doivent être bien gérés et bien intégrés. La communication et l'interaction entre les composants des systèmes IoT devraient également être étudiées. C'est pourquoi il devrait y avoir plus d'intérêt et

de travail sur la question de la modélisation des systèmes IoT. La modélisation formelle de l'architecture logicielle, de l'architecture physique, de l'architecture de bout en bout ainsi que des différents protocoles de communication et de sécurité semble être une problématique prometteuse qui permettra plus de contrôle sur la gestion des systèmes IoT [ČH18]. Dans la littérature, il existe des travaux sur la modélisation formelle des protocoles de communication destinés aux systèmes IoT [Azi16, HBGS07]. Il existe également des propositions pour la modélisation des systèmes IoT utilisant des systèmes multi-agents [AK15, WWZ+16]. Dans un travail précédent[HK15, HKK17, HKK18], nous proposons un modèle formel pour notre architecture ReDy adaptée aux systèmes IoT.

- Distributivité : un système IoT est caractérisé principalement par son aspect distribué. Il hérite ainsi des défis rencontrés lors de la conception des systèmes distribués combinés avec les caractéristiques spécifiques aux systèmes IoT. Nous citons dans la suite quatre principaux défis relatifs aux systèmes IoT comme étant un système distribué [MSDPC12, TM17, ATR16, RMJPC15].

a) Construire un réseau d'entités : Il s'agit de faire des décisions pour le choix de méthode permettant la conception de l'architecture suivant laquelle le réseau d'entités est construit. Ce choix doit être en conformité avec les exigences relatives à l'internet des objets [MSDPC12].

b) Communication au sein du réseau distribué : Choix du protocole de communication afin de répondre aux exigences du système [MSDPC12, TM17].

c) L'intelligence distribuée : Les entités du système doivent être capables d'effectuer des calculs pour pouvoir prendre des décisions sans intervention externe [MSDPC12, BV17].

d) La gestion des données massives échangées dans le réseau distribué [MSDPC12, ATR16].

- Aspect dynamique : un système IoT est généralemnt composé d'un très grand nombre d'entités. Au cours du fonctionnement du système, des entités peuvent rejoindre ou quitter le système. Un tel changement ne doit en aucun cas affecter le fonctionnement du système qui doit continuer à fournir les services requis. Il s'agit bien d'assurer l'aspect dynamique et potentiellement migratoire des systèmes IoT [Ray18, SSP+15, TM17].

- Disponibilité : la disponibilité de l'IoT doit être réalisée au niveau physique et au niveau logiciel. An niveau physique, les appareils en contact avec l'environnement doivent

être toujours en marche prêts à capter les données nécessaires. Au niveau logiciel, les services fournis par le système doivent être toujours accessibles par l'utilisateur final [RMJPC15, ČH18, AFGM+15].

- Fiabilité : consiste à assurer le bon fonctionnement du système en respectant ses spécifications. La fiabilité doit être assurée au niveau du réseau de communication en utilisant des protocoles de communication fiables afin d'assurer la tolérance aux pannes. La fiabilité doit être assurée également au niveau physique (appareils) ainsi qu'au niveau logiciel (services). La fiabilité et la disponibilité sont étroitement liées vu que la fiabilité consiste à assurer la disponibilité du système dans le temps [RMJPC15, ČH18, AFGM+15].

- Le passage à l'échelle (Scalability) : c'est l'un des défis les plus importants dans les systèmes IoT. En effet, il faut être en mesure de concevoir des architectures modulaires permettant l'extension du système sans affecter sa qualité de fonctionnement [ČH18, AFGM+15, RMJPC15, BV17].

- Hétérogénéité : les systèmes IoT sont caractérisés par l'hétérogénéité de ses objets physiques, d'où la nécessité d'un middlware permettant d'intégrer ces différents appareils hétérogènes dans un seul système. Le défi de l'hétérogénéité concerne également les couches logicielles, vu que les services proposés par le système peuvent également être hétérogène et nécessitent un travail de standardisation afin de pouvoir supporter plusieurs choix technologiques [ČH18, Ray18, TM17, RMJPC15, BV17].

- Intelligence et prise de décision : les dispositifs des systèmes IoT doivent être connectés au réseau, et doivent implémenter des solutions et des algorithmes leurs permettant de prendre des décisions [ČH18, Ray18].

- Mobilité : les appareils IoT sont généralement de petits appareils pouvant être déplacés en permanence. Des études sont effectuées afin de maintenir le bon fonctionnement du système malgré la mobilité des appareils [AFGM+15, VZS10, SDG13].

- Sécurité et confidentialité : il est nécessaire d'avoir une politique de sécurité et de confidentialité dans un système IoT couvrant toutes les couches de l'architecture IoT. En fait, il existe des protocoles qui devraient être implémentés pour assurer cet aspect. Cependant, il reste encore du travail à faire afin d'adapter les protocoles existants au contexte en évolution très rapide des systèmes IoT [ČH18, AFGM+15, RMJPC15].

- Faible consommation d'énergie en calcul et en communication : les dispositifs des systèmes IoT sont généralement limités en ressources, notamment en termes de capacité énergétique. Ce problème doit être pris en considération lors de la conception des systèmes IoT [MSDPC12].

- Sensibilité au contexte [YXM+14, RMJPC15].

2.4 Problématique de la thèse

On ne peut jamais répondre à tous les défis à la fois. Dans notre contribution, nous travaillons sur les applications IoT caractérisées par leur aspect distribué afin d'assurer la fiabilité et la disponibilité dans un environnement dynamique déstiné à être évolutif. En se basant sur les prévisions relatives à l'avancement dans les différents domaines d'application de l'IoT (voir figure 2.14), nous pouvons constater que nous sommes dans l'ère de l'intégration de l'IoT dans les services publics. Plusieurs applications existent dans ce domaine. Afin de bien étayer notre proposition, nous avons choisi les Smart Grids comme cas d'étude explicatif des différentes étapes de notre contribution. En effet, les Smart Grids commencent à intégrer des solutions IoT ce qui impose aux constructeurs de s'adapter aux nouveaux besoins relatifs à des applications d'une taille aussi grande que les Smart Grids. Ce type d'application requiert une grande sûreté lors de l'implémentation de solutions le concernant. Pour cette raison, nous avons choisi l'analyse formelle comme outil pour vérifier les critères recherchés à savoir la fiabilité, le dynamisme et l'extensibilité.

Dans la suite, nous allons présenter quelques travaux de recherche qui portent sur l'utilisation des méthodes formelles ou semi formelles pour aborder les systèmes IoT. Il est à noter qu'il s'agit d'une thématique de recherche très nouvelle et très prometteuse vu le besoin croissant d'avoir des applications IoT qui respectent le besoin de l'utilisateur. Par la suite, nous allons présenter les différents concepts et outils utilisés pour mener la validation formelle dans notre travail.

2.5 Validation formelle des architectures IoT

Dans cette partie, nous allons commencer par citer quelques travaux de recherche qui ont porté sur la thématique de l'analyse formelle des architectures IoT, ensuite nous allons présenter notre approche pour réaliser la validation formelle en expliquant les différentes notions nécessaires à la compréhension de cette approche. Enfin nous allons présenter la

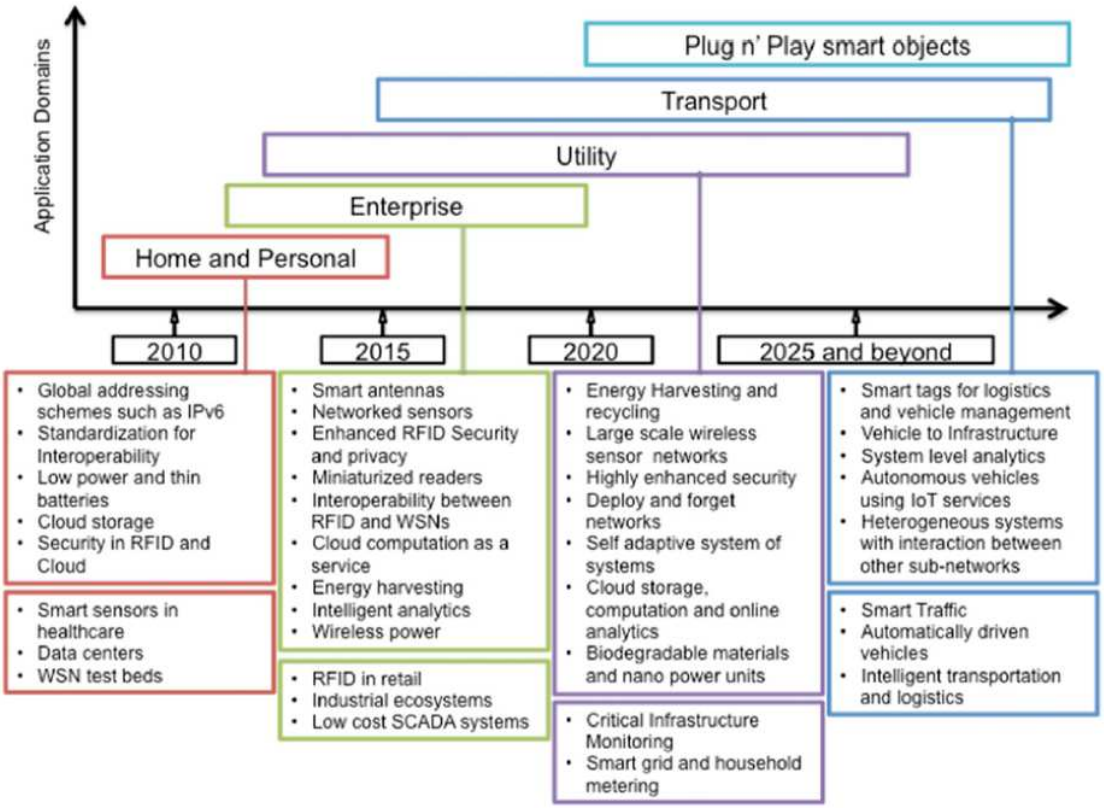


FIGURE 2.14 – Etude des domaines d'application des systèmes IoT [GBMP13]

boite à outils CAPD que nous avons utilisé pour mener le travail de validation formelle.

2.5.1 Analyse formelle des architectures IoT dans la littérature

La modélisation formelle des systèmes IoT est un domaine très intéressant, vu qu'il permet aux systèmes IoT d'acquérir une maturité en assurant le bon comportement du système avant l'implémentation. Dans la littérature, le besoin en modélisation formelle des systèmes IoT est évident et clairement exprimé [ČH18, Azi16, Sta14, BV17]. Cependant, les travaux de recherche portant sur ce paradigme concerne généralement juste un aspect parmi les différents aspects des systèmes IoT. En effet, dans les travaux existants, l'utilisation de la modélisation formelle s'applique principalement sur des protocoles spécifiques utilisés dans les systèmes IoT [Azi16, HBGS07, GGMT08], sans proposer une modélisation complète de l'architecture représentant le système IoT.

Plusieurs outils formels permettent de modéliser formellement les systèmes IoT et ainsi exploiter le modèle formel en l'analysant, en simulant le comportement et en vérifiant des

propriétés. Dans ce cadre, des travaux ont proposé l'outil PRISM [RSL17] pour modéliser le comportement des systèmes IoT temps réel.

Les Réseaux de Petri ont été proposés [YTN16] pour modéliser un service IoT par une approche multi-agents, puis le modèle a été utilisé dans la simulation et la vérification.

L'algèbre de processus synchrone temporisée TPi a été utilisée [Azi16] pour modéliser formellement un protocole de communication MQTT destiné aux applications IoT. Ce modèle a été utilisé pour étudier et analyser des propriétés concernant la sécurité du protocole MQTT.

Les techniques du model-checking probabilistic [HBGS07] ont été utilisées pour étudier des propriétés de fiabilité et d'exactitude (correctness) et pour détecter des ambiguités dans les protocoles publish/subscribe. Nous retrouvons également des travaux qui ont été menés pour vérifier le comportement des protocoles de composition de services dans des architectures orientées service en utilisant les techniques du model-cheking probabilistic [GGMT08].

Des travaux ont proposé d'utiliser les techniques de tests basés sur les modèles pour tester les aspects de sécurité et de fiabilité des dispositifs d'un système IoT dans le cadre des applications des villes intelligentes [KCL+17] et des maisons intelligentes [MEAV+18]. L'idée est de modéliser formellement le système et puis utiliser le modèle formel pour générer des tests à excécuter sur le système IoT puis utiliser des critères de couverture basées sur le modèle pour s'assurer d'avoir testé tous les aspects modélisés.

Des techniques de modélisation semi-formelle utilisant les systèmes multi-agent [WWZ+16] ont été utilisées pour modéliser une usine intelligente dans le contexte de l'industrie 4.0. L'analyse a concerné la flexiblité et l'auto-organisation. Cette modélisation a permis d'éviter les interblocages grâce à la simulation qu'une modélisation semi-formelle permet.

2.5.2 Approche adoptée de validation formelle

Vérification fonctionnelle

La validation des systèmes concurrents comprend deux branches : la vérification fonctionnelle et l'évaluation des performances. La vérification fonctionnelle consiste à vérifier que le système se comporte correctement, c'est à dire qu'il répond aux exigences fonctionnelles de la spécification. La vérification fonctionnelle se concentre sur les aspects

qualitatifs. L'évaluation des performances consiste à mesurer les temps de réponse et la communication de bout en bout du système, cette branche de validation des systèmes concurrents se concentre sur les aspects quantitatifs. Dans cette phase de validation, nous nous concentrons sur la vérification fonctionnelle, nous nous intéressons à l'exactitude du comportement du système.

Communication asynchrone et notion de rendez-vous

Les architectures des systèmes IoT et le modèle formel que nous utilisons font partie des systèmes distribués à communication asynchrone. Dans cette phase, nous nous intéressons aux problèmes posés par ce type de systèmes concurrents, notamment en matière de validation formelle avant de se lancer dans sa mise en œuvre, afin de garantir une solution correcte et évolutive. En effet, la mise en œuvre d'algorithmes distribués dans la littérature a connu deux orientations principales :

- La recherche d'invariants, visant à forcer l'ordre des opérations du système et à fonctionner dans l'esprit d'un algorithme séquentiel. Cette orientation crée des algorithmes de construction correcte mais qui ne sont pas optimaux et qui ne profitent pas de l'extensibilité offerte par la distribution du système.

- La seconde orientation est une approche empirique, c'est à dire que nous commençons par construire un algorithme puis nous passons par une phase de validation. Cette orientation est la plus utilisée dans les systèmes distribués. Dans la suite de ce travail, nous nous concentrons sur cette orientation.

Les algorithmes distribués utilisent des concepts similaires à ceux des réseaux, notamment l'envoi de messages, leur diffusion et leur reconnaissance. L'une des approches liées à la communication entre les différents composants du système est la notion du rendez-vous, ce qui représente une implémentation de la coordination entre les composants du système [QS82].

Un rendez-vous R entre n processus exprime que les n processus ne peuvent exécuter l'événement R que si les n processus sont prêts à l'exécuter. Dans un système distribué, le rendez-vous est le mécanisme de communication des processus complètement asynchrones. Le rendez-vous représente un point de synchronisation.

Généralement dans les systèmes distribués asynchrones, chaque processus est prêt à planifier un ensemble de rendez-vous. Les nominations qui peuvent être faites sont celles

pour lesquelles tous les processus pertinents sont prêts. Chaque processus a un choix non déterministe de rendez-vous à faire.

Les problèmes spécifiques aux rendez-vous sont :

- Coordination synchrone : si un processus s'engage sur un rendez-vous, tous les processus impliqués doivent s'y engager.

- Exclusion : un processus ne peut être effectué que sur un seul rendez-vous à la fois. Cela exprime l'exécution complètement asynchrone des rendez-vous, c'est-à-dire que si deux rendez-vous sont exécutés dans un système, cela ne peut jamais se produire simultanément mais dans un certain ordre : il y a un rendez-vous qui a lieu avant l'autre. Si cette dépendance à l'ordre n'est pas requise, cela implique l'existence de deux scénarios d'exécution relatifs aux deux ordres possibles qui sont possibles de manière non déterministe.

Ces problèmes doivent être validés pour implémenter la sémantique d'un rendez-vous. Par la suite, nous pensons que le rendez-vous est correctement mis en œuvre et nous nous concentrons sur les propriétés que ces systèmes doivent vérifier et sur les principaux problèmes qui peuvent exister dans un système qui inclut une sémantique de communication basée sur les rendez-vous.

Inter-blocage et blocage opérationnel

Un programme est correct lorsqu'il satisfait certaines propriétés. La propriété du programme concurrent est une affirmation qui est vraie pour toute exécution légale de ce programme. En général, les propriétés dépendent beaucoup de la politique d'ordonnancement utilisée, en particulier de la sémantique des rendez-vous utilisée dans notre contexte. Il existe deux catégories de proprétés qu'un programme concurrent doit satisfaire : la sûreté et la vivacité [LO82]. Une propriété de vivacité (liveness) exprime que quelque chose de bien se produit bien, contrairement aux propriétés de sûreté (safety) qui expriment que quelque chose de mal ne se produit jamais.

Deux propriétés importantes pour les programmes concurrents sont l'absence d'inter-blocage (deadlock freeness) et l'absence de blocage opérationnel (livelock freeness) [BPP98]. Un inter-blocage est un état où les processus sont bloqués en attendant quelque chose qui ne se produira jamais. Le blocage opérationnel est une situation où les processus exécutent des instructions inutilement, dans le sens qu'ils ne peuvent jamais faire de

progrès constructifs. Alors que l'inter-blocage peut être considéré comme une violation de la sûreté et de la vivacité à la fois, le blocage opérationnel n'est généralement pas considéré comme une violation de la sûreté, mais plutôt de la vivacité.

Les travaux concernant l'absence d'inter-blocages ou de blocages opérationnels reposent sur l'une des deux approches :

- La mise en place d'un système sans inter-blocage (respectivement sans blocage opérationnel) par construction

- La mise en place du système puis sa validation.

La première approche donne naissance à une série de conditions que le système doit respecter pour ne pas contenir d'inter-blocages (ou blocages opérationnels) [CEG, 1971], le principal inconvénient de cette approche est que les solutions trouvées ne sont généralement pas optimales, puisque nous tentons de définir des règles générales pour l'absence d'interblocages (respectivement blocages opérationnels), tandis que le cas spécifique de notre système cherche une solution optimale qui ne provoque aucun inter-blocage (ou blocage opérationnel).

Cela nous ramène à la deuxième approche qui construit un système puis le valide. Dans ce contexte, différentes approches ont été proposées. Leur point commun est l'expression de l'algorithme distribué dans un formalisme mathématiquement correct pour pouvoir tester la correction à travers la preuve mathématique ou à travers des analyses statiques ou dynamiques de comportements possibles, voire même à travers la vérification du modèle (model checking), c'est-à-dire l'exploration de l'espace des états possibles du système de manière exhaustive et en démontrant qu'il n'y a jamais un inter-blocage (respectivement un blocage opérationnel).

Algèbre des processus

L'algèbre des processus [Bae05], dite aussi calcul des processus, fournit des outils pour décrire les interactions, les communications et les synchronisations de haut niveau entre un ensemble de processus indépendants. Cette algèbre fournit des lois pour la manipulation et l'analyse des descriptions de processus. Par exemple, nous pouvons comparer deux processus selon une loi d'équivalence.

Une algèbre de processus se caractérise par trois caractéristiques principales :

- Premièrement, la représentation des interactions entre processus indépendants par le biais de communications (transmission de messages ou rendez-vous entre processus sur une porte de communication) et non sous forme de modification d'une variable partagée. Cela exprime une vue en boîte noire du système dans laquelle le système peut être analysé en analysant les communications entre les éléments du système sans avoir à traiter l'état interne des variables du système (vue en boîte blanche).

- Deuxièmement, la description des processus qui utilisent un ensemble restreint de primitives et d'opérateurs pour composer ces primitives (les primitives et les opérateurs sont définis dans la grammaire de chaque langage d'algèbre des processus).

- Troisièmement, la définition algébrique des lois correspondant aux opérateurs appliqués au processus. Cela permet de manipuler les expressions de processus avec le raisonnement sur des équations.

Systèmes à transitions étiquetées (LTS)

Les systèmes à transitions étiquetées (Labelled Transition Systems, LTS) sont des diagrammes d'états-transitions, dans lesquels les états ne contiennent aucune information à part une indication de l'état initial. Des informations représentant des étiquettes ou des actions sont attachées aux transitions.

Logique temporelle

Une logique temporelle est un système de règles et de symbolismes utilisés pour représenter et raisonner sur des propositions qui expriment l'aspect temporel. Avec une logique temporelle, des déclarations telles que "un événement A est toujours vrai", "un événement A se produira inévitablement" ou "un événement A sera toujours vrai tant qu'un événement B s'est produit" peuvent être exprimées. Les logiques temporelles sont utilisées pour exprimer des propriétés qui seront vérifiées dans le contexte d'une vérification formelle des systèmes.

Différentes logiques temporelles ont été proposées dans la littérature, dont deux familles principales :

- Logique linéaire [Pnu77] (logique LTL et ses dérivées) qui raisonne sur les traces des exécutions d'un système.

- Logiques avec branchement [BAPM83] (logique CTL et ses dérivées) qui raisonnent dans les branches de l'arborescence du LTS du système.

Dans notre validation, nous utilisons une logique temporelle avec branchement (branching logic), pour montrer son intérêt par rapport à une logique linéaire, nous présentons l'exemple classique d'une machine à café : nous considérons que deux machines à café (M1) et (M2) fonctionnent selon les deux automates suivants (figure 2.15) :

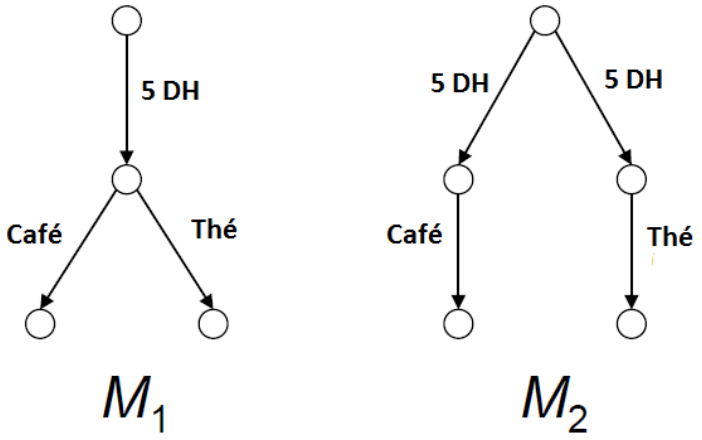


FIGURE 2.15 – Deux LTS représentant le fonctionnement des machines (M1) et (M2)

La machine (M1) fonctionne correctement : une fois la somme de 5 DH fournie, la machine est dans un état où vous pouvez choisir de boire du café ou du thé. Pour la machine (M2), lorsque la somme de 5 DH est fournie, la machine passe de manière non déterministe à l'un des deux états : un état dans lequel la machine ne peut fournir que du café et un autre état dans lequel la machine ne peut fournir que du thé. La logique avec branchement permet de détecter qu'il existe un état de la machine (M2) dans lequel il ne peut pas fournir de thé et un autre dans lequel il ne peut pas fournir de café.

En logique linéaire, les deux machines (M1) et (M2) ont des traces équivalentes : "5 DH, café" et "5 DH, thé". Dans cette logique qui raisonne sur les traces, le problème de la machine (M2) ne peut jamais être détecté.

2.5.3 Boite à outils CADP

Dans notre travail, nous utilisons la boîte à outils CADP (Construction and Analysis of Distributed Processus), qui est une boîte à outils modulaire pour l'analyse des systèmes asynchrones. Cette boîte à outils met en œuvre les résultats de la théorie de la concurrence

et se concentre sur la famille des algèbres de processus. CADP est une boîte à outils développée dans les années 80 par l'équipe VASY puis l'équipe CONVECS dès 2012 de l'Inria-France par l'ajout de nouveaux outils. CADP contient une cinquantaine d'outils formels [GLMS13].

Représentation LTS dans CADP

Les LTS sont représentés explicitement ou implicitement :

- Un LTS explicite est une énumération de tous les états, transitions et étiquettes. Il est enregistré dans un fichier BCG (Binary Coded Graph), qui est une forme d'archivage qui permet d'enregistrer de grands LTS. CADP fournit une barre d'outils pour gérer les fichiers BCG.

- Un LTS implicite est une représentation locale dite à la volée d'un LTS. En effet, le système est défini par un état initial et une fonction de calcul des successeurs. De cette façon, il est possible de tester des systèmes sans générer explicitement de LTS. Par exemple, il est possible de trouver un contre-exemple sans avoir à générer l'intégralité du système LTS, qui peut ne pas pouvoir être généré dans la mémoire disponible ou qui peut prendre au moins beaucoup de temps pour être entièrement généré. CADP propose l'API Open/Caesar pour manipuler le LTS implicite, qui est indépendant du langage dans lequel le LTS implicite est exprimé.

Langage de modélisation LNT

CADP prend en charge un certain nombre de langages de modélisation formels. Nous nous concentrons sur le langage LNT [CCG+17], qui est basé sur l'algèbre de processus. LNT présente deux notions d'algèbre de processus qui ne sont pas possibles dans les langages de programmation fonctionnels classiques :

- La composition parallèle des processus (avec un schéma de communication asynchrone).

- Le choix non déterministe au sein d'un processus.

De plus, LNT contient tous les formalismes classiques des langages de programmation, à savoir les boucles, les tests (if ... then ... else ...), les appels récursifs, les variables, les fonctions, les types de données. Des structures de données sont également possibles.

Langage de logique temporelle MCL

Le langage de logique temporel MCL (Model Checking Language) [MT08a] est un langage logique avec branchement. MCL est une extension du mu-calculus, qui offre la possibilité d'enregistrer des données au niveau d'une action, puis de les utiliser dans la propriété MCL spécifiée. MCL permet un traitement de complexité linéaire via des propriétés sans stockage des données.

La boîte à outils CADP contient le vérificateur de modèle (Model checker) EVALUATOR 4.0, qui permet de vérifier les propriétés MCL explicitement sur un LTS de type BCG, ou implicitement (à la volée) sur une spécification de type LNT.

2.6 Cas d'illustration : Les Smart Grids

Dans cette partie, nous allons présenter le réseau électrique intelligent (Smart Grids) comme cas d'étude. Pour ce faire, nous commençons par définir le contexte général des Smart Grids et les principaux éléments le composant, ainsi que les exigences principales de ce type de systèmes.

2.6.1 Présentation générale des Smart Grids

Le réseau électrique traditionnel est resté inchangé pendant presque un siècle. Les avancées technologiques que le monde connait nécessitent une adaptation du réseau électrique existant afin de pouvoir contourner un ensemble de nouveaux défis à savoir l'augmentation de la consommation d'énergie par habitant et l'augmentation de la population. Ceci nécessite de plus en plus de production d'énergie alors que les ressources conventionnelles sont limitées et arrivent de plus en plus à leur limite. En plus, les énergies conventionnelles ont un impact négatif sur l'environnement et participent aux changements climatiques que connait la planète. La réduction de l'émission des gazs à effets de serre nécessite la réduction de la production des énergies conventionnelles notamment les énergies fossiles.

C'est pourquoi l'humanité se dirige vers des énergies moins poluantes et à ressouces durables dites renouvelables et qui sont de part la nature de leurs ressources (solaires, éoliennes, hydrauliques, géothermiques, bio-masses, etc.) distribuées. La distribution de ces ressources durables impose une distribution de leur production. Dans cette nouvelle optique, on parle alors de génération distribuée d'énergie. La production de l'énergie

ne se fait plus uniquement dans de grandes centrales, mais elle devient aussi ditribuée géographiquement sur des petites centrales d'énergies renouvelables pouvant injecter de l'électricité directement dans le réseau. Cette distribution de la production est en plus un point positif vu qu'elle permet de rapprocher la production de la consommation qui était toujours très éparpillée et distribuée. Une production électrique proche géographiquement de la consommation économise les pertes d'énergie électrique au niveau du transport.

D'autre part les énergies renouvelables dépendent de la disponibilité de leurs ressources naturelles qui sont variables dans le temps. Ces énergies sont dites alors *intermitantes* car la production des énergies renouvelables dépend des conditions météorologiques. Par exemple, l'énergie solaire photovoltaique sera présente tant que la luminosité du soleil est présente durant la journée et sera absente la nuit. L'énergie éolienne est présente tant que le vent souffle et s'arrête dès sont arrêt.

Prenant en compte cette nouvelle configuration du réseau électrique et cette nouvelle dynamique de ces ressources, une nouvelle complexité est alors introduite au niveau de la gestion et de la stabilisation du réseau électrique. C'est pourquoi la notion de réseau électrique intelligent (*Smart Grid*) est introduite. Ces réseaux intelligents intègrent un réseau de données en parallèle du réseau de distribution d'électricité classique et proposent des outils de suivi, d'analyse et de contrôle de la production et de la consommation de l'énergie au niveau du réseau électrique [Dil20, BCFD16].

Les Smart Grids utilisent des technologies de communication moderne et des outils de contrôle automatisés afin d'assurer qu'à tout instant, la quantité d'électricité injectée dans le réseau soit égale à la quantité d'électricité prélevée par les consommateurs. Contrairement au réseau traditionnel où l'interaction se fait dans un seul sens afin d'acheminer l'électricité au point de consommation, l'idée principale dans un Smart Grid consiste à rendre tout point connecté au réseau interactif en assurant un dialogue dans les deux sens. Ainsi, l'électricité et l'information sont échangées entre les différents éléments du réseau afin d'assurer un équilibre au niveau du réseau électrique [Dil20, BCFD16].

On ne parle plus alors de producteur absolu et de consommateur absolu, mais on parle de noeud du réseau, tel qu'à un instant donné chaque noeud peut être globalement consommateur ou globalement producteur. Chaque noeud est composé d'un point d'accès et de connexion au réseau qui a un pouvoir de décision, et d'au moins un élément consommateur ou producteur ou les deux à la fois. La figure 2.16 représente un schéma générique d'un Smart Grid. Cet exemple comprend des noeuds principalement producteurs

comme les centrales conventionnelles, des noeuds principalement consommateurs comme les usines, en plus des noeuds assez équilibrés comme les bâtiments verts qui sont en général très efficace énergétiquement vu qu'ils comportent des sources de production d'énergies renouvelables locales. Ces derniers peuvent jouer le rôle de producteurs à certains moments et le rôle de consommateurs à d'autres moments. Les noeuds du réseau sont doublement connectés au réseau par le réseau de distribution d'électricité mais aussi par un réseau de données.

2.6.2 Exigences générales des Smart Grids

En se basant sur les défis des Smart Grids présentés en haut, ces derniers doivent répondre à un ensemble d'exigences que nous présentons dans la suite :

- Gestion dynamique de l'adhèsion des composants au système : les noeuds dans un réseau intelligent sont caractérisés par leur aspect dynamique, c'est à dire qu'un noeud peut se connecter ou se déconnecter du réseau à tout moment donné avec ou sans préavis. L'aspect critique du réseau électrique doit être en mesure d'assurer un bon fonctionnement sans interruption même si des noeuds s'ajoutent au réseau ou le quittent.

- La fiabilité : au sein du réseau intelligent, il faut avoir une grande fiabilité et ce en assurant une grande tolérance aux pannes. La communication entre les différents noeuds du réseau qui sont en marche doit être toujours établie même au cas où il existe des noeuds défaillants.

- Dans un réseau iltelligent, il faut assurer l'existence de capteurs qui sont en contact avec l'environnement pour capter tout événement ou information signifiante. En général, ces informations et événements sont en lien avec la production ou la consomation de l'électricité. En se basant sur les données collectées par les capteurs, il faut être en mesure de prendre des décisions adéquates. Ces décisions sont acheminées à des entités en contact avec l'environnement et qui sont responsables de l'exécution des décisions.

- Le réseau intelligent est caractérisé par sa distribution géographique sur une zone donnée. Il faut donc être en mesure de couvrir la zone géographique concernée.

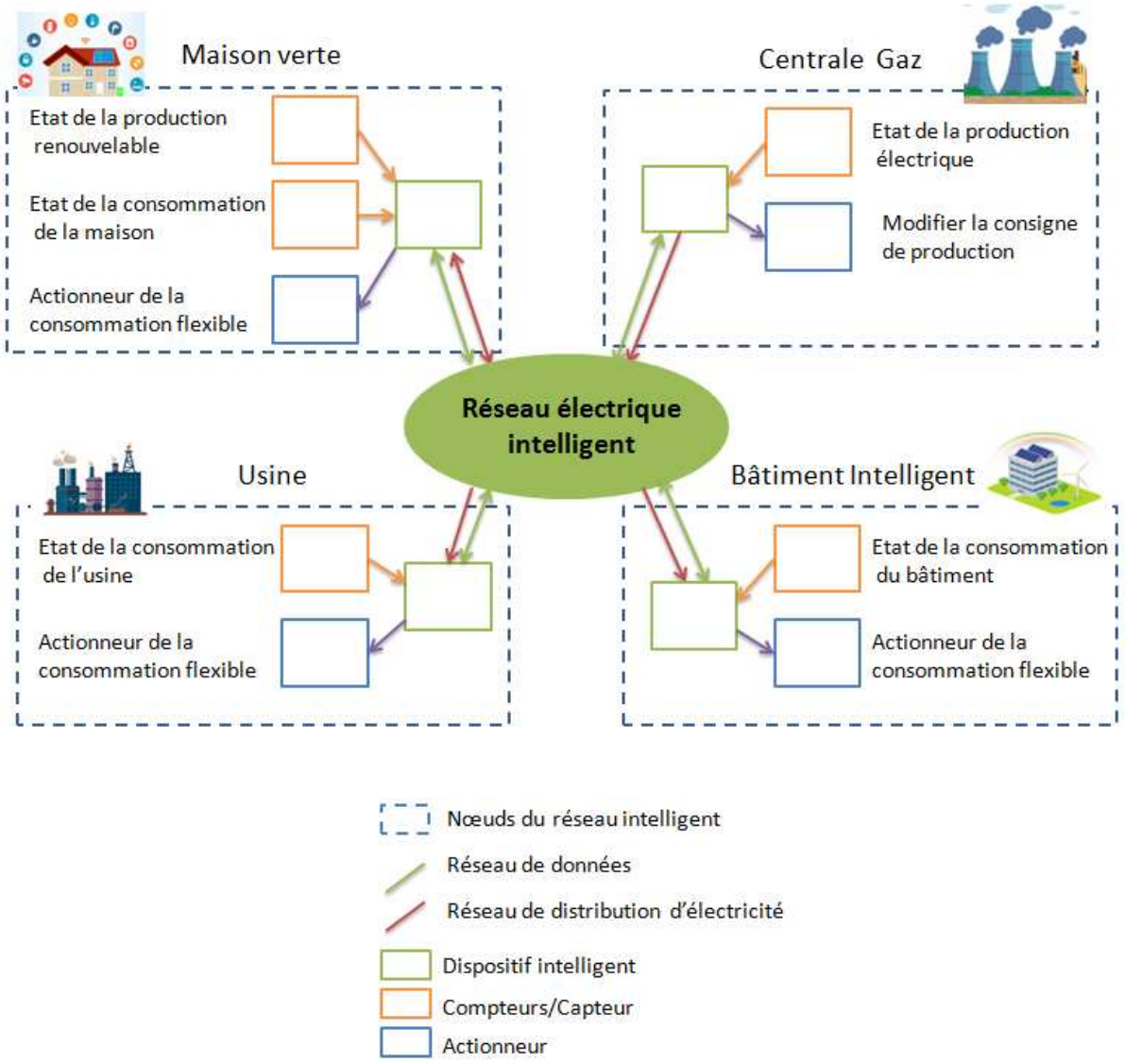


FIGURE 2.16 – Exemple de composition d'un Smart Grid

2.6.3 Présentation des systèmes AMI (Advanced Metering Infrastructure)

Pour mettre en place les Smart Grids, il existe des solutions technologiques adaptées. On trouve principalement les systèmes AMI (Advanced Metering Infrastructure) qui sont une solution composée de plusieurs modules ayant pour but d'aider les compagnies d'électricité à profiter des nouvelles technologies pour la gestion du réseau électrique [BCFD16]. Ainsi, les AMI permettent d'introduire des compteurs intelligents chez les clients finaux, de détecter les pertes d'énergies au cours de la distribution ainsi qu'un échange interactif avec les clients. Comme résultats, la qualité des services en relations avec l'énergie se verront s'améliorer et des économies peuvent être faites suite à la réduction de pertes d'énergie ce qui va faire également diminuer les emmissions du carbone.

Il existe plusieurs fabriquants qui proposent des systèmes AMI. Nous citons principalement HUAWEI [AMI] et INHEMETER. Ces deux compagnies ont commencé à mettre en place leurs systèmes AMI dans plusieurs pays.

Un système AMI se compose principalement des modules suivants (voir figure 2.17) :

1) Dispositifs

- Appareils électroniques équipés de consomètres ou de prises intelligentes : permettant ainsi de contrôler la consommation de chaque appareil électronique et de la transmettre au compteur intelligent.

Le consomètre est composé d'une prise électrique et d'un écran d'affichage permettant de capturer la consommation d'un appareil électrique. Il peut être connecté au réseau local.

La prise connectée intelligente peut être branchée à un appareil électrique et peut être contrôlée par une application Android/iOS. Cette application permet d'afficher la consommation de l'appareil électrique, d'arrêter, de démarrer ou de programmer le fonctionnement d'un appareil. La prise connectée ne possède pas d'écran d'affichage.

- Compteur intelligent : permet de remplacer le compteur analogique non connecté. Il est connecté à tous les appareils électroniques du clients. Ce compteur envoie directement l'état d'utilisation de l'électricité d'un client au fournisseur. Il n'y a plus besoin de factures estimées ou bien de lecture manuelle du compteur. Il existe principalement deux types de compteurs intelligents : Les compteurs monophasés déstinés à l'utilisation résidentielle à faible consommation, et les compteurs triphasés déstinés aux clients industriels et

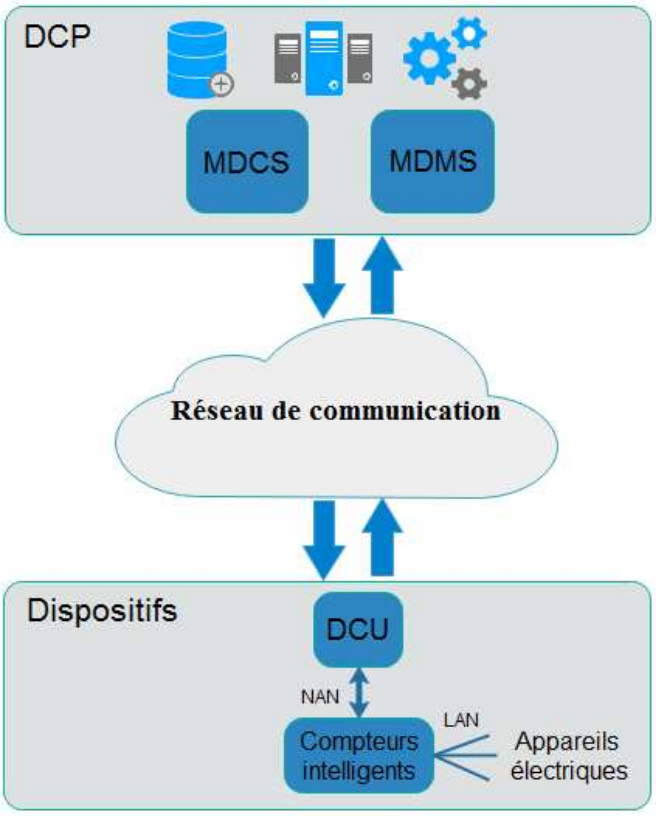


FIGURE 2.17 – Description des systèmes AMI (Advanced Metering Infrastructure)

commerciaux à grande consommation.

- DCU (Data Concentrator Unit) : Il s'agit d'une unité responsable de la collecte des données relatives à la consommation d'énergie envoyées par les compteurs intelligents. Cette unité est équipée de puces implémentant des protocoles de communication afin de pouvoir relier les compteurs intelligents au centre de données de la compagnie d'énergie.

La communication entre le compteur intelligent et les appareils électroniques se fait au sein d'un réseau à faible portée qu'on appelle LAN (Local Area Network). La communication entre les DCU et les compteurs intelligents se fait au sein d'un réseau NAN (Neighborhood Area Network) qui a une portée plus grande que le LAN.

2) Réseau de communication : Les dispositifs utilisés doivent supporter des protocles de communication adéquats afin d'assurer la communication entre les DCUs et la plateforme DCP centralisée qui assure la collecte de données.

3) DCP (Data Collection Platform) : Cette platforme est responsable de la collecte et de

l'analyse des données envoyées par les DCUs et les compteurs intelligents. La plateforme DCP est composée de deux systèmes :

- MDCS (Meter Data Collection System) : Ce système est responsable d'interagir avec les dispositifs en gérant les protocoles de communication et de collecter et stocker les données relatives à la consommation électrique.

- MDMC (Meter Data Management System) : Ce système est responsable de l'analyse des données relatives à la consommation électrique, d'effectuer des statistiques, de générer des rapports,etc.

2.6.4 Application de notre approche au Smart Grids

Plusieurs fabriquants s'intéressent à l'implémentation des Smart Grids en propososant des solutions technologiques adaptées. Le système AMI est déstiné à être intégré dans le fonctionnement global de tout noeud relié au Smart Grid afin d'avoir plus de visibilité et de contrôle sur la consommation et la production de l'énergie par les compagnies de gestion de l'électricité. Ce type de système peut être considéré comme un système IoT vu qu'il comporte tous les éléments spécifiques à ce type de systèmes. Cette contextualisation des Smarts Grids dans le domaine de l'IoT nous permet de relever plusieurs défis en relation avec la mise en place des Smarts Grids dans un contexte réel. Ces défis concernent tout d'abord la proposition d'une architecture modèle pour pouvoir modéliser les différents éléments impliqués dans ce système. Une telle architecture doit pouvoir supporter l'aspect dynamique des Smart Grids ainsi que le besoin de fiabilité et d'extensibilité.

Dans la suite de ce rapport, nous allons utiliser les Smart Grids comme cas d'étude de notre proposition. Nous allons présenter les Smart Grids selon l'architecture IoT de bout en bout. Ensuite, nous allons modéliser les Smart Grids selon l'architecture ReDy proposée. Enfin, nous allons modéliser formellement une configuration simplifiée appelée micro Smart Grid en utilisant le langage de modélisation formelle LNT et les outils de la boite à outils CADP afin de prouver les critères recherchés de fiabilité, de dynamisme et d'extensibilité.

2.7 Conclusion

Dans ce chapitre, nous avons étudié les concepts et approches nécessaires à la bonne initiation dans notre thématique de recherche. En effet, nous avons commencé par étudier plusieurs aspects intéressants concernant les systèmes IoT, principalement les architectures IoT proposées dans la littérature ainsi que les principaux défis et axes de recherches les concernant. Nous avons ensuite présenté la problématique de notre travail de recherche qui consiste à étudier les systèmes IoT implémentés dans un environnement dynamique tout en assurant leurs fiabilité et extensibilité. Afin de répondre à l'ensemble des défis relevés dans la problématique, nous avons choisi les méthodes formelles dans notre approche pour prouver les critères recherchés. Nous avons présenté dans ce chapitre également les travaux existants dans la littérature qui portent sur la modélisation et l'analyse formelle des architectures IoT en utilisant plusieurs outils et approches des méthodes formelles. Il s'agit d'un axe de recherche très prometteux et très récent au sein de la communauté de recherche. Nous avons également présenté les concepts et les outils que nous utilisons pour mener notre travail de validation formelle afin d'assurer les critères définis dans la problématique, à savoir le langage LNT pour exprimer le modèle formel, le langage MCL pour exprimer les propriétés de logique temporelle et la boite à outils CADP pour exploiter le modèle formel. Enfin, nous avons donné une présentation générale des Smart Grids représentant le cas d'étude de notre approche.

Chapitre 3

Architecture de type ReDy déstinée aux systèmes IoT

3.1 Introduction

Dans ce chapitre, nous commençons par présenter l'architecture IoT de bout en bout (End To End-E2E) que nous avons pu déduire de l'analyse des différentes architectures étudiées dans l'état de l'art. Nous allons ensuite montrer un cas pratique de déploiement de cette architecture dans le cas des réseaux électriques intelligents (Smart Grids). En étudiant ce cas pratique, nous allons déduire l'architecture ReDy qui est le coeur de contribution dans ce chapitre. Nous allons ensuite expliciter les principales caractéristiques et fonctionnalités assurées par notre architecture ReDy. En effet, l'architecture ReDy proposée est destinée aux applications IoT où les besoins en fiabilité, en dynamisme et en extensibilité doivent être satisfaits. En dernier lieu, nous allons résumer l'approche que nous avons suivie en présentant un processus expliquant les différentes phases exécutées.

3.2 Architecture générale de bout en bout (E2E) de l'IoT

3.2.1 Présentation de l'architecture générale de bout en bout

D'après les architectures de bout en bout étudiées dans le chapitre de l'état de l'art, nous pouvons déduire une architecture générale selon laquelle les architectures IoT de bout en bout sont conçues (figure 3.1). Cette architecture générale est composée de cinq

couches : 1) Dispositifs 2) Réseau 3) Middlware 4) Application 5) Business. Dans une architecture de bout en bout, on peut regrouper plusieurs couches, ou bien diviser une couche en plusieurs sous couches selon les objectifs et les exigences relatives au système IoT en question. Dans la suite nous allons donner une description de chaque couche de cette architecture générale.

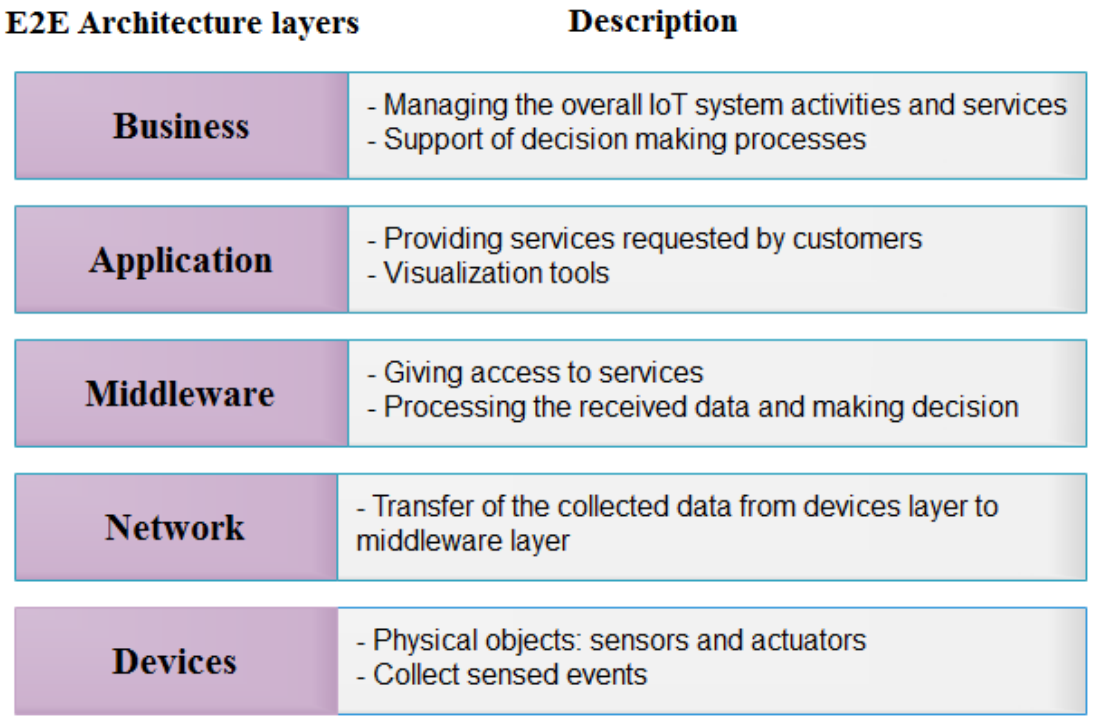


FIGURE 3.1 – Architecture IoT multi-couches générale de bout en bout

- Couche dispositifs : appelée également couche de perception. Cette couche contient les appareils en contact avec le monde physique pouvant être des détecteurs ou des actionneurs. Cette couche est responsable de regrouper les données du monde physique et de les transmettre à travers la couche réseau. La nature de ces appareils diffère selon le domaine d'application et la nature des données à collecter : humidité, luminosité, mouvement, chaleur, etc.

- Couche réseau : cette couche est responsable de transmettre les données collectées par la couche de perception à la couche middlware. Il faut garantir des canaux de communication fiables pour la transmission des données. Il existe plusieurs technologies qui peuvent être utilisées dans cette couche : Wifi, Zigbee, 3G/4G/5G, RFID, Bluetooth, etc.

- Couche Middlware : cette couche est responsable du stockage et de l'interprétation des

données collectées afin d'acheminer le service demandé au bon endroit. Cette couche permet principalement de supporter l'hétérogénéité des appareils et des applications et assure ainsi le lien entre le monde physique et la couche application.

- Couche application : cette couche fournit les services demandés à l'utilisateur final à travers des outils de visualisation. Les services fournis diffèrent largement selon le domaine d'application : maison intelligente, usine intelligente, véhicule intelligent, soin de la santé, etc.

- Couche business : cette couche gère les activités et les services du système IoT. Elle est responsable de faire l'étude des données reçues de la couche application et de les structurer et de les analyser afin d'organiser les résultats en tableau de bord constitué de diagrammes, tableaux, processus, etc, afin d'avoir une évaluation globale des solutions déployées.

3.2.2 Défis des systèmes IoT selon l'architecture IoT E2E

La conception et la mise en place des sytèmes IoT présentent plusieurs défis comme nous avons vu dans l'Etat de l'Art 2.3. Ces défis peuvent être répartis en couches selon l'architecture générale de bout en bout. Dans la Figure 3.2, nous présentons les défis relatifs à chaque couche. Ceci dit, il existe plusieurs défis transversaux qui figurent sur plusieurs couches mais leurs interprétations diffèrent à chaque couche.

Dans la suite, nous allons présenter les Smart Grids comme cas d'étude en présentant son organisation générale ainsi que les principaux défis rencontrés dans ce type de systèmes.

3.3 Modélisation des Smart Grids selon l'architecture IoT de bout en bout

Après avoir défini le contexte général des Smart Grids ainsi que les principaux éléments le composant dans le chapitre précédent, nous allons établir à présent l'organisation des Smart Grids selon l'architecture IoT de bout en bout.

En effet, les systèmes AMI implémentés dans les Smart Grids possèdent plusieurs composants ayant pour but d'avoir plus de visibilité et de contrôle sur la consommation et la production de l'énergie par les compagnies de gestion de l'électricité. Ce type de

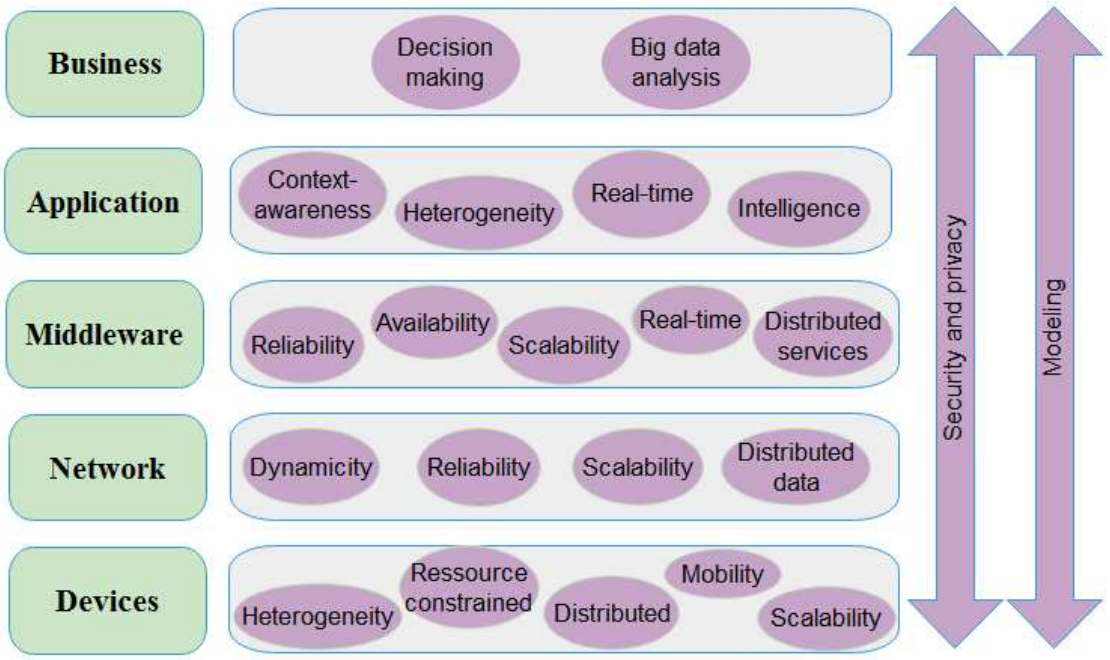


FIGURE 3.2 – Défis des systèmes IoT répartis par couches de l'architecture IoT E2E

système peut être considéré comme un système IoT vu qu'il comporte tous les éléments spécifiques à ce type de système. La figure 3.3 illustre l'organisation du système AMI selon l'architecture IoT de bout en bout [HKK18].

D'après l'organisation du Smart Grid selon l'architecture IoT de bout en bout, nous pouvons déduire une organisation hiérarchique des différents éléments du Smart Grid. Cette organisation hiérarchique peut être présentée comme suit :

- Chaque groupe d'appareils électroniques est relié à un compteur intelligent (via des consomètres ou des prises intelligentes).

- Plusieurs compteurs intelligents sont reliés à une DCU.

- Plusieurs DCUs sont reliées à la DCP.

- Les DCPs peuvent être reliées à des unités supérieures pour offrir des services de niveau application ou business.

Dans notre exemple, nous nous contentons de trois niveaux d'hiérarchisation. La figure 3.4 illustre cette organisation hiérarchique selon trois niveaux. Les noeuds appartenant à chaque niveau sont de même type. Les noeuds appartenent à deux niveaux différents sont

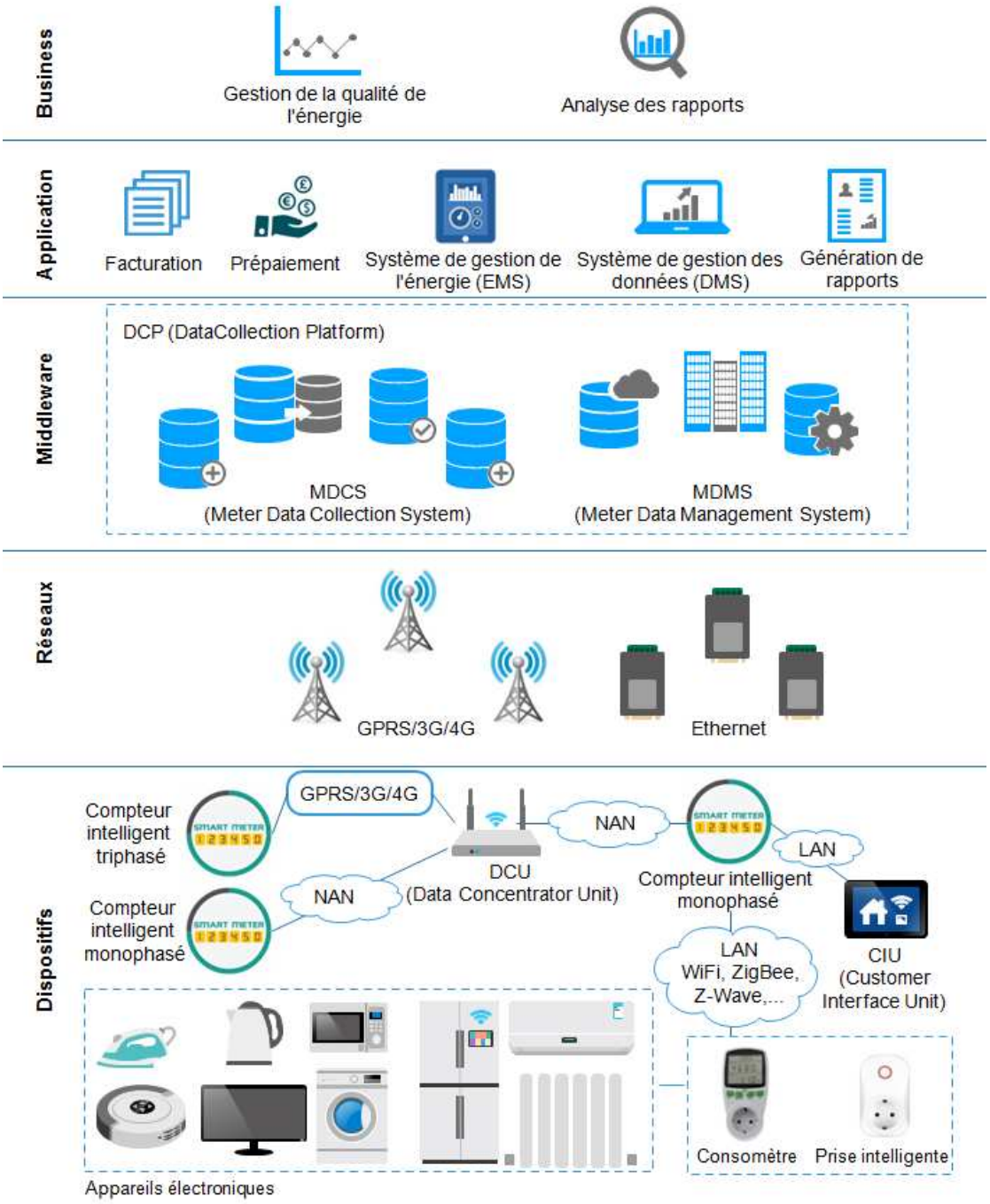


FIGURE 3.3 – Smart Grid selon l'architecture IoT de bout en bout

de types différents.

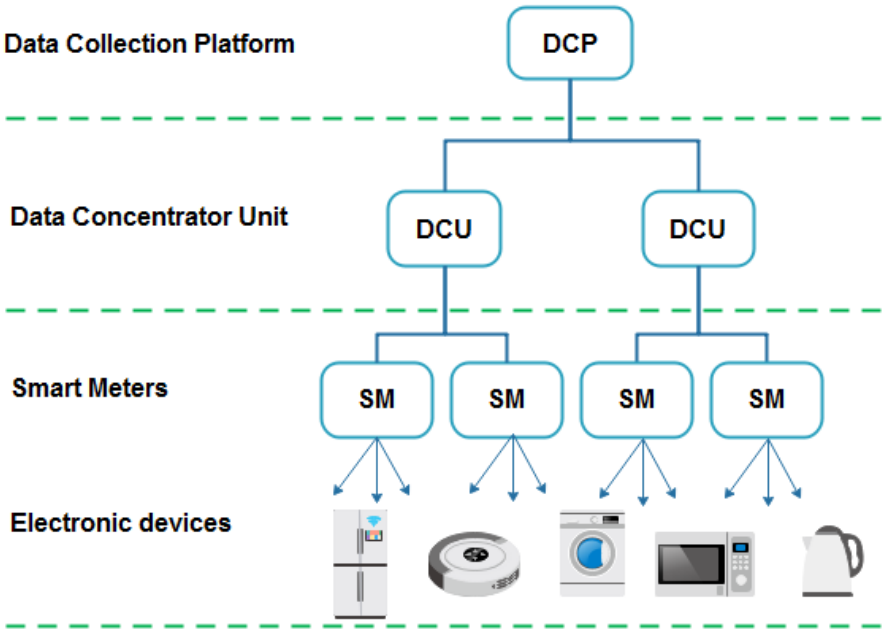


FIGURE 3.4 – Composants du Smart Grid organisés par niveau

A partir des éléments étudiés dans les Smart Grids, nous allons pouvoir présenter notre posposition de l'architecture ReDy applicable pour une grande famille de systèmes IoT.

3.4 Spécification d'un système IoT de type ReDy

Plusieurs systèmes IoT (les Smart Grids comme exemple) suivent une organisation hiérarchique et possèdent un ensemble de critères à vérifier afin de répondre aux besoins spécifiques. Le type de systèmes IoT que nous addressons doivent garantir la fiabilité et se caractérisent par leur aspect dynamique. Pour répondre à ce besoin, nous proposons notre architecture dite ReDy (Reliable and Dynamique) qui vient pour renforcer la conception des systèmes IoT fiables et dynamiques, comme est le cas pour les Smart Grids. Les critères de distributivité et d'extensibilité s'ajoutent également à l'ensemble des critères à satisfaire pour ce type de systèmes IoT.

Ainsi, notre objectif est de concevoir un modèle d'une architecture qui exprime les principales fonctionnalités relevées par les nouvelles applications des systèmes IoT. Ce modèle possède un ensemble de propriétés et répond à un ensemble d'exigences que l'on retrouve dans une large famille d'applications des systèmes IoT. Notre but est de

simplifier la conception des systèmes IoT en combinant plusieurs choix faits durant les différentes phases du processus de conception afin d'obtenir un niveau satisfaisant de fiabilité.

Dans cette section, nous commençons par présenter les exigences générales des systèmes IoT concernés par notre proposition. En second lieu, nous proposons une architecture du système montrant comment les composants du système sont organisés. Enfin, nous présentons les fonctionnalités du système qui doivent être implémentées par l'architecture logique afin de satisfaire les exigences générales du système.

3.4.1 Exigences générales des systèmes ReDy

Dans ce qui suit, nous donnons une caractérisation de la famille des systèmes IoT de type ReDy. Cette famille de systèmes est définie en garantissant les exigences suivantes :

- La fiabilité du système. En effet, le système doit avoir une grande résistance aux défaillances. Le fonctionnement général du système ne doit pas être affecté par un nombre donné de déffaillances touchant les composants du système. La fiabilité consiste à assurer la disponibilité du système dans le temps. Par conséquent, la fiabilité du système implique sa disponibilité.

- La gestion dynamique de l'adhèsion des composants au système. Cet aspect permet aux composants de rejoindre ou de quitter le système sans aucune détérioration de la performance générale du système.

- Le système doit être en contact avec l'environnement extérieur afin de pouvoir capter tout événement ou information signifiante. Le type de cette information varie largement selon la nature et les objectifs tracés du système à concevoir. Par exemple, on peut chercher à capturer un seuil de température dans une pièce, ou bien une certaine vibration lors d'un tremblement de terre.

- Le système est étalé géographiquement sur une surface donnée et il est responsable de couvrir cette zone afin d'accomplir les objectifs attendus. Une fois l'information ou l'événement est capturé, il existe un mécanisme dans le système permettant de traiter et d'analyser cette information afin de pouvoir prendre une décision adéquate. La prise de décision est suivie par l'exécution d'une action permettant ainsi d'assurer le bon fonctionnement général du système.

3.4.2 Architecture de type ReDy

Afin de satisfaire les exigences citées en haut, nous proposons de concevoir une certaine famille de systèmes distribués selon l'architecture de type ReDy destinée aux applications IoT. Notre architecture ReDy stipule que le système global est composé de plusieurs sous-systèmes et que chaque sous-système est lui même un sytème distribué constitué de plusieurs composants. Il existe trois types de composants dans notre système : les unités de détection, les unités d'action et les unités de gouvernance. Dans la suite, nous détaillons les caractéristiques de chaque type de composant.

1) Unités de détection : ce sont des composants en contact avec l'environnement externe et qui sont responsables de détecter les changements affectant l'environnement extérieur. En général, ces unités représentent les capteurs. La nature du capteur utilisé dépend du type de l'événement qui doit être détecté.

2) Unités d'action : ce sont des composants en contact avec l'environnement externe et qui sont responsables d'exécuter des actions affectant l'environnement externe. En général, ces unités représentent les actionneurs. Le type de l'actionneur dépend du type de l'action qui doit être menée.

3) Unités de gouvernance : ce sont des composants chargés de collecter les informations de la part des unités de détection et de prendre des décisions adéquates après l'analyse des informations reçues. Le processus d'analyse et de traitement des données recueillies dépend des objectifs et du domaine d'application du système distribué en question. La décision prise est alors envoyée aux unités d'actions. Il faut noter que chaque sous-système possède une unique unité de gouvernance et que les unités de gouvernances des différents sous-systèmes peuvent communiquer entre elles.

Dans l'architecture ReDy, un sous-système peut être :

- Un sous-système unitaire qui est composé d'une unité de gouvernance et de plusieurs unités de détection et d'action.

- Composé de plusieurs sous-systèmes unitaires et/ou non unitaires

Un sous-système est toujours muni d'une unité de gouvernance. Cette unité de gouvernance est dite :

- Unité de gouvernance unitaire (Unit Governance Unit-UGU) : Assure la gestion des

unités de détection et d'action.

- Unité de gouvernance de gestion (Management Governance Unit-MGU) : Assure la gestion d'autres unités de gouvernance (de type UGU et MGU).

Les unités de gouvernance sont réparties selon plusieurs niveaux. L'unité de gouvernance réalise trois tâches :

- Gérer la communication avec le niveau supérieur.

- Gérer la communication avec les autres unités de gouvernances du même sous-système et du même niveau dans une communication P2P.

- Gérer la communication avec le niveau inférieur.

Dans le cas d'une unité de gouvernance unitaire (UGU), la communication avec le niveau inférieur consiste à gérer les unités de détection et d'action dans une communication client/serveur ou la UGU joue le rôle du serveur et les DUs et les AUs jouent le rôle du client.

Exemple : La figure 3.5 montre un exemple de l'architecture globale d'un système de type ReDy. Dans cet exemple, le système global est organisé selon quatre niveaux et il est composé de deux sous-systèmes de niveau 4.

- Le premier sous-système de niveau 4 (subsystem1) est compososé de deux sous-systèmes de niveau 3 :

– USS11 : sous-système unitaire de niveau 3 composé d'une unité de gouvernance unitaire et de deux unités d'action et d'une unité de détection.

– Subsystem12 : sous-système de niveau 3 composé de deux sous-systèmes de niveau 2 :

— USS121 : sous-système unitaire de niveau 2 composé d'une unité de gouvernance unitaire et d'une unité d'action et d'une unité de détection.

— Subsystem122 : sous-système de niveau 2 composé de deux systèmes unitaires de niveaux 1 (USS1221 et USS1222). Chaque sous-système est composé d'une unité de gouvernance, une unité de détection et une unité d'action.

- Le deuxième sous-système de niveau 4 (subsystem2) est composé de trois sous-systèmes unitaires de niveau 3 : USS21, USS22, USS23. Chaque sous-système unitaire est composé

d'une unité de gouvernance unitaire et d'unités de détection et d'action.

3.4.3 Etude fonctionnelle

Les fonctionnalités requises d'un système distribué ReDy hiérarchique sont divisées en trois catégories selon la nature des unités communicantes et le niveau hiérarchique auquel elles appartiennent. Ces trois catégories sont les suivantes : Mode local, Mode distant et Mode inter-niveaux. Dans la suite, nous décrivons chaque mode en détails (voir figure 3.6).

Mode local

Le mode local concerne les échanges faits entre une unité de gouvernance (unitaire) et les unités de détection et d'action, c'est à dire les échanges faits au sein d'un même sous sytème unitaire. Les fonctionnalités les plus importantes qui doivent être satisfaites dans ce mode sont comme suit :

1. Détection d'un événement provenant de l'environnement extérieur. Cette fonctionnalité doit être assurée par l'unité de détection.
2. Envoi de l'information détectée à l'unité de gouvernance du même sous-système.
3. Prise de la décision adéquate par l'unité de gouvernance.
4. Envoi des ordres d'exécution de la part de l'unité de gouvernance aux unités d'action concernées. L'unité de gouvernance et les unités d'action concernées appartiennent au même sous-système.
5. Exécution des décisions par les unités d'action.

Le mode local d'un sous-système ne doit pas dépendre des défaillances survenant en dehors de ce sous-système, même en cas d'interruption de connection avec les autres sous-systèmes. Le fonctionnement en mode local est assuré tant qu'il existe au moins une unité de détection et une unité d'action connectées à l'unité de gouvernance du sous-système.

Mode distant

La fonctionnalité principale à remplir dans le mode distant est la communication entre les unités de gouvernance des différents sous-systèmes appartenant au même niveau et

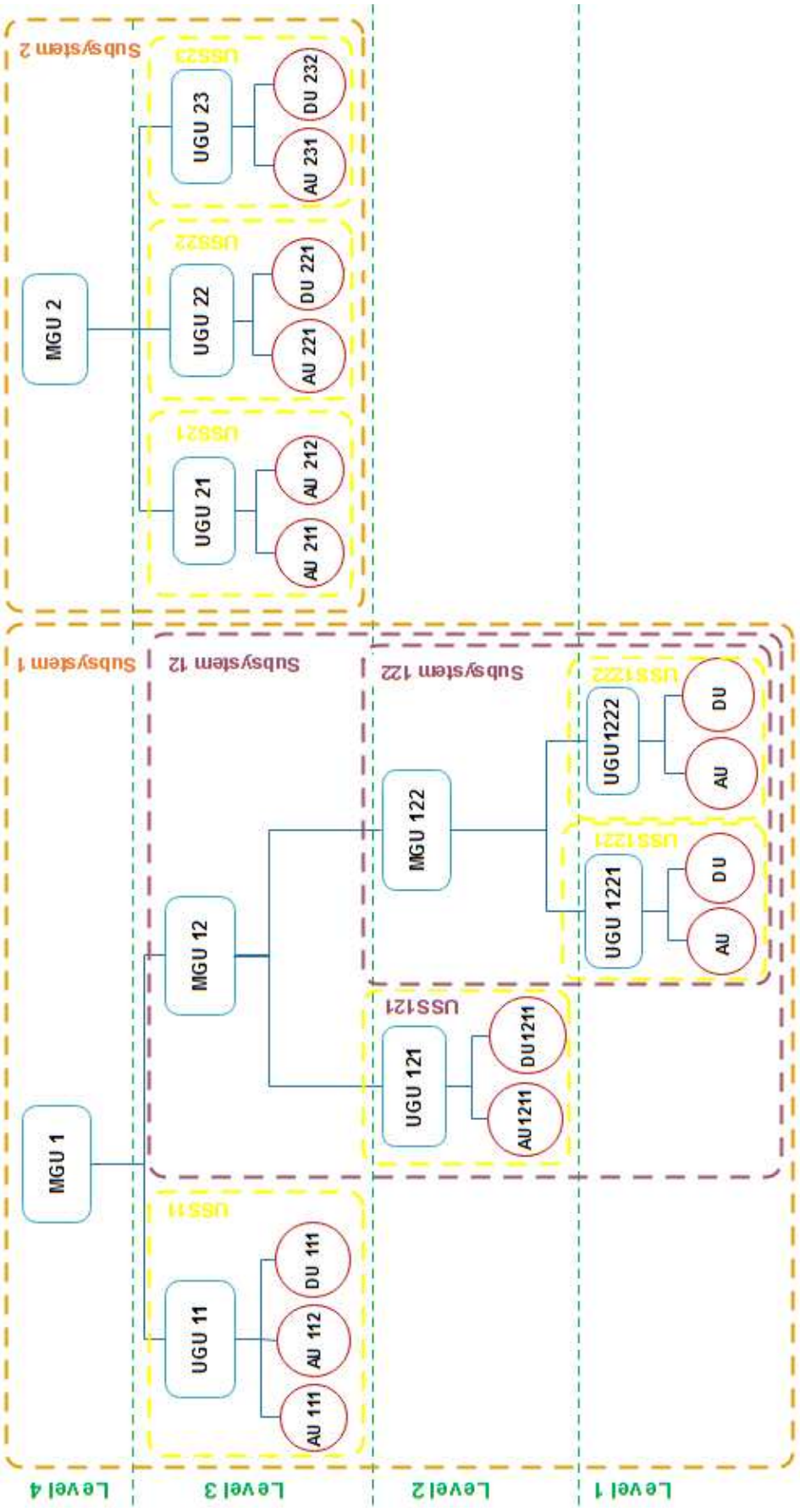


FIGURE 3.5 – Architecture ReDy en couches

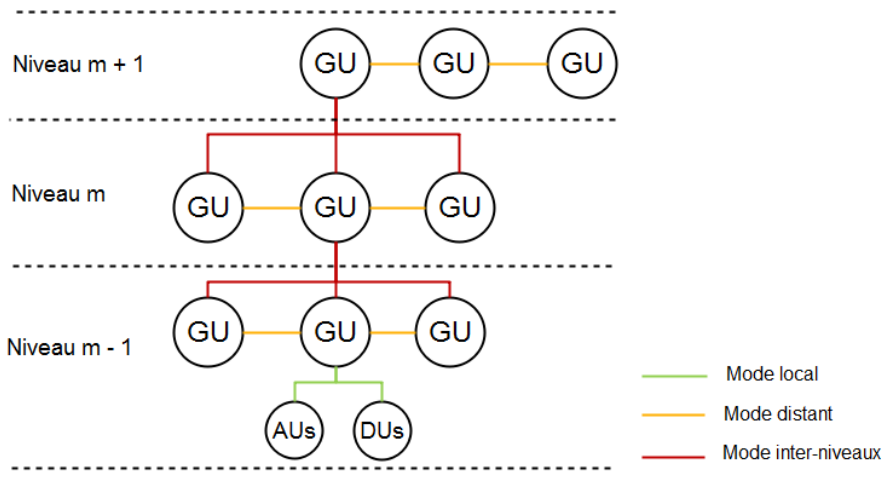


FIGURE 3.6 – Modes de communication dans l'architecture ReDy

reliées à la même unité de gouvernance du niveau supérieur. Chaque unité de gouvernance informe les autres unités de gouvernance de toute détection ou action effectuée au niveau de son sous-système spécifique. Ainsi, chaque unité de gouvernance possède une connaissance de l'état actuel de toute la zone couverte par le système distribué global.

Il existe un autre cas où l'on a besoin de la communication en mode distant. Une fois l'information est reçue de la part de l'unité de détection, l'unité de gouvernance peut décider que l'action à faire doit être exécutée par des unités d'action faisant partie d'un autre sous-système que le sien. Dans ce cas, une communication est établie entre deux unités de gouvernance appartenant à deux sous-systèmes différents afin de faire parvenir la requête à la bonne destination.

Dans le mode distant, nous devons assurer une communication fiable de bout en bout entre deux unités de gouvernance. Tant que l'expéditeur et le récepteur se comportent correctement, la communication doit être assurée même en cas de défaillance d'autres unités de gouvernance dans le système global.

Mode inter-niveau

Le mode inter-niveaux concerne les échanges faits entre une unité de gouvernance de niveau n avec toute unité de gouvernance de niveau supérieur $n+1$, ou de niveau inférieur $n-1$. Les fonctionnalités principales à satisfaire par une unité de gouvernance dans ce mode sont comme suit :

- Maintenir un lien avec le niveau supérieur (à travers des échanges périodiques avec une unité de gouvernance de niveau supérieur).

- Maintenir une connaissance des unités de gouvernance qui lui sont directement attachées et qui appartiennent au niveau inférieur.

3.4.4 Représentation simplifiée de l'architecture ReDy

Globalement, nous avons deux types de communication : le mode local revient à une communication entre une unité de gouvernance et une unité d'action ou de détection. Le modes distant et le mode inter-niveau reviennent tous les deux à une communication entre des unités de gouvernance. Une représentation simplifiée de l'architecture ReDy serait équivalente à un système composé de plusieurs sous-systèmes tels que chaque sous-système est gouverné par son unité de gouvernance qui gère les deux types de communication.

Exemple : La figure 3.7 montre un exemple de l'architecture globale d'un système IoT de type ReDy. Dans cet exemple, le système global est composé de trois sous-systèmes. Chaque sous-système comporte une seule unité de gouvernance. Le premier sous-système est composé de trois unités de détection et de deux unités d'action. Le deuxième sous-système est composé d'une unité de détection et de trois unités d'action. Le troisième sous-système est composé de deux unités de détection et de deux unités d'action. Cet exemple représente un modèle simplifié à but illustratif. Les unités de gouvernance sont connectées entre elles, alors que les unités de détection et d'action de chaque sous-système sont reliées uniquement à l'unité de gouvernance du même sous-système.

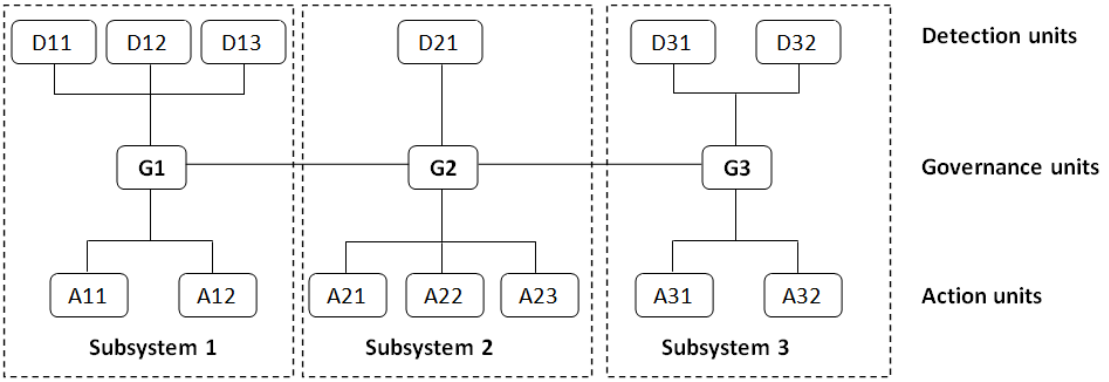


FIGURE 3.7 – Exemple d'une représentation simplifiée de l'architecture ReDy

3.5 Conception des systèmes ReDy

Dans la phase de conception de nos systèmes ReDy, nous avons opté pour une architecture hybride qui combine des solutions centralisées, c'est à dire des solutions clients-serveur pour la communication entre les composants appartenant au même sous-système (mode local), et des solutions décentralisées pour la communication entre les différentes unités de gouvernance (mode distant). Notre système ReDy est divisé en plusieurs sous-systèmes. Dans un sous-système, l'unité de gouvernance représente le serveur et les unités de détection et d'action représentent les clients : c'est le mode centralisé. Les unités de gouvernance communiquent entre elles afin d'échanger des services. Ainsi, les unités de gouvernance se comportent à la fois comme client et serveur : c'est le mode décentralisé.

Dans la suite, nous allons nous focaliser sur l'étude de la manière avec laquelle le réseau des unités de gouvernance est construit ainsi que le protocole de communication établi entre les unités de gouvernance. Ceci dit, nous allons nous concentrer sur l'étude de la partie décentralisée de notre système.

3.5.1 Algorithme de brassage pour la gestion de l'adhésion des composants au système

Il existe plusieurs modes permettant d'organiser les composants d'un système distribué. Dans la suite, nous discuttons l'organisation des unités de gouvernance selon une architecture distribuée, tout en prenant en considération les exigences générales de la conception du système. Il est à noter que tous les composants communiquants dans les abstractions à venir représentent des unités de gouvernance.

Dans notre système, toute information doit être propagée sur toutes les unités de gouvernance. Chaque unité de gouvernance est responsable d'une zone donnée et possède les informations concernant l'état de cette zone. L'unité de gouvernance est aussi responsable de la transmission de l'état actuel de la zone qui est sous sa responsabilité à toutes les autres unités de gouvernance du système, et reçoit de la part des autres unités de gouvernance les états actuels de leurs zones de responsabilité. Par conséquent, chaque unité de gouvernance possède une connaissance de l'état de toute la zone couverte par le système global.

Afin d'assurer ces caractéristiques, la partie décentralisée de notre système distribué est construite suivant *une architecture peer-to-peer non structurée* [LCP+05]. Dans la

suite, les unités de gouvernance représentent les noeuds de l'architecture peer-to-peer non structurée. La raison principale qui motive notre choix pour ce type d'architecture réside dans le fait que c'est l'architecture la plus adaptée pour les environnements très dynamiques, c'est à dire que la performance du système n'est pas déteriorée par des noeuds rejoignant ou quittant le système [LCP+05]. Comme résultat, cette architecture est adaptée pour les systèmes sujets à de grands risques de défaillances. En plus de cela, la communication dans de telles architectures est effectuée par broadcast épidémique, ce qui correspond à nos exigences, puisque la communication entre les unités de gouvernance se fait par diffusion de l'information à toutes les unités du système.

La construction du graphe : Les systèmes peer-to-peer non structurés sont construits à base d'algorithmes randomisés (randomized). L'idée principale de ce type de systèmes est que chaque noeud possède une liste de voisins construite de manière plus ou moins randomisée [TVS07]. On appelle cette liste de voisins *vue partielle*. Il existe plusieurs méthodes pour construire cette vue partielle. Dans notre travail nous utilisons *l'algorithme de brassage renforcé* (enhanced shuffling algorithm) proposé par Voulgaris et al [VGVS05].

L'utilisation de l'algorithme de brassage est très utile dans les milieux dynamiques et qui représentent un grand risque de défaillance pour les noeuds du système.

3.5.2 Modèle de communication

Dans cette partie, nous discuttons le modèle de communication choisi pour notre système. La communication au sein de notre système s'effectue par diffusion de l'information à travers tous les noeuds communiquants du système. Cela signifie que le modèle de communication le plus adapté dans notre situation est le broadcast.

Il existe plusieurs variantes du broadcast [CGR11], dépendemment des hypothèses faites à propos de plusieurs composants du système. Nous allons étudier deux abstractions importantes permettant d'avoir un modèle de système distribué précis en les combinant. Ces deux abstractions concernent la nature des processus et la nature des liaisons (canal de communication). Dans la suite, nous commençons par une explication de ces deux abstractions en argumentant les hypothèses faites dans notre cas. Ensuite, nous abordons la variante du broadcast adaptée pour les abstractions supposées.

Notre système doit avoir un grand degré de fiabilité. Ceci est réalisé en assurant et en gérant la tolérance aux fautes afin que le fonctionnement général du système ne

soit pas affecté. La sélection de la nature des processus ainsi que la nature du canal de communication est d'une grande importance afin de satisfaire notre objectif. Pratiquement parlant, on doit assurer qu'un processus qui tombe en panne et se rétablit par la suite peut continuer à participer dans le système. Il s'agit de l'abstraction faite à propos du processus que l'on appelle : *l'abstraction du processus de type crash-recovery.* L'abstraction concernant le canal de communication définit la liaison point-à-point (point-to-point link). Le choix de cette liaison doit être en harmonie avec le choix de l'abstraction des processus et doit assurer que même en cas de défaillance du processus, le canal sera capable de délivrer les messages envoyés à travers le réseau après récupération du processus. Cette caractéristique est assurée en utilisant *l'abstraction du canal de type têtu (stubborn).* Expliquons en détails ces deux abstractions.

l'abstraction du processus de type crash-recovery : dans cette abstraction, un processus peut tomber en panne, mais il peut se remettre en marche par la suite. Dans ce cas, on dit qu'un processus est *défectueux* s'il tombe en panne et ne se remet jamais en marche, ou bien s'il continue à tomber en panne et se remettre en marche de façon infine. Sinon, le processus est dit *correct*, c'est à dire qu' un processus qui tombe en panne et se remet en marche un nombre fini de fois est considéré comme étant correct dans notre modèle.

Il existe un problème majeur que l'on rencontre lorsqu'on utilise l'abstraction de type crash-recovery qui est le problème d'amnésie du processus, c'est à dire que le process peut oublier les actions qu'il a faites avant de tomber en panne. Ce problème est résolu en utilisant l'abstraction du canal de type stubborn [GOS98].

l'abstraction du canal de type têtu (stubborn) : cette abstraction se base sur un mécanisme de retransmission réalisée par le processus expéditeur. Cette retransmission assure que même en cas de défaillance du process, ce dernier va pouvoir délivrer le message qui lui a été envoyé quand il sera de nouveau en marche, puisque l'expéditeur continue de retransmettre le message jusqu'à réception d'un accusé de réception. Comme conséquence, cette abstraction peut engendrer des réceptions multiples par le processus récepteur.

Afin d'éliminer le problème de réceptions multiples, on définit une variable dans un espace de stockage stable où l'on stocke tout message reçu par le processus lorsqu'il est en marche. Lorsque le processus tombe en panne et se remet en marche, la première opération qu'il exécute c'est de récupérer l'information à partir de l'espace de stockage stable afin de retrouver l'état où il était avant de tomber en panne. Comme résultat, le problème de réceptions multiples est résolu.

Une fois nous avons établi les abstractions pour notre système distribué, nous abordons l'algorithme utilisé pour la communication entre les composants du système. Cet algorithme prend en considération les abstractions faites auparavant, et assure les objectifs fixés pour notre système. L'algorithme utilisé implémente *un broadcast fiable et uniforme* [CGR11] et vérifie les propriétés suivantes :

— Validité : Si un processus qui n'est jamais tombé en panne fait le broadcast d'un message m, alors tout processus correct va finir par délivrer (réception avec accusé de réception) le message m.
— Pas de duplication : Il n'y a pas de message délivré plus qu'une fois.
— Pas de création : Si un processus délivre un message m venant de l'expéditeur s, alors m a été diffusé (broadcast) auparavant par le processus s.
— L'accord uniforme : Si un message m est délivré par un processus quelconque (qu'il soit correct ou défectueux), alors m est sûrement délivré par tout process correct.

L'algorithme de broadcast fiable et uniforme est conçu à base de quatre principaux événements : l'événement d'initialisation, l'événement de récupération, l'événement de broadcast et l'événement de livraison (delivery). Dans la suite, nous expliquons ces événements en détails.

1) Événement d'initialisation : cette opération est réalisée lorsque le processus rejoint le système pour la première fois (figure 3.8). Il existe trois variables importantes qui sont initialisées à des ensembles vides dans l'événement d'initialisation et qui sont manipulées dans les autres événements. La première variable est *delivered* qui représente l'ensemble des messages délivrés par le processus. La deuxième variable est *pending* qui représente l'ensemble des messages qui sont déja vus et reçus par le processus, mais qui ne sont pas encore délivrés. La troisième variable est $ack[m]$ qui est un tableau contenant l'ensemble des processus que le processus actuel sait qu'ils ont bien vu le message m. En plus des variables d'initialisation, les valeurs des variables *pending* et *delivered* sont stockées dans l'espace de stockage stable.

2) Événement de récupération : après la panne, le processus perd toutes les informations stockées au niveau de sa mémoire locale. C'est pourquoi il est important de prévoir un mécanisme permettant au processus de retrouver son état d'avant la panne. Le processus doit récupérer la liste des messages délivrés auparavant ainsi que la liste des messages qui ont été vus mais pas encore délivrés, ce qui correspond à la récupération des variables

upon event $< Init >$ **do**
  $delivered := \emptyset$;
  $pending := \emptyset$;
  forall m **do** $ack[m] := \emptyset$;
  $store(pending,\ delivered)$;
end event

FIGURE 3.8 – Événement d'initialisation

delivered and *pending*. Ces deux variables sont stockées dans un espace de stockage fixe (les données ne sont pas supprimées de cet espace de stockage lorsque le processus tombe en panne). La première requête à exécuter par un processus qui vient de se remettre en marche est la récupération de ces deux variable à partir de l'espace de stockage stable (voir figure 3.9). La deuxième opération à faire est de délivrer la liste de messages contenus dans la variable *delivered*. La dernière opération est de faire un broadcast des messages contenus dans l'ensemble *pending* afin d'assurer que tous les autres processus vont recevoir ces messages.

upon event $< Recovery >$ **do**
  $retrieve(pending,\ delivered)$;
  trigger $< Deliver \mid delivered >$;
  forall $(s, m) \in pending$ **do**
    trigger $< Broadcast \mid [\ s,\ m] >$;
  end forall
end event

FIGURE 3.9 – Événement de récupération (Recovery)

3) Événement de broadcast : lorsqu'un processus fait le broadcast d'un message m, il commence par ajouter le champs $(self, m)$ à la variable *pending* (voir figure 3.10), ensuite la nouvelle valeur de la variable *pending* est stockée dans l'espace de stockage stable. Enfin, le message m est envoyé à tous les processus du système en utilisant une liason point-à-point (l'opération *send*). π désigne l'ensemble des processus du système.

upon event $< Broadcast \mid m >$ **do**
  $pending := pending \cup (self, m)$;
  $store(pendig)$;
  forall $q \in \pi$ **do**
    trigger $< Send \mid q,\ m >$;
  end forall
end event

FIGURE 3.10 – Événement de broadcast

4) Événement de livraison (delivery) : l'algorithme correspondant à cet événement montre que tout message reçu par le processus n'est pas forcément délivré. A chaque réception d'un message m, originallement envoyé par s et retransmis par p, il existe deux conditions que l'on doit vérifier (voir figure 3.11). La première est de vérifier si (s, m) est inclus dans l'ensemble *pending*. Si ce n'est pas le cas, alors la variable *pending* est mise à jour en lui ajoutant (s, m). La nouvelle valeur de *pending* est alors stockée dans l'espace de stockage stable. Ensuite, le processus fait un broadcast du message m déstiné à tous les processus du système afin de s'assurer que le message sera vu par tous les processus corrects (même en cas de défaillance de l'expéditeur d'origine s).

La deuxième condition consiste à verifier si le processus p (retransmettant le message m) est inclus dans la variable *ack*. Si ce n'est pas le cas, le processus p est ajouté au tableau *ack*. Après cet ajout, on compte le nombre de processus inclus dans la variable *ack* et on vérifie si ce nombre est plus grand que $N/2$ (la moitié du nombre total des processus du système, avec l'hypothèse que la majorité des processus sont corrects). Si cela est vérifié, on conclut que la majorité des processus ont vu ce message, et comme résultat, le processus peut délivrer le message m (déclencher l'événement *deliver*). Sinon, si $(ack[m]) < N/2$, alors le message ne peut pas encore être délivré.

```
upon event < Receive | p, [ s, m] > do
   if (s, m) ∉ pending then
      pending := pending ∪ (s, m) ;
      store(pendig) ;
       forall q ∈ π do
          trigger < Send | q, m > ;
       end forall
   end if
   if p ∉ ack[m] then
      ack[m] := ack[m] ∪ p ;
       if (ack[m]) > N/2 ∧ (s, m) ∉ delivered then
      delivered := delivered ∪ (s, m) ;
      store(delivered) ;
       trigger < Deliver|delivered > ;
       end if
   end if
end event
```

FIGURE 3.11 – `Événement de livraison (delivery)`

La variante fiable de cet algorithme assure que même si l'expéditeur tombe en panne au milieu d'une opération de broadcast, tous les processus corrects vont délivrer le message. La variante uniforme de l'algorithme assure que si un processus défectueux délivre un

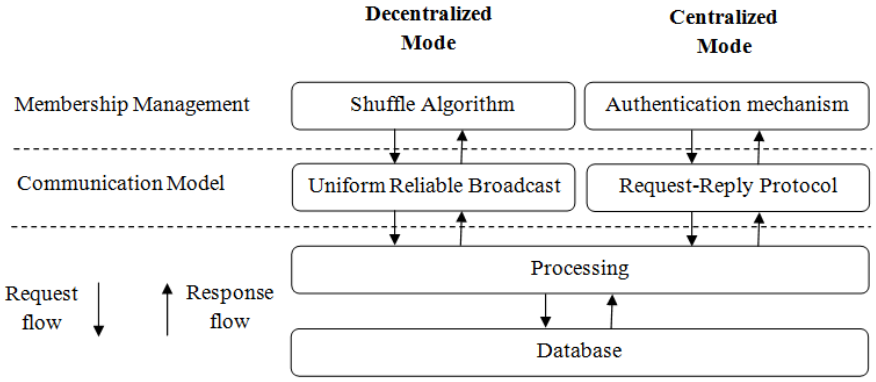


FIGURE 3.12 – Architecture logicielle d'une unité de gouvernance

message, alors tous les processus corrects délivrent ce message, c'est à dire que l'ensemble des messages délivrés par les processus défectueux est un sous-ensemble des messages délivrés par les processus corrects.

3.6 Conception d'un composant du système : architecture logicielle

Dans cette partie, nous discutons l'architecture logicielle de chaque unité de gouvernance du système ReDy [TMD09]. Cette architecture correspond à l'organisation logique des modules logiciels qui implémentent tous les services du système distribué. On se focalise principalement sur comment les modules logiciels sont connectés entre eux et comment les interactions se font entre ces modules.

Dans notre système, les modules logiciels dans chaque unité de gouvernance sont organisés selon *une architecture en couches* [EK08]. L'idée principale de cette architecture est la suivante : un composant dans la couche L_i a le droit de demander des services de la part de la couche en-dessous L_{i-1}, alors que cette dernière a le droit d'envoyer une réponse à la couche d'en haut.

Les modules logiciels de l'unité de gouvernance sont organisés en quatre couches (voir figure 3.12). Dans la suite, nous décrivons les caractéristiques relatives à chaque couche.

1) `La couche de la gestion de l'adhésion au réseau` : dans cette couche, on implémente les services relatifs à la construction du graphe du système. Sachant que notre système est conçu selon une architecture hybride, cette couche est composée de deux

modules logiciels implémentant les deux modes suivants :

(a) *Le mode décentralisé* : ce mode est implémenté par l'algorithme de brassage qui est responsable de la construction du graphe des unités de gouvernance. Ce module instantie l'algorithme de brassage vu auparavant. L'ensemble des composants change périodiquement dans notre système, en particulier l'ensemble des voisins de l'unité de gouvernance actuelle. Cette couche est responsable de la prise de décision, à chaque instant, concernant la composition actuelle de l'ensemble des composants considérés comme voisins par l'unité de gouvernance actuelle. Cette information est périodiquement envoyée à la couche en dessous.

(b) *Le mode centralisé* : ce mode est implémenté par un module de mécanisme d'authentification qui est responsable de la gestion des unités d'action et des unités de détection du sous-système en question. Ces unités sont connectées à l'unité de gouvernance actuelle avec une connection client/serveur, avec l'unité de gouvernance agissant comme serveur et les unités de détection et d'action agissant comme ses clients correspondants. L'ensemble des clients dans ce sous-système est construit en utilisant un mécanisme d'authentification, c'est à dire, une unité d'action (respectivement une unité de détection) doit s'authentifier à l'unité de gouvernance afin de rejoindre le sous-système. De même que pour le mode décentralisé, cette couche envoie l'ensemble des clients participant dans le système à la couche en dessous.

2) `La couche de communication` : les règles de communication entre les entités du système sont implémentées dans cette couche. Les informations reçues au niveau de cette couche sont envoyées à la couche d'en bas qui est la couche de traitement. De même que pour la première couche, nous avons un mode décentralisé et un mode centralisé :

(a) *Le mode décentralisé* : la communication entre les unités de gouvernance est assurée en utilisant un protocole de broadcast uniforme et fiable instancié dans un module spécifique. L'information à diffuser est envoyée à l'ensemble des voisins définis par le module implémentant l'algorithme de brassage.

(b) *Le mode centralisé* : la communication entre l'unité de gouvernance actuelle et les unités de détection et d'action est assurée en utilisant un protocole classique de *requête-réponse (request-reply protocol)*. A n'importe quel moment, l'unité de gouvernance peut recevoir une requête de la part de n'importe quelle unité de détection (respectivement unité d'action) et peut envoyer une requête ou une réponse à une unité d'action ou de

détection reconnue. Les unités d'action et de détection reconnues sont des unités présentes dans la liste des clients participant à la communication. Cette liste est reçue de la part de la couche d'en haut.

3) `la couche de traitement` : cette couche est responsable de la prise de décisions adéquates, dépendemment de l'information reçue de la part des modules implémentant le mode centralisé et le mode décentralisé. Toutes les informations échangées lors de la communication sont redirigées vers la couche de traitement qui est implémentée par un module unique. Ce module effectue une analyse de l'information ou de l'événement reçu, utilise les données stockées afin de prendre des décisions (relatives au cas d'application), et envoie ces décisions à la couche de communication afin d'exécuter ces décisions que ce soit dans le mode local en utilisant le module de communication du mode centralisé, ou bien dans le mode distant en utilisant le module de communication du mode décentralisé. En plus, cette couche est toujours en contact avec la couche de données en y envoyant des requêtes de stockage, de mise à jour ou d'affichage de données. La couche de traitement utilise les services de la couche de données afin d'utiliser les données de façon optimale.

4) `La couche de données` : un sytème de gestion de bases de données est implémenté dans cette couche. Toutes les informations de l'unité de gouvernance sont regroupées dans cette base de données. Cette couche est responsable de collecter, stocker, traiter et afficher les données.

3.7 Processus incrémental et itératif de la conception de l'architecture ReDy basé sur la validation formelle

Afin de concevoir des systèmes IoT qui répondent aux exigences requises, nous utilisons une approche itérative et incrémentale. La figure 3.13 illustre le processus suivi pour réaliser cette approche. Dans la suite, nous allons expliquer les étapes suivies dans ce processus.

Tout d'abord, nous commençons par la phase de spécification qui consiste à préciser les exigences générales (besoins non fonctionnels) de notre système ainsi que l'architecture adoptée (modèle de l'architecture), ce qui permet de définir, par la suite, les fonctionnalités requises (besoins fonctionnels).

Ensuite, nous passons à la phase de conception du système qui consiste à définir comment

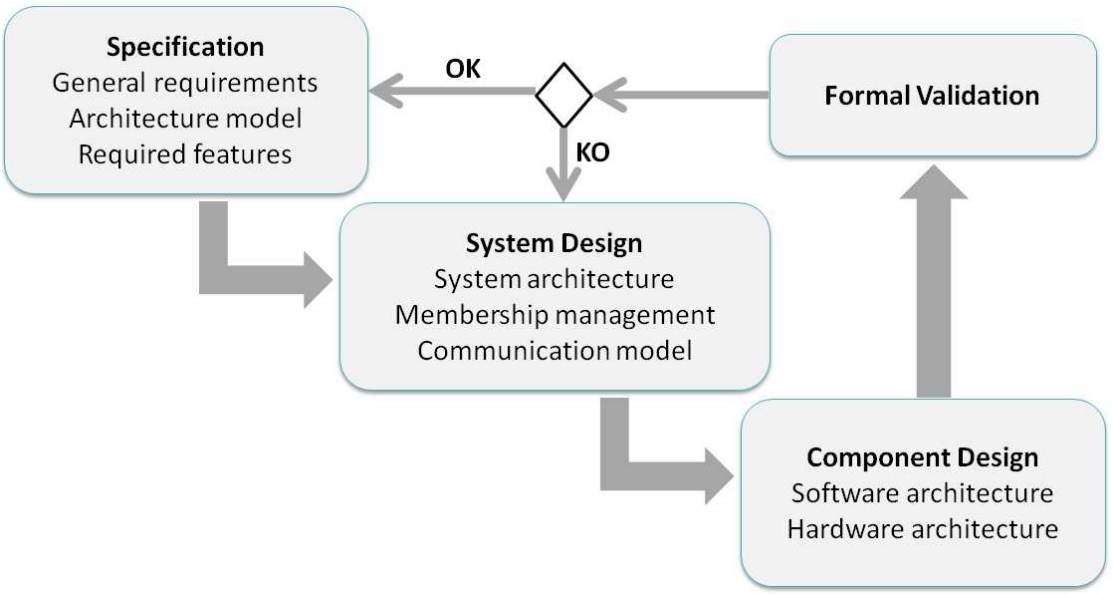


FIGURE 3.13 – Processus incrémental et itératif basé sur la validation formelle

les composants du système IoT sont organisés (architecture du système), et comment le réseau des composants de ce système est construit (membership management), ainsi que le mécanisme de communication entre les différents composants (modèle de communication).

La troisième phase concerne la conception d'un composant du système qui consiste à préciser l'architecture logicielle d'un composant, ce qui correspond à définir comment les différents modules logiciels d'un composant sont logiquement organisés. Ensuite nous définissons l'architecture physique du composant qui est principalement reliée au champs d'application en question.

Passées ces trois étapes, nous passons à la phase de validation formelle. A cette phase, nous modélisons formellement la solution proposée et nous la validons. Si notre modèle n'est pas valide formellement, nous modifions les paramètres définis dans la phase de conception à plusieurs reprises (plusieurs itérations) jusqu'à l'obtention d'une version valide. Une fois le modèle est formellement valide, alors nous pouvons modifier dans les paramètres de la phase de spécification afin d'incrémenter notre dernière version valide et nous exécutons à nouveau notre processus. Ainsi nous obtenons une conception incrémentale et itérative basée sur les résultats de la validation formelle.

Il est à noter que la phase de la validation formelle sera explicitée en détails dans le chapitre suivant.

3.8 Conclusion

Dans ce chapitre, nous avons tout d'abord présenté l'architecture IoT de bout en bout formée de plusieurs couches, ensuite nous avons présenté notre cas d'étude qui porte sur les Smart Grids et son organisation selon cette architecture de bout en bout. Après, nous avons présenté notre principale contribution qui consite à proposer une architecture ReDy (Reliable and Dynamic) qui est destinée à être utilisée dans la conception des systèmes IoT fiables, dynamiques et extensibles. Nous avons présenté l'architecture ReDy modélisant un système composé de différents sous-systèmes. Cette architecture permet de modéliser les systèmes IoT existants. Une représentation simplifiée de cette architecture a été proposée pour permettre l'analyse et la validation des systèmes IoT. A la fin du chapitre, nous avons présenté l'approche adoptée pour la conception des systèmes ReDy et qui repose sur un processus incrémental et itératif basé sur les résultats de la validation formelle. L'idée principale de ce processus consiste à définir une configuration d'un système IoT et d'effectuer des itérations en se basant sur la validation formelle jusqu'à l'obtention d'une version correcte. A ce niveau, il est possible d'incrémenter la configuration et de relancer les itérations de validation formelle afin d'aboutir à une nouvelle version correcte qui est plus étendue que l'ancienne version. Dans le chapitre suivant, nous allons détailler la validation formelle en présentant la démarche suivie dans cette phase afin de construire un modèle formel et afin de pouvoir l'exploiter.

Chapitre 4

Modélisation et validation formelle de l'architecture ReDy proposée

4.1 Introduction

L'architecture ReDy proposée est déstinée aux systèmes IoT afin de garantir leur fiabilité, dynamisme et extensibilité. Dans notre travail, nous proposons la validation formelle comme outil pour assurer ces critères. Dans ce chapitre, nous commençons par expliquer le processus de validation formelle que nous utilisons en explicitant les différentes étapes de ce processus. Ensuite, nous allons présenter les différents modules de notre modèle formel exprimés en LNT afin de représenter tous les aspects de structure de notre architecture ReDy. Enfin, nous présentons la phase de vérification formelle en citant les différentes propriétés de logique temporelle (exprimées en MCL) vérifiées dans le modèle. A partir du modèle valide et des propriétés verifiées, nous pouvons déduire la fiabilité et l'extensibilité de l'architecture ReDy.

4.2 Présentation du processus de validation formelle

Afin de valider formellement notre architecture ReDy, nous avons suivi un ensemble d'étapes que nous présentons dans la figure 4.1. Tout d'abord, nous commençons par exprimer notre architecture en fichiers LNT. Ces fichiers sont transmis en entrée à l'outil

GENERATOR qui compile les fichiers LNT et donne en sortie un fichier BCG. Il existe plusieurs outils qui permettent de manipuler le fichier BCG. Nous citons BCG_DRAW et BCG_EDIT qui permettent de dessiner le graphe LTS correspondant et BCG_INFO qui permet d'aficher les informations relatives à ce graphe : nombre d'états et de transitions, absence ou présence de deadlocks, etc. Afin de procéder à la vérification formelle nous appliquons une minimisation du BCG en utilisant l'outil BCG_MIN pour avoir un graphe BCG équivalent mais réduit en nombre d'états et de transitions. A ce fichier BCG minimisé, nous exécutons le model checker EVALUATOR 4.0 qui prend en entrée également un fichier MCL afin de vérifier la formule de logique temporelle dans le fichier BCG du modèle. En résultat, si la propriété est vérifiée EVALUATOR 4.0 affiche la sortie, sinon, si la propriété est violée, alors le premier contre exemple rencontré est retourné.

En plus de ce processus de validation classique, nous utilisons également d'autres outils et démarches afin de pouvoir corriger les différentes erreurs syntaxiques et sémantiques rencontrées lors de la validation formelle de notre modèle. Nous utilisons l'outil OCIS qui permet une navigation pas à pas dans le modèle à partir du fichier LNT, ce qui nous donne la possibilité de visualiser les scénarios de simulations à travers une interface graphique. Il est possible également de récupérer ces simulations dans des fichiers BCG.

Nous utilisons également l'outil TERMINATOR qui prend les fichiers LNT en entrée et vérifie directement la présence ou non de deadlocks dans notre modèle. En entrée, il faut préciser également la profondeur de recherche que nous désirons. En cas de présence de deadlocks, TERMINATOR retourne le premier contre exemple rencontré.

4.3 Présentation du modèle de l'architecture ReDy

Dans cette partie, nous présentons la modélisation formelle de l'architecture ReDy en utilisant le langage LNT. Afin de réaliser cet objectif, nous définissons six modules communiquant entre eux. Nous identifions deux parties dans le langage LNT : la partie données où l'on retrouve la définition des types de données, et la partie contrôle où on définit les différents processus. Seul le module *types* fait partie de la partie données de notre programme LNT. Les autres modules font partie de la partie contrôle. Chaque module de la partie contrôle définit un ou plusieurs processus afin de décrire le comportement du système. Le module *main* utilise le module *sub_system*. Le module *sub_system* utilise trois modules : *governance_unit*, *action_unit* et *detection_unit*. Les trois modules précédents utilisent le module *types* (figure 4.2).

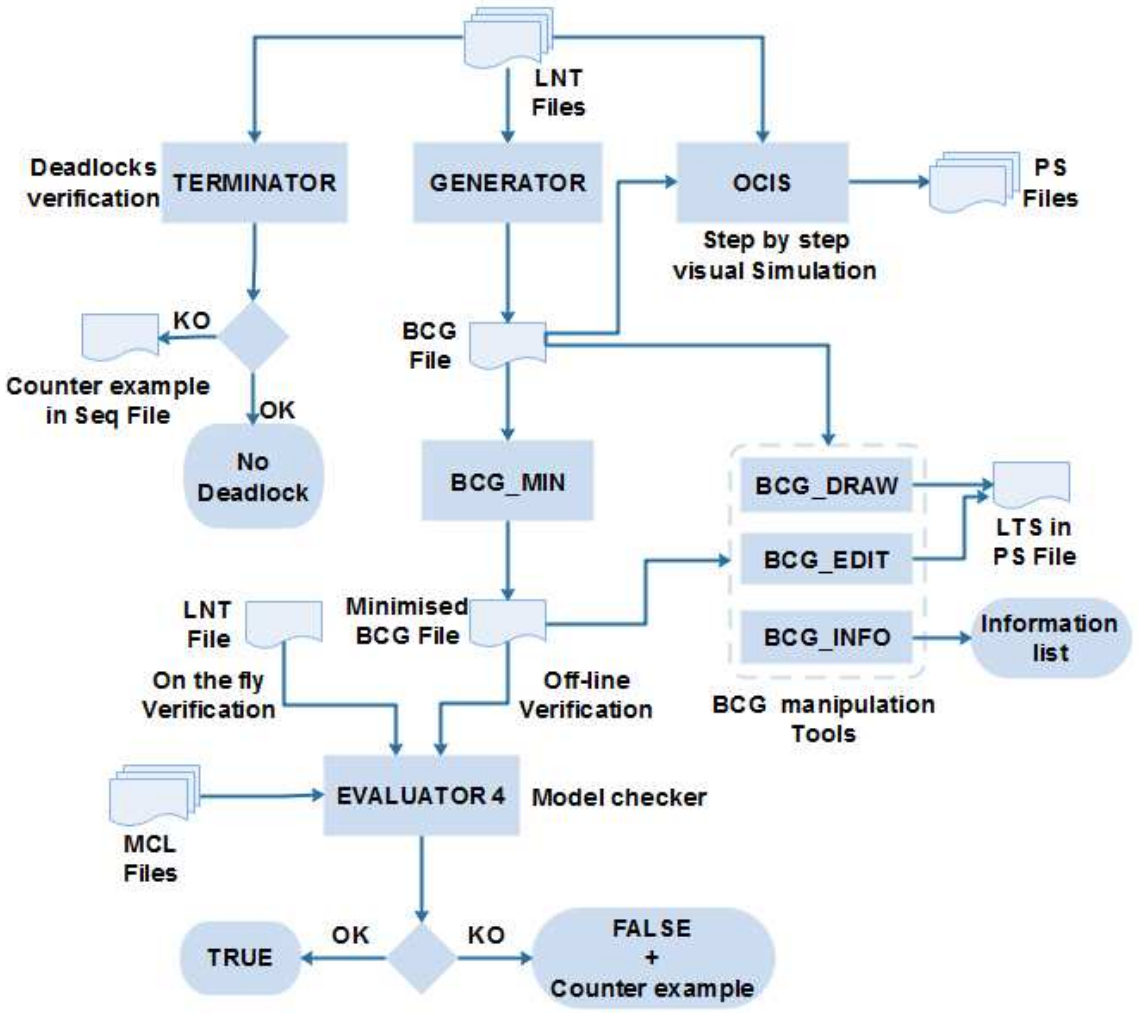


FIGURE 4.1 – Processus de la validation formelle

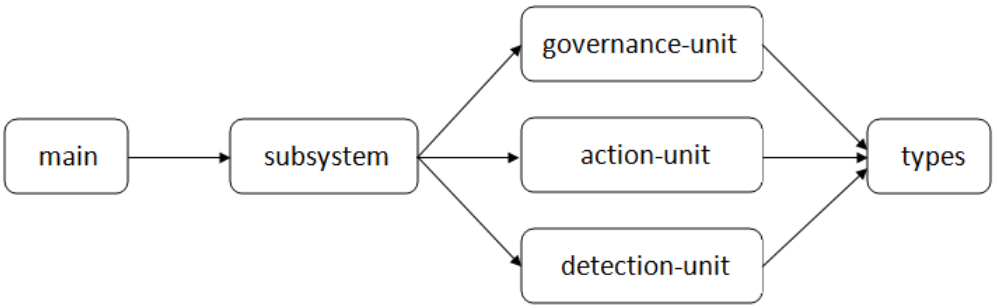


FIGURE 4.2 – Modules LNT de l'architecture ReDy

4.4 Modélisation des types de données

Le module *types* définit les différents types de données utilisés dans ce modèle. Tout d'abord, on définit le type *index_sub_system* comme étant un indice des sous-systèmes composant notre système global. Dans notre cas, nous limitons le nombre d'indices possibles à six sous-systèmes car nous définissons au maximum six sous-systèmes dans notre système global dans un premier temps. On instancie différentes fonctions prédéfinies pour ce type qui sont : la comparaison d'égalité `"=="`, la comparaison d'inégalité `"<>"`, l'infériorité `"<="`, et l'infériorité stricte `"<"`.

```
type index_sub_system is
   range 1 .. 6 of Nat
   with "==","<>","<=", "<"
end type
```

On définit également les types *index_detection* et *index_action* qui représentent les indices des unités de détection et des unités d'action. Ces deux types sont définis avec les mêmes fonctions prédéfinies du type *index_sub_system*. Dans notre cas, nous limitons le nombre d'unités de détection / d'action par sous-système à trois unités.

```
type index_detection is
   range 1 .. 3 of Nat
   with "==","<>","<=", "<"
end type

type index_action is
```

```
   range 1 .. 3 of Nat
   with "==","<>","<=","<"
end type
```

4.5 Modélisation formelle d'un sous-système de l'architecture ReDy (mode local)

Nous allons commencer par modéliser un sous-système unitaire composé d'une unité de gouvernance, de plusieurs unités de détection et de plusieurs unités d'action. Chaque type d'unité est modélisé par un module LNT comportant un ou plusieurs processus. La communication entre les différentes unités est modélisée par la communication entre les différents processus par rendez-vous sur des portes de communication (Gates). La communication entre l'unité de gouvernance et les unités de détection se fait par rendez-vous sur la porte DETECTION. La communication entre l'unité de gouvernance et les unités d'action se fait par rendez-vous sur la porte ACTION (figure 4.3). Il n'y a pas de communication entre les unités de détection et les unités d'action.

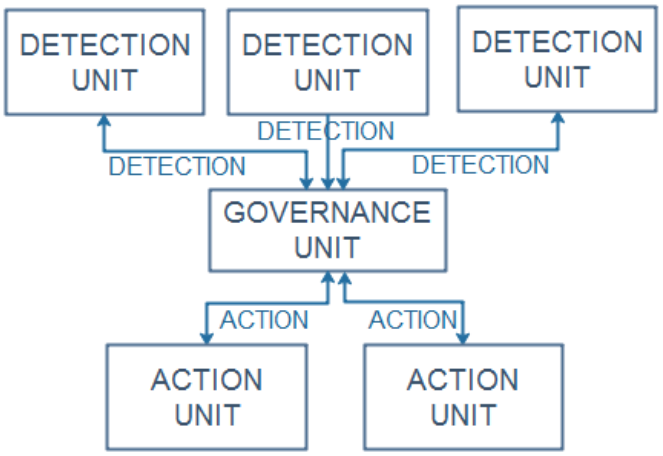


FIGURE 4.3 – Portes de communication dans un sous-système

4.5.1 Modélisation d'une unité de détection DU (Detection Unit)

Le module *Detection_unit* est implémenté par un processus contenant une boucle permanente. Cette boucle exprime des itérations permanentes d'une action de type DETECTION, c'est à dire que l'unité de détection informe l'unité de gouvernance correspondante qu'un événement est détecté. Si l'unité de détection ne détecte aucun événement au cours

de cette itération, alors l'état du système ne va pas avancer.

```
module detection_unit (types) is

process detection_unit[DETECTION : any]
    (id_sub:index_sub_system,id:index_detection)
is
   loop

          DETECTION(id_sub,id)

   end loop
end process

end module
```

4.5.2 Modélisation d'une unité d'action AU (Action Unit)

Le module *Action_unit* est implémenté par un processus contenant une boucle permanente. Cette boucle exprime des itérations permanentes d'une action de type ACTION, c'est à dire que l'unité d'action reçoit l'ordre d'exécution d'une action de la part de l'unité de gouvernance correspondante. Si l'unité d'action ne reçoit aucun ordre d'exécution d'une action de la part de l'unité de gouvernance correspondante au cours de cette itération, alors l'état du système ne va pas avancer.

```
module action_unit (types) is

process action_unit[ACTION:any]
   (id_sub:index_sub_system,id:index_action)
is
   loop

          ACTION(id_sub,id)

   end loop
end process
```

```
end module
```

4.5.3 Modélisation d'une unité de gouvernance GU (Governance Unit)

Le module *Governance_unit* est composé de quatre processus. Le processus *governance_unit* initialise et appelle le processus *membership_management*. Ce processus est responsable de la construction du réseau des composants. En particulier, ce processus est responsable de l'ajout et de la suppression des unités de gouvernance du système actuel. Le processus *membership_management* appelle le processus *local_communication* qui est responsable de la communication à l'intérieur du même sous-système : la communication entre l'unité de gouvernance et les unités de détection est assurée par rendez-vous à la porte DETECTION, alors que la communication entre l'unité de gouvernance et les unités d'action est assurée par rendez-vous à la porte ACTION.

Le processus *local_communication* appelle le processus *distant_communication*. Ce dernier assure la communication entre le sous-système actuel et les autres sous-systèmes, en particulier il assure la communication entre l'unité de gouvernance actuelle avec ses voisins par rendez-vous sur la porte FORWARD. Le processus distant_communication appelle récursivement le processus *membership_management*.

la figure 4.4 illustre les appels entre les processus du module *Governance_unit*.

Pour modéliser l'unité de gouvernance, nous avons opté pour une méthode récursive (au lieu d'une méthode itérative) car c'est la représentation la plus fidèle pour garder la notion de processus implémenté réellement dans des unités IoT. Chaque étape dans la récursion est définie par un choix non déterministe (choix de faire une action ou de ne rien faire) ce qui assure la possibilité d'exécution dans tous les ordres possibles. Ainsi, à un instant donné, une unité de gouvernance peut exécuter trois choix possibles (tous les ordres sont possibles) :

- Exécuter une itération de brassage.

- Exécuter une opération en mode local.

- Exécuter une opération en mode distant.

Dans la suite nous allons expliquer comment le processus de chaque module est modélisé.

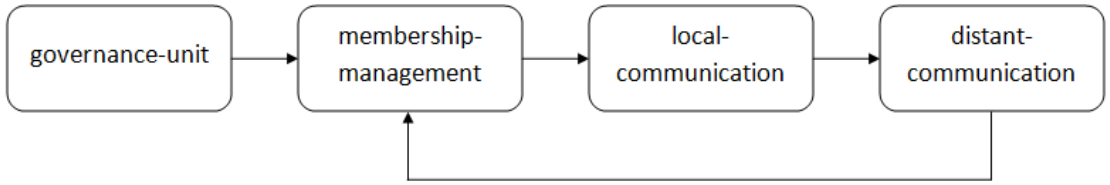


FIGURE 4.4 – Processus du module Governance_unit

Initialisation : Le processus gouvernance_unit

Dans un premier temps, nous commençons par définir le processus gouvernance_unit qui est paramétré par quatre portes : DETECTION, ACTION et FORWARD qui représentent des portes de communication entre les différentes unités du système, et la porte DECISION qui représente une action à exécuter en interne de l'unité de gouvernance. Ce processus est paramétré également par la variable id_sub qui est un indice de type index_sub_system.

Dans le corps du processus, nous définissons trois variables : La variable new_detection est un booléen qui prend la valeur True lorsqu'une détection est faite. La variable decision_done est un booléen qui prend la valeur True lorsqu'une décision est exécutée. La variable action_to_do est un booléen qui prend la valeur True lorsqu'il y a une action à exécuter. Initialement, ces variables prennent la valeur False. Les valeurs de ces variables changent au fil de l'exécution du modèle afin de pouvoir assurer une modélisation correcte du fonctionnement du système.

Dans ce processus, nous effectuons l'initialisation des noeuds du système i.e. nous définissons la vue partielle de chaque noeud dans le système. Cette initialisation sera explicitée dans le chapitre VI.

Enfin, le processus gouvernance_unit appelle le processus membership_management en lui passant les variables initialisées précédemment en paramètres.

```
process gouvernance_unit[DETECTION, DECISION, ACTION, FORWARD : any]
   (id_sub:index_sub_system)
is
   var
     new_detection : bool,
       decision_done : bool,
```

```
        action_to_do : bool
    in
       new_detection := false;
       decision_done := false;
       action_to_do := false;

-- Nodes initialization here --

       membership_management[DETECTION, DECISION, ACTION, FORWARD]
          (id_sub, new_detection, decision_done, action_to_do,  id_sub)

    end var
end process
```

Exécution d'une itération de brassage : Le processus membership_management

L'étape suivante consiste à exécuter une itération de brassage en définissant le processus membership_management. Ce processus est paramétré par quatre portes de communication et par cinq variables. Le but de ce processus est de modéliser l'algorithme de brassage responsable de la gestion de l'adhésion des composants au système. Le coprs de ce processus sera explicité dans le chapitre VI.

Ce processus appelle le processus local_communication en lui passant en paramètre les cinq variables déclarées.

```
process membership_management[DETECTION, DECISION, ACTION, FORWARD : any]
    (id_sub:index_sub_system,
     new_detection : bool,
     decision_done : bool,
     action_to_do : bool,
     id_sub_other :index_sub_system
     )
is
    -- shuffling algorithm here  --
```

```
   local_communication [DETECTION, DECISION, ACTION, FORWARD]
      (id_sub, new_detection, decision_done, action_to_do, id_sub_other)
end process
```

Exécution d'une opération en mode local : Le processus local_communication

Le processus local_communication assure l'exécution d'une communication en mode local, ce qui correspond à une communication entre l'unité de gouvernance et l'unité de détection ou l'unité d'action. Ce processus est paramétré par quatre portes de communication et par cinq variables.

Dans le corps de ce processus, nous commençons par définir deux variables : id_detection_unit de type index_detection et id_action_unit de type index_action. Ces deux variables permettent d'identifier l'unité de détection ou l'unité d'action engagée dans une communication.

Ensuite, nous définissons un choix non déterministe avec quatre branches. La première branche comporte un null, ce qui permet au système de ne pas exécuter une opération en mode local et de passer au processus suivant. C'est grâce à cette branche que nous assurons toutes les combinaisons possibles (ordre d'exécution) entre les différents processus définis. La deuxième branche définit un rendez-vous sur la porte DETECTION entre l'unité de gouvernance d'indice id_sub et l'unité de détection d'indice id_detection_unit. Une fois le rendez-vous exécuté, la variable new_detection prend la valeur True. La troisième branche définit une action de type DECISION. En effet, si nous avons une nouvelle détection (la variable new_detection est True), alors nous avons DECISION qui est exécuté par l'unité de gouvernance d'indice id_sub. Ensuite la variable new_detection prend la valeur False signifiant que le système peut faire d'autres détections, et la variable action_to_do prend la valeur True, ce qui signifie que le système doit exécuter une action correspondant à la détection faite. La quatrième branche définit un rendez-vous sur la porte ACTION entre l'unité de gouvernance d'indice id_sub et l'unité d'action d'indice id_action_unit. Une fois le rendez-vous exécuté, la variable action_to_do prend la valeur False ce qui signifie que le système n'a pas d'action à exécuter, et la variable new_detection prend la valeur False signifiant que le système peut faire d'autres détections.

Une fois la définition des quatre branches du choix non déterministe est faite, on appelle

le processus distant_communication en lui passant en paramètres quatre portes de communication et cinq variables.

```
process local_communication [DETECTION, DECISION, ACTION, FORWARD:any]
   (id_sub:index_sub_system,
    new_detection : bool,
    decision_done : bool,
    action_to_do : bool,
    id_sub_other:index_sub_system)
is
   var id_detection_unit: index_detection,
       id_action_unit: index_action
   in

   select
      null
   []
      only if not(new_detection) and not (action_to_do) then
         DETECTION(id_sub, ?id_detection_unit);
         new_detection :=true
      end if
   []
      only if new_detection then
         DECISION(id_sub);
         new_detection :=false;
         action_to_do := true;
         id_sub_other := id_sub
      end if
   []
      only if action_to_do then
         ACTION(id_sub_other, id_sub, ?id_action_unit);
         action_to_do := false;
         new_detection :=false
      end if
```

```
   end select;
   distant_communication [DETECTION, DECISION, ACTION, FORWARD]
      (id_sub, new_detection, decision_done, action_to_do, id_sub_other)
   end var
end process
```

Exécution d'une opération en mode distant : Le processus distant_communication

Le processus distant_communication assure l'exécution d'une communication en mode distant, ce qui correspond à une communication entre les unités de gouvernance. Ce processus est paramétré par quatre portes de communication et par cinq variables.

Dans le corps de ce processus, nous définissons un choix non déterministe avec trois branches. La première branche comporte un null, ce qui permet au système de ne pas exécuter une opération en mode distant et de passer au processus suivant. C'est grâce à cette branche que nous assurons toutes les combinaisons possibles entre les différents processus définis. La deuxième branche définit une action de type FORWARD de la part de l'unité de gouvernance d'indice id_sub. Cette branche est exécutée dans le cas où nous avons une nouvelle détection mais l'unité de gouvernance décide que l'action doit être exécutée par une autre unité de gouvernance. Une fois le rendez-vous exécuté, la variable new_detection prend la valeur False pour préparer le système à une nouvelle détection. La troisième branche définit la réception d'une action de type FORWARD de la part de l'unité de gouvernance d'indice id_sub, envoyée par l'unité de gouvernance d'indice id_sub_other. Une fois le rendez-vous exécuté, la variable action_to_do prend la valeur True ce qui signifie qu'une action doit être exécutée au niveau du sous-système d'indice id_sub.

Une fois la définition des trois branches du choix non déterministe est faite, on appelle récursivement le processus membership_management en lui passant en paramètre quatre portes de communication et cinq variables.

```
process distant_communication [DETECTION, DECISION, ACTION, FORWARD:any]
   (id_sub:index_sub_system,
    new_detection : bool,
```

```
    decision_done : bool,
    action_to_do : bool,
    id_sub_other:index_sub_system)
is

    select
      null
   []
      only if (new_detection and not (action_to_do)) then
         FORWARD(id_sub);
         new_detection := false
      end if
   []
      FORWARD(?id_sub_other) where (id_sub_other<>id_sub);
      action_to_do:=true

   end select;

   membership_management [DETECTION, DECISION, ACTION, FORWARD]
      (id_sub, new_detection, decision_done, action_to_do, id_sub_other)
end process
```

4.5.4 Modélisation d'un sous-système unitaire

Dans le module *sub_system*, nous définissons une composition parallèle entre les composants du sous-système i.e les processus modélisant l'unité de gouvernance, l'unité de détection et l'unité d'action (figure 4.5).

on peut définir différentes structures de sous-systèmes avec différents nombres d'unités d'action et d'unités de détection. Nous utilisons deux méthodes pour composer un sous-système unitaire :

- Une méthode figée qui permet d'instanstier des sous-systèmes de mêmes caractéristques (même nombre d'unités de détection et même nombre d'unités d'action pour tous les sous-systèmes).

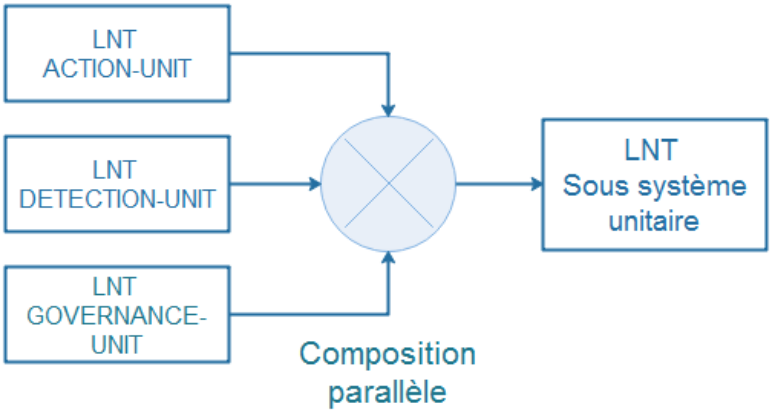


FIGURE 4.5 – Composition parallèle d'un sous-système unitaire

- Une méthode paramétrable qui permet d'instantier des sous-systèmes de caractéristiques différentes.

Dans la suite, nous présentons les deux méthodes de modélisation des sous-systèmes.

Modèle figé d'un sous-système

Le modèle figé permet de définir une seule combinaison possible pour tous les sous-systèmes. Nous présentons l'exemple d'un sous-système composé d'une unité de gouvernance, de deux unités d'action et de trois unités de détection. Pour ce faire, nous définissons un processus *sub_system* qui est défini par la mise en parallèle d'un processus *governance_unit* avec deux processus *action_unit* et trois processus *detection_unit* sur les portes de communication ACTION et DETECTION.

```
module sub_system(governance_unit, action_unit, detection_unit) is

process sub_system [DETECTION, DECISION, ACTION, FORWARD,
SHUFFLING_TRANSFER : any](nb_neighbors:NAT,SL:NAT,id_sub:index_sub_system)
is
      par DETECTION, ACTION in
         governance_unit[DETECTION, DECISION, ACTION, FORWARD,
   SHUFFLING_TRANSFER](nb_neighbors,SL,id_sub)
      ||
         par
              par
```

```
                action_unit[ACTION](id_sub, index_action(1))
              ||
                action_unit[ACTION](id_sub, index_action(2))
              end par
         ||
              par
               detection_unit[DETECTION](id_sub, index_detection(1))
              ||
               detection_unit[DETECTION](id_sub, index_detection(2))
              ||
               detection_unit[DETECTION](id_sub, index_detection(3))
              end par
         end par
      end par
end process

end module
```

Modèle paramétrable d'un sous-système

Afin d'avoir plus de choix pour définir un sous-système, nous définissons le modèle paramétrable permettant d'avoir plusieurs combinaisons possibles. L'exemple illustré dans le module *sub_system* nous permet d'instantier quatre types de sous-systèmes : (1GU, 1DU, 1AU) , (1GU, 1DU, 2AU) , (1GU, 2DU, 1AU), (1GU, 2DU, 2AU). Le choix exprimé (par if elsif) dans la combinaison parallèle entre les différents processus permet d'instancier le type de sous-système requis.

Dans l'exemple donné, le nombre maximal d'unités de détection (et d'actions) est limité à deux. Pour augmenter le nombre d'unités par sous-système, il suffit d'ajouter une condition dans la branche concernée et de passer les paramètres adéquats dans la condition et de mettre en parallèle un nombre de processus égal au nombre d'unités désiré.

```
module sub_system(gouvernance_unit, action_unit, detection_unit) is

process sub_system [DETECTION, DECISION, ACTION, FORWARD : any]
   (id_sub:index_sub_system, nb_detection:index_detection, nb_action:index_action)
```

```
is

      par DETECTION, ACTION in
         gouvernance_unit[DETECTION, DECISION, ACTION, FORWARD](id_sub)
      ||
         par
            -- action units
              if (nb_action==index_action(1)) then
                 action_unit[ACTION](id_sub,index_action(1))
              elsif (nb_action==index_action(2)) then
                 par
                    action_unit[ACTION](id_sub,index_action(1))
                 ||
                    action_unit[ACTION](id_sub,index_action(2))
                 end par
              end if
         ||
            -- detection units
              if (nb_detection==index_detection(1)) then
                 detection_unit[DETECTION](id_sub,index_detection(1))
              elsif (nb_detection==index_detection(2)) then
                 par
                    detection_unit[DETECTION](id_sub,index_detection(1))
                 ||
                    detection_unit[DETECTION](id_sub,index_detection(2))
                 end par
              end if
         end par
      end par
end process

end module
```

4.5.5 Génération d'un LTS d'un sous-système

Une fois le modèle du sous-système est exprimé en LNT, nous pouvons générer le fichier BCG qui contient une représentation graphique (LTS) du comportement du sous-système. La figure 4.6 représente le LTS généré pour un sous-système composé d'une unité de gouvernance, d'une unité de détection et d'une unité d'action.

Pour obtenir un graphe plus simple équivalent au graphe généré, nous appliquons une minimisation à ce dernier en utilisant l'outil BCG_MIN de la boite à outils CADP . Le résultat est un graphe réduit en nombre d'états et de transitions (figure 4.7).

4.5.6 Exemples de génération du modèle du sous-système et leurs caractéristiques

Dans la suite, nous donnons des exemples de génération du modèle du sous-système en utilisant la méthode paramétrable. Le tableau 4.1 résume les caractéristiques des exemples générés du modèle formel exprimé en LNT. Le résultat de génération de ce modèle est un fichier BCG représentant un LTS. La colonne d (respectivement a) représente le nombre d'unités de détection (respectivement d'action) dans le sous-système généré. Le nombre d'états et de transitions du LTS généré est exprimé dans la colonne *Sans réduction*. A ce LTS généré, nous appliquons une réduction (minimisation) pour avoir un LTS équivalent de taille réduite. Le nombre d'états et de transitions de ce LTS réduit est exprimé dans la colonne *Avec réduction*.

TABLE 4.1 – Exemples de génération du modèle du sous-système

	d	a	Sans réduction		Avec réduction	
			Etats	Transitions	Etats	Transitions
Exemple 1	1	1	19	56	10	32
Exemple 2	2	1	47	153	24	81
Exemple 3	1	2	19	63	10	39
Exemple 4	2	2	47	170	24	98
Exemple 5	3	3	103	436	52	255
Exemple 6	4	4	215	1038	108	616

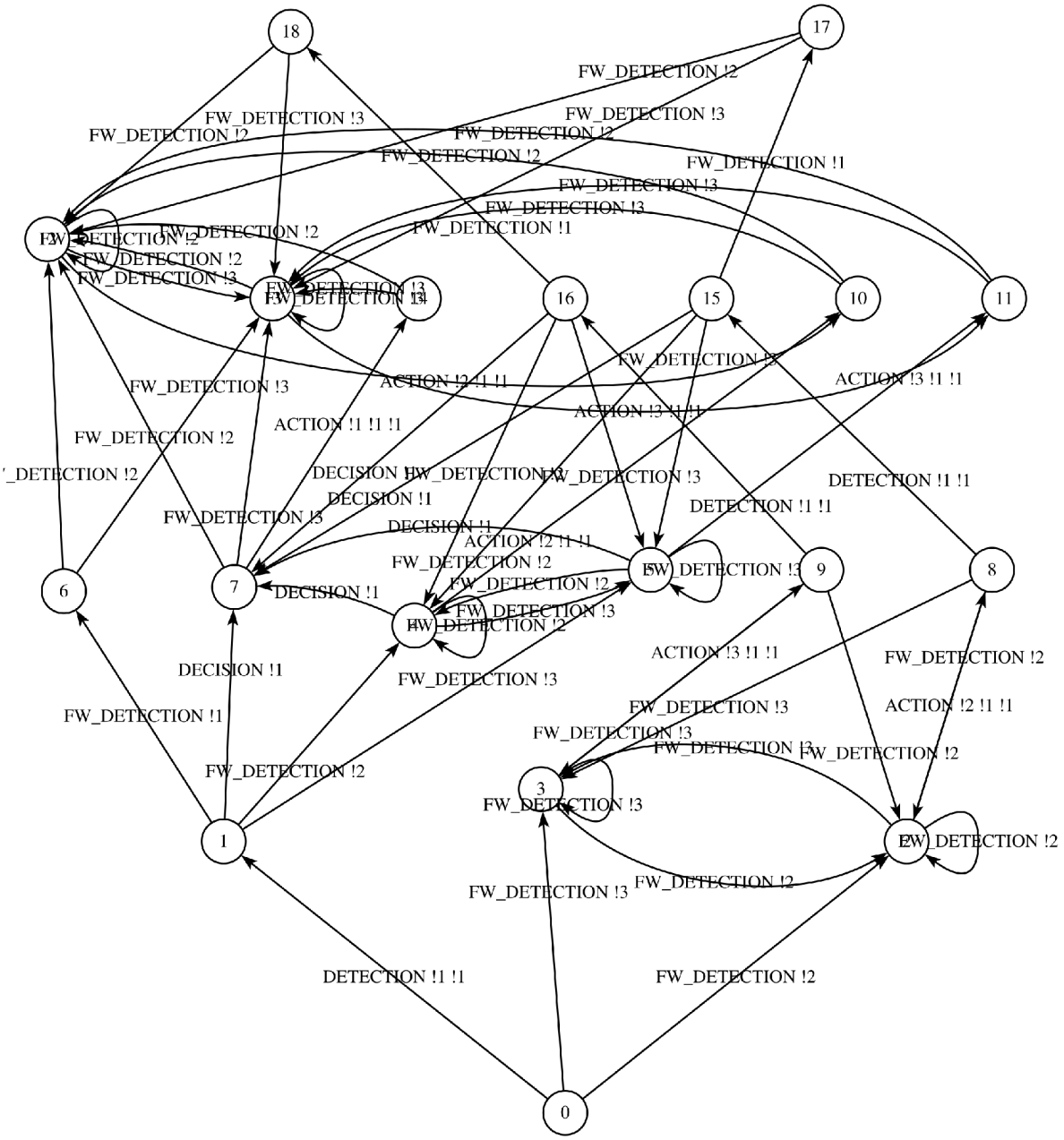


FIGURE 4.6 – LTS d'un sous-système

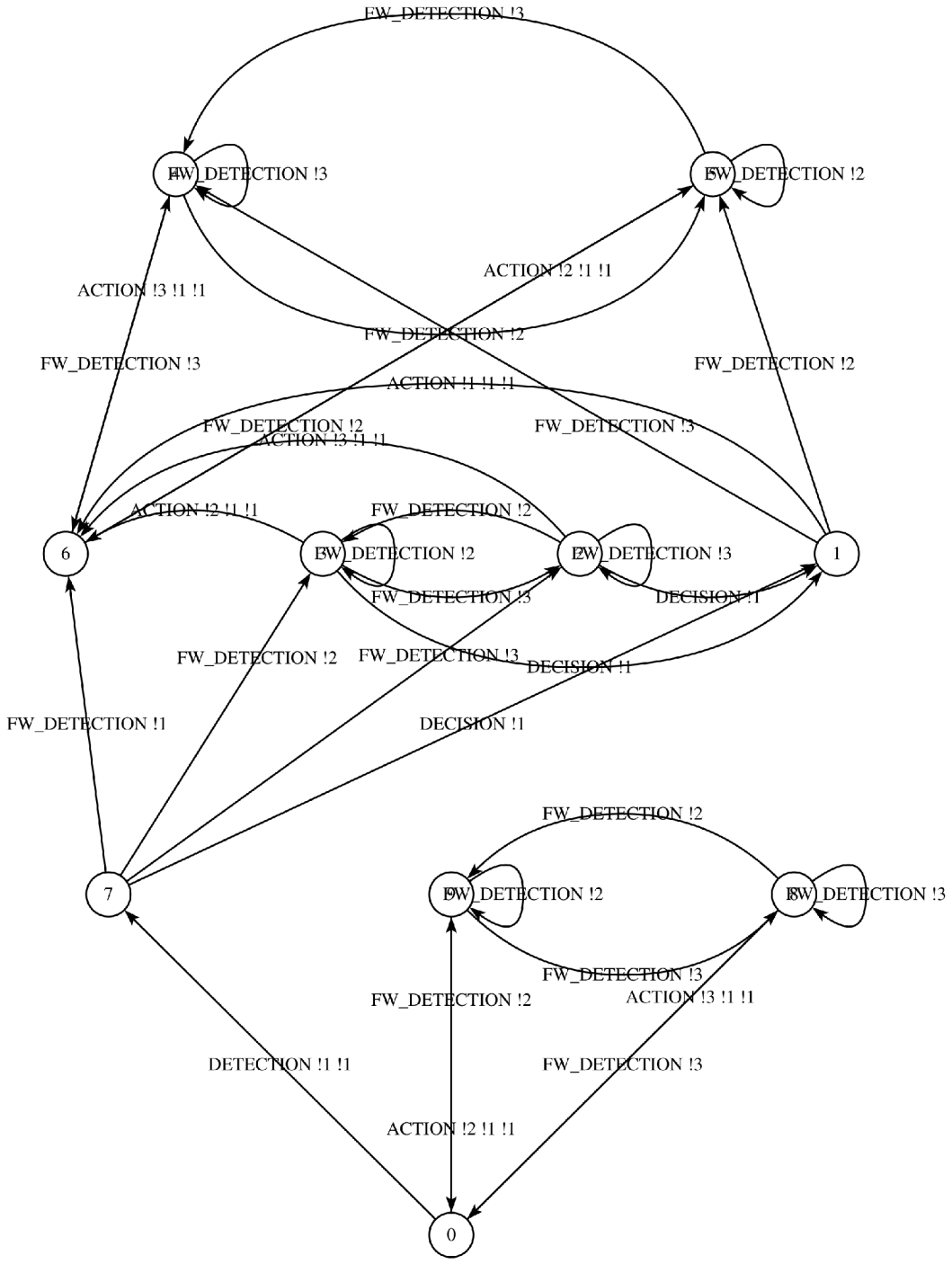


Figure 4.7 – LTS minimisé d'un sous-système

4.6 Modélisation formelle de l'architecture ReDy (mode distant)

4.6.1 Présentation du modèle formel de l'architecture ReDy

Afin de modéliser le système global de l'architecture ReDy, nous effectuons une composition parallèle de plusieurs sous-systèmes (voir figure 4.8). Les différents sous-systèmes sont exprimés en LNT ainsi que la composition parallèle qui en résulte.

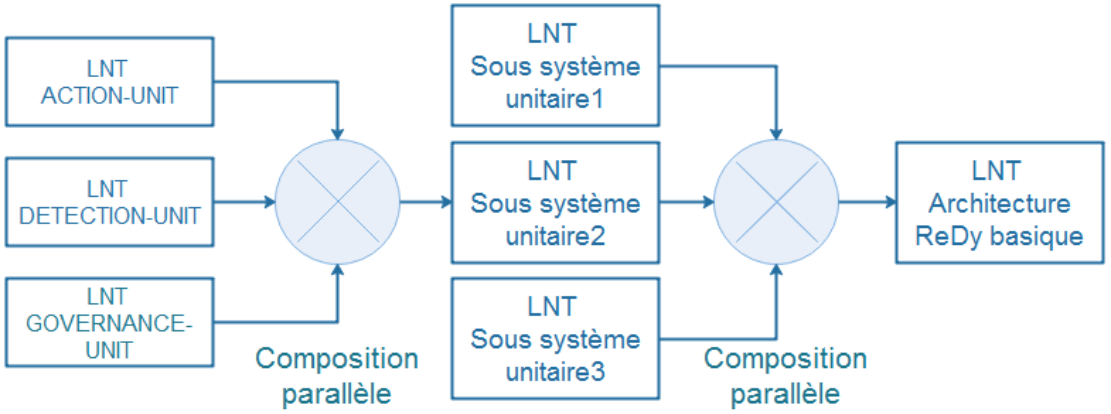


FIGURE 4.8 – Composition parallèle d'une architecture ReDy

Les portes utilisées pour la communication centralisée au sein d'un sous-système sont les suivants : DETECTION, DECISION, ACTION. La porte utilisée pour la communication décentralisée entre les différents sous-systèmes est la porte FORWARD, qui est utilisée pour informer les autres sous-systèmes qu'un problème est détecté (détection) (voir figure 4.9).

Dans la suite, nous donnons un modèle d'un système ReDy exprimé en LNT. Nous définissons une composistion parallèle entre plusieurs sous-systèmes. Dans notre cas, nous avons quatre sous-systèmes communiquants. La composition parallèle est définie par une communication sur la porte FORWARD.

```
module main(sub_system) is
!nat_bits 3

process MAIN [DETECTION, DECISION, ACTION, FORWARD : any]
is
```

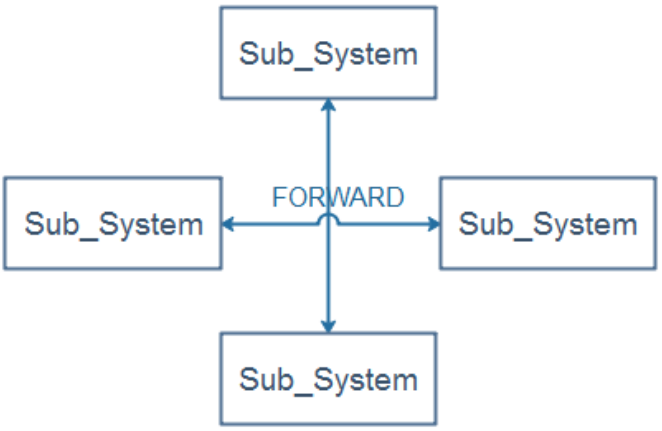


FIGURE 4.9 – Portes de communication dans un système ReDy

```
      par FORWARD in
         sub_system [DETECTION, DECISION, ACTION, FORWARD]
                    (index_sub_system(1),index_detection(1),index_action(1))
                     -- sub_system number 1 with 1 DU and 1 AU
      ||
         sub_system [DETECTION, DECISION, ACTION, FORWARD]
                    (index_sub_system(2),index_detection(1),index_action(2))
                     -- sub_system number 2 with 1 DU and 2 AUs
      ||
         sub_system [DETECTION, DECISION, ACTION, FORWARD]
                    (index_sub_system(3),index_detection(2),index_action(1))
                     -- sub_system number 3 with 2 DUs and 1 AU
||
         sub_system [DETECTION, DECISION, ACTION, FORWARD]
                    (index_sub_system(4),index_detection(1),index_action(1))
                     -- sub_system number 4 with 1 DUs and 1 AU
      end par

end process

end module
```

4.6.2 Exemples de génération du modèle de l'architecture ReDy et leurs caractéristiques

Dans la suite, nous donnons des exemples de génération du modèle d'un système conçu selon l'architecture ReDy. Le tableau 4.2 résume les caractéristiques des exemples générés du modèle formel exprimé en LNT. Le résultat de génération de ce modèle est un fichier BCG représentant un LTS. La colonne g représente le nombre d'unités de gouvernance (et par la suite le nomrbre de sous-systèmes) présents dans le système. La colonne d (respectivement a) représente le nombre d'unités de détection (respectivement d'unités d'action) dans chaque sous-système généré. Le nombre d'états et de transitions du LTS généré sont exprimés dans la colonne *Sans réduction*. A ce LTS généré, nous appliquons une réduction (minimisation) pour avoir un LTS équivalent de taille réduite. Le nombre d'états et de transitions de ce LTS réduit sont exprimés dans la colonne *Avec réduction*.

TABLE 4.2 – Exemples de génération du modèle de l'architecture ReDy

	g	d	a	Sans réduction		Avec réduction	
				Etats	Transitions	Etats	Transitions
Exemple 1	2	2	2	163	627	37	141
Exemple 2	3	3	3	2087	8955	171	714
Exemple 3	3	6	6	2087	13079	171	1116
Exemple 4	3	9	9	2087	17203	171	1518
Exemple 5	4	4	4	43708	251975	1257	7004
Exemple 6	5	5	5	798292	5760947	8403	58520
Exemple 7	6	6	6	13305262	115204621	52605	439590

4.7 Validation formelle du modèle de l'architecture ReDy

A partir du modèle formel de l'architecture ReDy présenté dans la section précédente, nous pouvons appliquer les techniques de model checking pour vérifier un ensenmble de propriétés de logique temporelle exprimées en MCL.

Il existe deux types de propriétés : une propriété de vivacité (liveness) qui exprime que quelque chose de bien arrive bien, par opposition aux propriétés de sûreté (safety) exprimant que quelque chose de mal n'arrive jamais.

Une propriété de vivacité peut détecter deux types de problèmes :

- L'inter-blocage (deadlock) qui se produit à cause d'une attente mutuelle de deux processus.

- Le blocage opérationnel (livelock) qui se produit quand le système entre dans une boucle qui ne fait pas avancer l'état du système. C'est un système qui tourne dans le vide.

La vérification fonctionnelle nous amène à garantir les critères recherchés dans notre architecture ReDy, à savoir :

- La fiabilité de la structure de notre architecture ReDy par l'absence d'inter-blocages et de blocages operationnels.

- L'extensibilité de l'architecture ReDy par l'absence d'inter-blocages et de blocages operationnels dans toutes les versions incrémentées de l'architecture ReDy (présentées dans le tableau 4.2)

Dans la suite, nous allons présenter la propriété de vivacité ordre Detection Action.

4.7.1 Propriété ordre Detection Action

La propriété ordre Detection Action exprime que toute détection faite dans le système doit être suivie d'une action lui correspondant. Afin d'exprimer cette propriété, nous définissons une macro MCL classique dans les langages temporels qui exprime l'inévitabilité.

```
macro inevitable (L) =
   mu X . ( < true > true and [ not L ] X )
end_macro
```

Cette macro utilise un opérateur du point fixe minimal « mu » (défini par le mu-calcul [MT08b]) appliqué sur la proposition « X ». Cette macro exprime que toutes les transitions qui sortent de l'état courant vont mener à une action « L » après un nombre fini d'étapes, ce qui veut dire intuitivement : depuis l'état actuel, on arrivera à « L » quelque soit la branche prise.

La formule MCL exprimant la propriété de l'ordre Detection Action est la suivante :

```
[ true * . {DETECTION ?gu:Nat ...} ]
inevitable ( {ACTION !gu ...} )
```

Cette formule exprime que si on exécute une DETECTION (détection est faite), on arrivera inévitablement (dans toutes les branches) à une ACTION lui correspondant. On schématise cette propriété dans la figure 4.10.

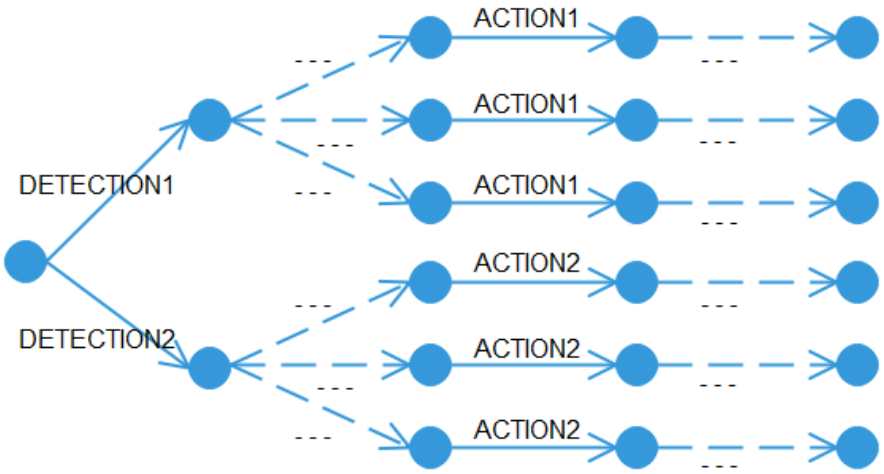


FIGURE 4.10 – Propriété MCL ordre Detection Action

4.7.2 Absence d'inter-blocages (deadlocks)

Inter-blocage relevé par la propriété de vivacité

La propriété exprimée n'a détecté aucun inter-blocage dans le modèle actuel.

Inter-blocage relevé par d'autre outils CADP

Afin de vérifier l'absence d'inter-blocages dans le modèle formel de l'architecture ReDy, nous avons deux possibilités :

- Utiliser l'outil TERMINATOR : cet outil prend directement le fichier LNT et vérifie l'existence ou non de deadlocks dans le modèle. Cet outil prend en paramètre la profondeur de recherche désirée dans le graphe. Il existe des cas particuliers où le TERMINATOR ne détecte pas l'inter-blocage.

- Utiliser l'outil EVALUATOR 4.0 : cet outil prend deux fichiers en entrée, le fichier BCG minimisé du modèle LNT et le fichier MCL exprimant la propriété de logique temporelle, dans ce cas la propriété exprimant l'inter-blocage. L'utilisation de cet outil est plus sûre mais le temps nécessaire est plus important.

Nous avons vérifié l'absence d'inter-blocage dans tous les modèles générés dans la partie précédente (les deux tableaux 4.1 et 4.2), ce qui garantit la fiabilité et l'extensibilité de notre architecture ReDy.

4.7.3 Absence de blocage operationnel (livelocks)

Le model checker utilisé (EVALUATOR 4.0) a détecté un blocage opérationel lors de la vérification de la propriété de vivacité ordre Detection Action. En effet, le model checker a retourné un contre exemple qui prouve qu'un inter-blocage existe. Ce contre exemple représente le problème classique d'une famine (starvation), *i.e.* un processus monopolise l'exécution et ne laisse pas les autres procesuss avancer. La figure 4.11 représente ce contre exemple. Nous remarquons que dans ce contre exemple l'unité de gouvernance d'indice 2 monopolise les communications et ne donne plus la main aux autres unités de gouvernance, ce qui amène le système global à un blocage operationnel.

Il est à noter que ce problème a été détecté dans une version antérieure du modèle LNT et qu'il a été corrigé dans le modèle actuel.

4.8 Conclusion

Dans ce chapitre, nous avons présenté le modèle formel de notre architecture ReDy en expliquant la démarche suivie pour effectuer cette modélisation. En effet, nous avons divisé notre système global en plusieurs niveaux de granularité : unité de détection, unité d'action, unité de gouvernance et sous-système unitaire. A partir de ces éléments modélisés en LNT, nous construisons notre modèle général de l'architecture ReDy. Nous avons réalisé plusieurs expérimentations pour extraire les caractéristiques des modèles présentés. Nous sommes ensuite passés à l'étape de validation formelle qui nous a permis de vérifier l'absence de blocage et d'interblocage de notre architecture ReDy. La vérification de ces propriétés nous garantit les critères de fiabilité et d'extensibilité de notre architecture.

Dans le chapitre suivant, nous allons décrire l'algorithme de brassage et sa modélisation

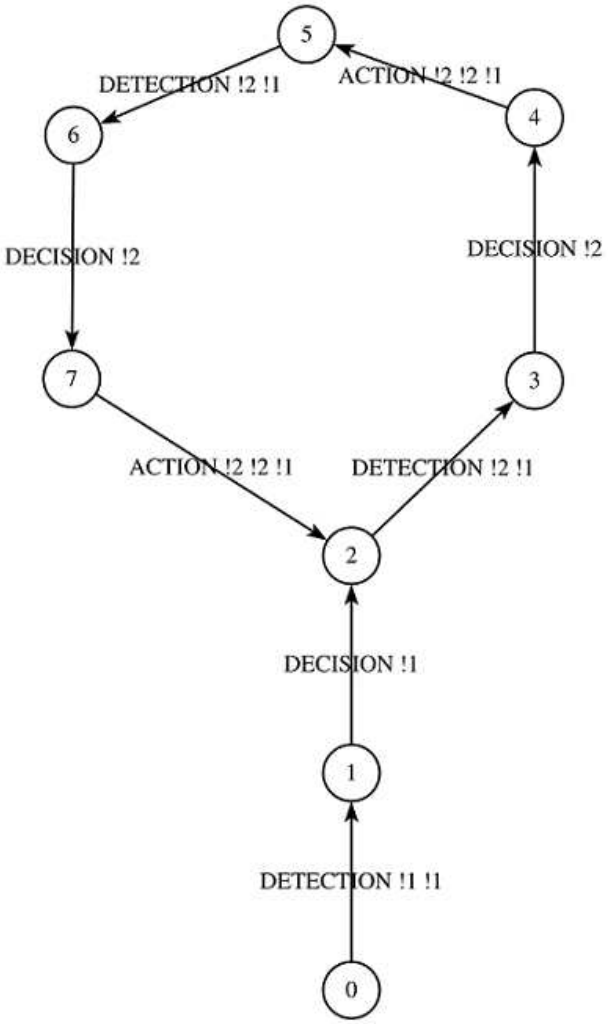


FIGURE 4.11 – Détection d'un inter-blocage-Problème de famine

formelle afin de garantir plus de fiabilité pour notre architecture ReDy mais surtout pour prouver l'aspect dynamique relatif à notre architecture.

Chapitre 5

Algorithme de brassage proposé dans l'architecture et sa formalisation

5.1 Introduction

Après avoir présenté l'architecture ReDy proposée ainsi que sa modélisation et sa validation formelle, nous allons passer dans ce chapitre à l'étude de la gestion de l'adhésion des composants au système. En effet, dans notre architecture ReDy, cet aspect est assuré en utilisant l'algorithme de brassage amélioré (enhanced shuffling algorithm). Il s'agit d'un algorithme proposé dans la littérature en langue naturelle et nous proposons dans ce chapitre sa formalisation en algorithmique afin de préparer sa modélisation formelle voir même son implémentation par la suite. Nous nous focalisons sur les éléments communiquants en mode distant, ce qui correspond aux unités de gouvernance. L'algorithme de brassage assure un mode décentralisé tel que chaque unité de gouvernance représente un noeud dans une communication point-à-point (peer to peer) avec les autres noeuds.

Dans l'algorithme de brassage, chaque noeud du système possède une liste de voisins (appelée vue partielle) et des échanges entre les listes sont effectués de façon périodique afin d'assurer l'aspect dynamique et fiable du système. Dans ce chapitre, nous présentons les types de données utilisés dans l'algorithme, puis nous modélisons une itération de brassage impliquant un noeud initiateur et un noeud récepteur de cette itération. Nous

exprimons le comportement de ces deux noeuds impliqués dans l'itération de brassage ainsi que le comportement de tout autre noeud non concerné par cet itération.

5.2 Présentation de l'Algorithme de Brassage

Dans l'algorithme de brassage renforcé, chaque noeud maintient une liste de c voisins. L'idée principale de cet algorithme est que les noeuds effectuent des échanges de leurs listes de voisins de façon périodique (opération de brassage). La liste échangée est de même longueur pour tous les noeuds. On appelle cette longueur *longueur de la liste de brassage* (SL pour Shuffle Length) qui est toujours plus petite que c (le nombre de voisins). Un autre paramètre très important qui est utilisé dans cet algorithme est l'âge des voisins. Pour un noeud donné de notre réseau, possédant une liste de voisins, on affecte à chaque voisin un nombre que l'on appelle *Age*. L'âge de tous les voisins est incrémenté par un lorsqu'on exécute une opération de brassage. Ce paramètre permet de connaitre le voisin le plus ancien dans la liste [VGVS05].

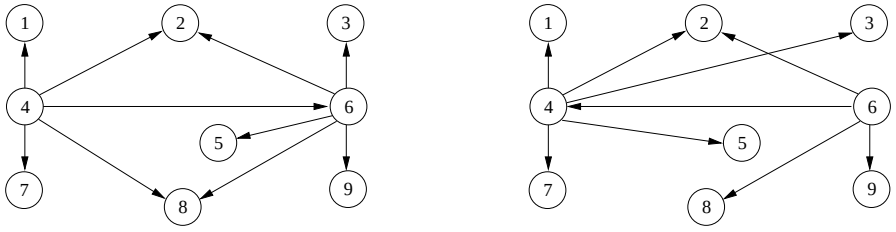
Dans l'algorithme de brassage, nous avons principalement deux noeuds interagissant entre eux : le noeud initiateur P qui initie le shuffling, et le noeud récepteur Q qui répond à la requête initiative de P. Le noeud initiateur P exécute les étapes suivantes :

1. Incrémenter par un l'âge de tous les voisins.
2. Sélectionner le voisin Q possédant l'âge le plus grand parmis tous les voisins. Sélectioner $SL - 1$ autres voisins de façon aléatoire.
3. Remplacer l'entrée de Q dans la liste préparée par une nouvelle entrée ayant l'âge 0 et l'adresse de P.
4. Envoyer le sous-ensemble de voisins préparé au noeud Q.
5. Recevoir de la part de Q un sous-ensemble dont les éléments ne dépassent pas SL voisins.
6. Ignorer les entrées ayant l'adresse de P et les entrées déja existants dans la liste de voisins de P.
7. Mettre à jour la liste de voisins de P pour inclure les entrées restantes, en commençants en premier par remplir les cases vides (s'il y'en a), et ensuite remplacer les entrées déjà envoyées à Q.

Concernant le noeud récepteur Q, son rôle est de répondre à la requête de P en envoyant

un sous-ensemble de voisins préparé aléatoirement de nombre ne dépassant pas SL. Ensuite, le noeud Q met à jour la liste de ses voisins en prenant en considération la liste envoyée par le noeud P. Il est à noter que le noeud Q n'incrémente pas le champs âge de ses voisins. Cette opération est exécutée uniquement par le noeud initiateur du brassage P.

Exemple : La figure 5.1 illustre une itération d'une opération de brassage. Le noeud 4 initie l'opération de brassage. Il choisit le noeud 6 car c'est le voisin le plus ancien (possédant l'âge le plus grand). La longueur de la liste des voisins est de 5 ($c = 5$) et la longueur de la liste de brassage est de 3 ($SL = 3$). Le noeud 4 envoie la liste $\{4, 2, 8\}$ au noeud 6, qui met à jour sa liste de voisins et renvoie au noeud 4 la liste $\{2, 5, 3\}$. Le noeud 4 met à jour sa liste de voisins. A la fin de cette opération, les deux noeuds 4 et 6 maintiennent une liste de voisins actualisée.



(a) Before shuffling (b) After shuffling

FIGURE 5.1 – Exemple d'une itération de l'algorithme de brassage

Joindre ou quitter le réseau : afin de rejoindre le réseau de noeuds, un nouveau noeud doit juste trouver un noeud appartenant au système et doit lui être lié. Le nouveau noeud rejoignant le réseau doit construire sa liste de voisins et doit être inclu dans les listes de voisins des autres noeuds du réseau. Ceci est réalisé grâce aux opérations de brassage périodiques.

Pour quitter le réseau de noeuds, il n'y a rien à faire : il suffit de quitter. Le système s'adapte et ignore le noeud qui a quitté quand ce dernier ne donnera pas de réponse. Cette caractéristique permet une grande résistance aux défaillances puisqu'un noeud qui est tombé en panne ne peut pas informer le système qu'il est en panne.

5.3 Formalisation de l'algorithme de brassage

5.3.1 Définition des variables et des types de données

Nous commençons par définir toutes les variables et les types de données dont nous avons besoin.

- *nb_neighbors* : un entier représentant le nombre de voisins d'un noeud donné.

- *SL* : un entier représentant le nombre d'éléments contenus dans la liste de brassage. Ce nombre est toujours le même dans toutes les itérations de brassage et il est toujours plus petit que le nombre de voisins ($SL < nb_neighbors$).

- *add_T* : un type de données utilisé pour définir le type *address*. Chaque noeud possède une adresse spécifique et unique qui est un entier vérifiant $1 \leq add_T \leq nb_neighbors$.

- *neighbor_T* : un type de données utilisé pour définir un voisin. Un voisin peut avoir la valeur *empty*, ce qui correspond à un voisin vide, ou bien il peut avoir une valeur non vide définie par une adresse et un âge. L'âge de tous les voisins est incrémenté par un à chaque itération de brassage.

- *neighbors_list* : l'ensemble des voisins d'un noeud.

- *shuffle_list* : un ensemble de voisins du noeud initiant l'opération de brassage contenus dans sa liste de voisins. Cette liste est envoyée par le noeud initiateur au noeud récepteur au cours d'une itération de brassage.

- $shuffle_list_Q$: un ensemble de voisins du noeud récepteur contenus dans sa liste de voisins. Cette liste est envoyée par le noeud récepteur au noeud initiateur dans une itération de brassage.

- $neighbor_Q$: le voisin le plus ancien dans la liste des voisins du noeud initiateur. add_Q et age_Q représentent l'adresse et l'âge de $neighbor_Q$.

- *Myadd* : élément de type *add_T*, utilisé pour définir l'adresse du noeud actuel qui est en cours de communication.

$nb_neighbors : int$
$SL : int$
$add_T : int\ where\ 1 \leq add_T \leq nb_neighbors$

$neighbor_T = (add : add_T, age : int)$
$neighbors_list = neighbor_1 : neighbor_T, \ldots, neighbor_{nb_neighbors} : neighbor_T$
$shuffle_list = neighbor_1, \ldots, neighbor_{SL} : neighbor_T$
$shuffle_list_Q = neighbor_{Q1}, \ldots, neighbor_{QSL} : neighbor_T$
$add_Q : add_T$
$age_Q : int$
$neighbor_Q : neighbor_T = (add_Q, age_Q)$
$myadd : add_T$

5.3.2 Formalisation d'une communication de brassage

Dans chaque itération de brassage, nous avons deux noeuds interagissant : le noeud initiateur et le noeud récepteur. Dans la suite, on note P le noeud initiateur et Q le noeud récepteur.

Comportement d'un noeud initiateur d'une communication de brassage

La première étape dans une itération de brassage est d'incrémenter par un l'âge de tous les voisins du noeud initiateur P. Pour ce faire, on définit une boucle *For* afin d'incrémenter le champs *Age* des voisins contenus dans *neighbors_list*.

for $(i = 1, i \leq nb_neighbors, i++)$
  $neighbors_list[i].age = neighbors_list[i].age + 1$
end for

La deuxième étape consiste à sélectionner un ensemble de voisins à partir de la liste des voisins, ce qui correspond à la liste de brassage qui va être échangée dans une itération de brassage. Nous commençons par sélectionner $neighbor_Q$ ayant l'âge le plus grand parmi tous les voisins de P. Ceci est réalisé en parcourant la liste des voisins *neighbors_list* du noeud initiateur P, cherchant l'âge le plus grand. $neighbor_Q$ correspond au noeud récepteur dans une itération de brassage. Ensuite, on sélectionne aléatoirement $SL - 1$ autres voisins que l'on met dans la liste de brassage *shuffle_list*. *Random_neighbor* est une fonction qui retourne un voisin choisi arbitrairement à partir de la liste *neighbors_list*.

$add_Q = neighbors_list[1].add$

for $(i = 1, i < nb_neighbors, i++)$
  if $neighbors_list[i+1].age > neighbors_list[i].age$
  then $add_Q = neighbors_list[i+1].add$;
    $age_Q = neighbors_list[i+1].age$;
  end if
end for
$neighbor_Q = (add_Q, age_Q)$

$shuffle_list[1] = neighbor_Q$
for $(i = 2, i \leq SL, i++)$
  $shuffle_list[i] = Random_neighbor$;
end for

La troisième étape consiste à remplacer le noeud Q dans *shuffle_list* par une entrée ayant l'âge 0 et l'adresse de P.

$shuffle_list[1].age = 0$;
$shuffle_list[1].add = Myadd$;

Ensuite, nous envoyons le sous-ensemble *shuffle_list* au noeud Q en utilisant la fonction *Send()*. Cette fonction possède trois paramètres : l'adresse du noeud expéditeur, une liste de voisins du noeud expéditeur et l'adresse du noeud récepteur.

$Send(Myadd, shuffle_list, add_Q)$

Le noeud P reçoit de la part de Q un sous-ensemble dont le nombre d'éléments ne dépasse pas SL éléments parmi les voisins de Q. Il s'agit de la liste $shuffle_list_Q$. On utilise la fonction *Receive()* qui possède deux paramètres : l'adresse du noeud expéditeur Q ainsi qu'une liste de voisins du noeud Q.

$Receive(shuffle_list_Q, add_Q)$

Lorsque le noeud P reçoit $shuffle_list_Q$, on vérifie s'il existe des noeuds dans cette liste possédant l'adresse de P, ou bien s'il existe des noeuds qui figurent déjà dans la liste *neighbors_list*. Ces noeuds sont alors négligés. Ensuite, on commence à mettre à jour la

liste *neighbors_list* de P afin d'inclure tous les noeuds restants dans la liste $shuffle_list_Q$ (à l'exeption des noeuds qui ont été négligés auparavant). On commence par utiliser les cases vides, ensuite on remplace les voisins qui ont été déjà envoyés au noeud Q (les noeuds contenus dans la liste *shuffle_list*)

for $(i = 1, i \leq SL, i++)$
  $neighbor_i = shuffle_list_Q[i]$
  if $neighbor \neq empty$
  and $(neighbor_i.add \neq Myadd)$
  and (**forall** $(j = 1, j \leq nb_neighbors, j++)$
    $neighbor_i.add \neq neighbors_list[j].add$
  end forall)
  then
    for $(j = 1, j \leq nb_neighbors, j++)$
      $neighbor_j = neighbors_list[j]$
      if $(neighbor_j = empty)$
      then $neighbor_j = neighbor_i$;
        $done = true$;
      end if
    end for
    if $(done = false)$
    then
      for $(j = 1, j \leq nb_neighbors, j++)$
        $neighbor_j = neighbors_list[j]$;
        if $(Is_in_shuffle_list(neighbor_j) = true)$
        then $neighbor_j = neighbor_i$;
          $Break$;
        end if
      end for
    end if
  end if
end for

On définit la fonction *Is_in_shuffle_list()* afin de vérifier si un noeud est inclus dans la liste *shuffle_list*. Cette fonction retourne un booléen selon le résultat obtenu.

define function $Is_in_shuffle_list(neighbor : neighbor_T)$ *return boolean is*
$Res : Bool$;
$Res = False$;
for $(1 \leq k \leq SL)$
$neighbor_k = shuffle_list[k]$
if $(neighbor == neighbor_k)$
then $Res = True$;
end if
break ;
end for
Return Res;
end define

Comportement d'un noeud récepteur d'une communication de brassage

Concernant l'algorithme du noeud récepteur Q, nous avons deux étapes à exécuter. Une fois le noeud récepteur Q reçoit la liste de brassage *shuffle_list* de la part du noeud P, le noeud Q exécute la première étape consistant à préparer une liste aléatoire d'éléments ne dépassant pas SL voisins, ensuite le noeud Q envoie cette liste au noeud P.

for $(i = 1, i \leq SL, i++)$
$shuffle_list_Q[i] = Random_Neighbor$;
end for
$Send(Myadd, shuffle_list_Q, add_P)$

Dans la deuxième étape, le noeud Q met à jour la liste de ses voisins en prenant en considération la liste envoyée par le noeud P. Cette étape est exécutée de même que pour le noeud initiateur P.

Comportement d'un noeud non concerné par la communication de brassage

Tous les noeuds restants du système sont des noeuds non concernés. Ce sont des noeuds passifs qui n'interviennent pas dans les échanges entre le noeud initiateur et le noeud récepteur.

5.4 Conclusion

Dans ce chapitre, nous avons présenté l'algorithme de brassage utilisé dans la gestion de l'adhésion des composants au système conçu selon l'architecture ReDy. Nous considérons principalement la communication entre les différentes unités de gouvernance des sous-systèmes (mode distant). Nous présentons ensuite la formalisation en algorithmique de cet algorithme en explicitant les différentes étapes d'exécution décrivant le comportement des noeuds participant dans une itération de brassage. L'utilisation de l'algorithme de brassage assure l'aspect dynamique du système car il permet d'ajouter et d'enlever des noeuds du système au cours de son fonctionnement. Cette formalisation va servir dans le chapitre suivant pour présenter la modélisation formelle de cet algorithme afin de prouver le dynamise des systèmes ReDy mais aussi afin de renforcer leur fiabilité.

Chapitre 6

Modélisation et analyse formelle de l'algorithme de brassage proposé

6.1 Introduction

Dans ce chapitre, nous proposons une modélisation formelle de la formalisation de l'algorithme de brassage. Nous proposons d'implémenter cet algorithme dans notre modèle formel présenté dans le chapitre 4. Ensuite, nous validons formellement notre modèle afin de prouver l'absence d'inter-blocages et de blocages opérationnels. Après nous présentons la modélisation formelle de deux scénarios intéressants : le scénario d'ajout d'un noeud au système et le scénario de suppression d'un noeud du système. La validation formelle de ces deux scénarios nous permet de prouver l'aspect dynamique de notre architecture ReDy et de renforcer sa fiabilité. Il est à noter que la validation formelle est limitée par le problème d'explosion de l'espace des états, c'est pourquoi nous nous focalisons sur la validation de comportements compliqués tels que l'algorithme de brassage dans notre cas ainsi que les deux scénarios d'ajout et de suppression. Nous commençons par éliminer tous les détails non pertinents par rapport au comportement que nous voulons valider.

6.2 Modélisation formelle de l'agorithme de brassage

Afin de modéliser l'algorithme de brassage, nous proposons un modèle réduit comportant uniquement les unités de gouvernance. Ce modèle est composé de trois modules principaux : le module *main*, le module *types* et le module *governance_unit*. Dans la suite nous allons définir ces trois modules.

6.2.1 Modélisation des types de données relatifs à l'algorithme de brassage

Dans le module *types*, on définit de nouveaux types relatifs à l'algorithme de brassage dont on aura besoin dans notre modèle. Dans ce module, on définit cinq types :

- *INDEX_SUB_SYSTEM* : un entier dont la valeur est comprise entre 1 et 4. Ce type exprime l'indice de l'unité de gouvernance (qui est le même que l'indice du sous-système auquel appartient l'unité de gouvernance) qui est unique pour chaque unité de gouvernance. Pour ce type, on peut utiliser les fonctions prédéfinies d'égalité, d'inégalité, de superiorité et de strict superiorité.

```
type INDEX_SUB_SYSTEM is
   range 1 .. 4 of Nat
   with "==","<>","<=", "<"
end type
```

- *TAB_IS_NEIGHBOR* : un tableau de booléens définissant, pour chaque noeud, si les autres noeuds sont des voisins ou non. La longueur du tableau correspond au nombre total des noeuds dans le système. Dans notre cas, le nombre total de noeuds dans le système est de quatre.

```
type TAB_IS_NEIGHBOR is
   array [1 .. 4 (*nb of nodes*)] of BOOL
end type
```

- *NEIGHBOR_T* : un constructeur utilisé pour définir le type *neighbor*. Un voisin peut être soit de valeur vide (*empty_neighbor*) ou bien un couple de deux champs. Le premier

champ est l'indice du noeud (*INDEX_SUB_SYSTEM*). Le deuxième champ est un entier indiquant l'âge du noeud (*Age*). Pour ce type, on peut utiliser quatre fonctions prédéfinies : deux fonctions pour accéder aux champs de *NEIGHBOR_T* qui sont la fonction *get* pour la lecture et la fonction *set* pour la modification, en plus de deux autres fonctions permettant de comparer deux voisins qui sont la fonction d'égalité et la fonction d'inégalité.

```
type NEIGHBOR_T is
   empty_neighbor,
   NEIGHBOR(ind:INDEX_SUB_SYSTEM,age:NAT)
   with "get", "set", "==", "<>"
end type
```

- *NEIGHBORS_LIST_T* : un constructeur utilisé pour définir la liste des voisins. Il contient des éléments de type *NEIGHBOR_T*. La longueur de cette liste est la même pour tous les noeuds (*nb_neighbors*, qui est une variable définie et initialisée dans le module *main*). Dans notre cas le nombre de voisins est de deux.

```
type NEIGHBORS_LIST_T is
   array [1 .. 2 (*nb_neighbors*)] of NEIGHBOR_T
end type
```

- *SHUFFLE_LIST_T* : un constructeur utilisé pour définir la liste de brassage. Il contient des éléments de type *NEIGHBOR_T*. La longueur de cette liste est *SL*, qui est une variable définie et initialisée dans le module *main*. Dans notre cas, la longueur de la liste de brassage est de deux.

```
type SHUFFLE_LIST_T is
   array [1 .. 2 (*SL*)] of NEIGHBOR_T
end type
```

Dans ce module, nous définissons également les fonctions que nous allons utiliser dans la modélisation. Nous définissons la fonction *IS_NOT_IN_SHUFFLE_LIST* qui prend en paramètre un noeud *NT* et une liste de noeuds *SList* et vérifie si ce noeud appartient

à la liste ou pas. Si le noeud n'est pas dans la liste, alors la fonction retourne la valeur True. Sinon, si le noeud appartient à la liste des noeuds, alors la fonction retourne la valeur False.

```
function IS_NOT_IN_SHUFFLE_LIST (NT : NEIGHBOR_T , SList : SHUFFLE_LIST_T,
                                  SL: NAT) : BOOL is
  var index_SSL : NAT in
   index_SSL:=0;
   loop loopSL in
      index_SSL:=index_SSL+1;
      if(NT<>empty_neighbor) then
         if not (SList[index_SSL]==empty_neighbor) then
            return not (SList[index_SSL].ind==NT.ind)
         end if
      end if;
      if(index_SSL >= SL) then
         break loopSL
      end if
   end loop;
   return true
  end var
end function
```

6.2.2 Modélisation de l'architecture globale du système utilisé pour la validation de l'algorithme de brassage

Dans le module *main*, on définit le process *MAIN* qui est paramétré par deux portes : la porte *MEMBERSHIP_INIT* et la porte *SHUFFLING_TRANSFER*. Dans le corps du processus, on commence par définir deux variables : la variable *nb_neighbors* indiquant le nombre de voisins que chaque noeud ne peut dépasser, et la variable *SL* qui indique la longueur de la liste de brassage de chaque noeud. Ces deux variables sont initialisées dans le processus *main* et restent les mêmes durant l'exécution de notre programme. Dans notre cas, nous supposons que le nombre de voisins pour chaque noeud est de deux (nb_neighbors := 2) et la longueur de la liste de brassage est de deux (SL := 2).

Ensuite, on définit une composition parallèle entre quatre processus instanciés. L'ensemble global de synchronisation est composé de deux portes : la porte *MEMBERSHIP_INIT* et la porte *SHUFFLING_TRANSFER*. Les quatre processus dans la composition parallèle représentent les unités de gouvernance composant notre système distribué. Ces processus communiquent via les portes *MEMBERSHIP_INIT* et *SHUFFLING_TRANSFER*. Ces processus possèdent deux paramètres communs *nb_neighbors* et *SL*, et un seul paramètre spécifique à chaque processus qui est *index_sub_system*. Dans notre cas, nous avons quatre unités de gouvernance communiquantes entre elles, chacune est instanciée par un indice spécifique *index_sub_system* de valeur comprise entre 1 et 4.

```
process MAIN [MEMBERSHIP_INIT, SHUFFLING_TRANSFER : any]
is
   var nb_neighbors : NAT, -- Number of neighbors
       SL : NAT            -- Shuffle length
   in
      nb_neighbors := 2;
      SL := 2;
      par MEMBERSHIP_INIT, SHUFFLING_TRANSFER in
         gouvernance_unit[MEMBERSHIP_INIT, SHUFFLING_TRANSFER]
(index_sub_system(1), nb_neighbors, SL)
      ||
         gouvernance_unit[MEMBERSHIP_INIT, SHUFFLING_TRANSFER]
(index_sub_system(2), nb_neighbors, SL)
      ||
         gouvernance_unit[MEMBERSHIP_INIT, SHUFFLING_TRANSFER]
(index_sub_system(3), nb_neighbors, SL)
      ||
         gouvernance_unit[MEMBERSHIP_INIT, SHUFFLING_TRANSFER]
(index_sub_system(4), nb_neighbors, SL)
      end par
   end var
end process
```

6.2.3 Modélisation d'une unité de gouvernance opérant dans l'algorithme de brassage

Dans le module *governance_unit*, on déclare deux processus : le processus *governance_unit* et le processus *membership_management*.

Le processus *governance_unit* est un processus d'initialisation. Il est exécuté une seule fois quand le programme est lancé. Dans ce processus, on définit les connections initiales entre les voisins. Dans notre cas, les connections résultant du processus d'initialisation sont présentées dans la figure 6.1. Un choix non déterministe est utilisé afin de permettre à chaque unité de gouvernance de définir sa liste de voisins.

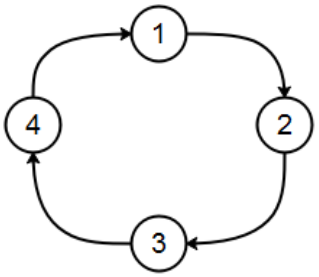


FIGURE 6.1 – Initialisation des noeuds du système

A la fin du choix non déterministe, on appelle le processus *membership_management* qui représente le comportement d'un noeud impliqué dans l'algorithme de brassage.

```
process gouvernance_unit[MEMBERSHIP_INIT, SHUFFLING_TRANSFER, sync : any]
   (id_sub:INDEX_SUB_SYSTEM, nb_neighbors:NAT,SL:NAT)
is
   var neighbors_list : NEIGHBORS_LIST_T,
       node_is_neighbor: TAB_IS_NEIGHBOR
   in
      node_is_neighbor:= TAB_IS_NEIGHBOR(false);
      neighbors_list := NEIGHBORS_LIST_T(empty_neighbor);

      select
         only if (id_sub==index_sub_system(1)) then
            neighbors_list[1]:=NEIGHBOR(index_sub_system(2),0);
            -- pour la GU d'indice 1 on definit la GU d'indice 2 comme voisin
```

```
            node_is_neighbor[2]:=true
         end if
      []
         only if (id_sub==index_sub_system(2)) then
            neighbors_list[1]:=NEIGHBOR(index_sub_system(3),0);
            -- pour la GU d'indice 2 on definit la GU d'indice 3 comme voisin
            node_is_neighbor[3]:=true
         end if
      []
         only if (id_sub==index_sub_system(3)) then
            neighbors_list[1]:=NEIGHBOR(index_sub_system(4),0);
          -- pour la GU d'indice 3 on definit la GU d'indice 4 comme voisin
            node_is_neighbor[4]:=true
         end if
      []
         only if (id_sub==index_sub_system(4)) then
            neighbors_list[1]:=NEIGHBOR(index_sub_system(1),0);
            -- pour la GU d'indice 4 on definit la GU d'indice 1 comme voisin
            node_is_neighbor[1]:=true
         end if
      end select;

      membership_management[MEMBERSHIP_INIT, SHUFFLING_TRANSFER, sync]
         (id_sub, id_sub, nb_neighbors,SL,neighbors_list,node_is_neighbor)
   end var
end process
```

Le processus *membership_management* est paramétré par deux portes de communication : *MEMBERSHIP_INIT* et *SHUFFLING_TRANSFER*. Ce processus possède cinq paramètres formels : *id_sub* et *id_sub_other* définissent deux indices d'unités de gouvernance, *nb_neighbors* définit le nombre de voisins, *SL* définit la longueur de la liste de brassage, *neighbors_list* définit la liste des voisins de l'unité de gouvernance d'indice *id_sub*, *node_is_neighbor* un tableau de booléens pour vérifier si les unités de gouvernance du système sont des voisins de l'unité de gouvernance d'indice *id_sub*.

Ensuite, nous définissons les variables du système : *index_q* l'indice du voisin le plus ancien, *index_1* et *index_2* des indices utilisés pour identifier d'autres unités de gouvernance, *q_age* l'âge du voisin le plus ancien, *shuffle_list* la liste de voisins envoyée pendant une opération de brassage, *q_shuffle_list* la liste de voisins reçue durant une opération de brassage, *index_NL* un entier qui parcourt la liste des voisins, *index_SL* un entier qui parcourt la liste de brassage, *index_SLQ* un entier qui parcourt la liste de brassage reçue. Dans la suite, nous expliquons la modélisation de l'algorithme de brassage étape par étape.

```
process membership_management[MEMBERSHIP_INIT, SHUFFLING_TRANSFER:any]
   (id_sub, id_sub_other:index_sub_system,
    nb_neighbors:NAT,SL:NAT,
    neighbors_list : NEIGHBORS_LIST_T,
    node_is_neighbor: TAB_IS_NEIGHBOR)
is
    var
   index_q, index_1, index_2 : index_sub_system,
       q_age : NAT,
       q_shuffle_list : SHUFFLE_LIST_T,
       shuffle_list : SHUFFLE_LIST_T,
 index_NL, index_SL, index_SLQ : NAT,
       find_empty : bool

<<membership management code here>>

end process
```

Dans une opération de brassage, une unité de gouvernance est soit un noeud initiateur, soit un noeud récepteur, soit un noeud non concerné. Ce comportement est modélisé par un choix non déterministe avec trois cas. Dans la suite, nous explicitons ces trois cas séparémment. On note P le noeud initiateur et Q le noeud récepteur.

Modélisation du comportement d'un noeud initiateur d'une itérationn de brassage

La première étape de l'algorithme de brassage est d'incrémenter par un l'âge de tous les voisins dans la liste *neighbors_list* du noeud initiateur. Ce comportement est exprimé dans la boucle *NL*.

Dans la même boucle, on vérifie à chaque itération l'âge du noeud en cours afin de trouver le voisin le plus ancien. De cette manière, on réduit la complexité de notre algorithme.

```
index_NL:=0;
index_q := index_sub_system(1);
q_age := 0;
loop NL in
   index_NL:=index_NL+1; -- increment the index_NL
   if( neighbors_list[index_NL]<> empty_neighbor) then
-- increase the age of the "index_NL"th neighbor by 1
      neighbors_list[index_NL]:=
      neighbors_list[index_NL].{age=>neighbors_list[index_NL].age+1};

      if(neighbors_list[index_NL].age > q_age) then
-- select the (first) oldest neighbor
         index_q := neighbors_list[index_NL].ind;
         q_age := neighbors_list[index_NL].age
      end if
   end if;
   if(index_NL >= nb_neighbors) then
      break NL
   end if
end loop;
```

Ensuite, nous allons exécuter un rendez-vous entre tous les noeuds du système afin de définir le rôle de chaque noeud dans l'itération de brassage en cours. Le noeud d'indice *id_sub* est initiateur, le noeud d'indice *index_q* est récepteur et tout autre noeud d'indice différent de *id_sub* et *index_q* est un noeud non concerné par l'itération de brassage.

```
MEMBERSHIP_INIT("INIT", id_sub, "REC", index_q);
```

Ensuite, on commence à remplir la liste de brassage. Dans la première case, on met le noeud P avec l'age 0.

```
shuffle_list[1]:= NEIGHBOR(id_sub,0);
```

Ensuite, on remplit les autres cases de la liste de brassage de façon aléatoire. Pour réaliser cet objectif, nous définissons deux boucles SL et NLL. Tout d'abord, le compteur *index_SL* parcourt les cases de la liste de brassage *shuffle_list* en utilisant la boucle SL. A l'intérieur de cette boucle nous définissons la boucle NLL afin de parcourir les cases de la liste des voisins *neighbors_list* en utilisant le compteur *index_NL*. Avant de remplir une case dans la liste de brassage, nous avons trois conditions à remplir. Premièrement, il faut vérifier que la case d'indice *index_NL* dans la lise des voisins est non vide. Deuxièmement, il faut vérifier que l'indice du noeud de cette case est différent de l'indice du noeud le plus âgé (le récepteur de l'itération de brassage). Troisièmement, il faut vérifier que la case d'indice *index_SL* dans la liste de brassage est vide. Si les trois conditions sont vérifiées, alors nous avons un choix non déterministe entre deux branches : dans la première branche, la case d'indice *index_SL* prend en valeur le noeud dans la case d'indice *ind_NL* de la liste des voisins, dans la deuxième branche, on ne fait rien (null). Grâce à ce choix non déterministe, nous assurons un remplissage aléatoire de la liste de brassage.

A la fin de ces deux boucles, la liste de brassage est bien préparée et elle est prête à être envoyée au noeud récepteur.

```
index_SL:=1;
index_NL:=0;
   loop SL in
      index_SL:=index_SL+1;
      loop NLL in
         index_NL:=index_NL+1;
         if( neighbors_list[index_NL]<> empty_neighbor) then
            if( neighbors_list[index_NL].ind <> index_q) then
```

```
            if (shuffle_list[index_SL]==empty_neighbor) then
               shuffle_list[index_SL]:= neighbors_list[index_NL] --pk
            else
               select
                  shuffle_list[index_SL]:= neighbors_list[index_NL]
               []
                  null
               end select
            end if
         end if
      end if;
      if(index_NL >= nb_neighbors) then
         break NLL
      end if
   end loop;
   if(index_SL >= SL) then
      break SL
   end if
end loop;
```

Une fois la liste *shuffle_list* est prête, nous effectuons une communication par rendez-vous sur la porte *SHUFFLING_TRANSFER* entre les noeuds du système. Durant cette communication, quatre paramètres sont échangés : *id_sub* l'indice du noeud initiateur P, *shuffle_list* la liste de brassage préparée par P et envoyée de P à Q, *index_q* l'indice du noeud récepteur Q qui est envoyé de P à Q, *q_shuffle_list* la liste de brassage de Q qui est envoyée de Q à P.

```
SHUFFLING_TRANSFER("INIT", id_sub, shuffle_list,
                   "REC", index_q, ?q_shuffle_list);
```

Une fois la liste *q_shuffle_list* est reçue par le noeud Q, il faut modifier l'âge du noeud Q dans la liste des voisins de P en lui donnant la valeur 0.

```
index_NL:=0;
      loop NL in
```

```
         index_NL:=index_NL+1;
         if ( (neighbors_list[index_NL]<>empty_neighbor)
       and (neighbors_list[index_NL].ind==index_q)) then
            neighbors_list[index_NL]:=neighbors_list[index_NL].{age=>0}
-- set the age of Q to 0 in neighbors_list
         end if;
         if(index_NL >= nb_neighbors) then
            break NL
         end if
      end loop;
```

Ensuite, on commence la mise à jour de la liste *neighbors_list* en utilisant la liste reçue *q_shuffle_list*. On néglige les noeuds ayant l'indice du noeud initiateur P ainsi que les noeuds déjà contenus dans la liste *neighbors_list* du noeud P. Ensuite, on commence par remplir les cases vides (*empty_neighbor*), puis on remplace les noeuds déjà envoyés à Q dans la liste *shuffle_list*.

Pour réaliser cette mise à jour, on définit tout d'abords la boucle *SLQ* qui parcourt les cases de la liste *q_shuffle_list* par le compteur *index_SLQ*. On commence par vérifier si l'élément sélectionné est non vide. Ensuite, on vérifie deux conditions : que l'élément séléctionné est différent du noeud P et qu'il ne fait pas partie de la liste des voisins de P. Une fois ces conditions vérifées, nous avons deux possibilités :

- Remplir les cases vides dans la liste des voisins de P : Pour ce faire, on définit une boucle NL qui parcourt la liste des voisins *neighbors_list* par le compteur *index_NL*. S'il existe une case vide, alors on met dedans le noeud d'indice *index_SLQ*. S'il n'y a pas de case vide, on passe à la boucle suivante (*NL2*).

- Remplacer les cases des noeuds déjà envoyés dans la liste de brassage *shuffle_list* : Pour ce faire, on définit une boucle *NL2* qui parcourt la liste des voisins par le compteur *index_NL*, et à l'intérieur de cette boucle on définit une autre boucle SL pour parcourir la liste de brassage *shuffle_list* par le compteur *index_SL*.

```
index_SLQ:=0;
loop SLQ in -- parcourir la liste q_shuffle_list  recu
   index_SLQ:=index_SLQ+1; -- increment the index_SL
   if(q_shuffle_list[index_SLQ]<> empty_neighbor) then
```

```
if(q_shuffle_list[index_SLQ].ind<>id_sub)
and (node_is_neighbor[NAT(q_shuffle_list[index_SLQ].ind)]==false) then
  find_empty:=false;
  index_NL:=0;
  loop NL in -- parcourir la liste neighbors_list
      index_NL:=index_NL+1; -- increment the index_NL
      if(neighbors_list[index_NL]==empty_neighbor) then
         -- si emplacement vide mettre le noeud selectionne
         neighbors_list[index_NL]:=q_shuffle_list[index_SLQ];
         node_is_neighbor[NAT(q_shuffle_list[index_SLQ].ind)]:=true;
         find_empty:=true;
         break NL
      end if;
      if(index_NL >= nb_neighbors) then
         break NL
      end if
   end loop;
   if(find_empty==false) then
    index_NL:=0;
    loop NL2 in -- parcourir la liste neighbors_list
    index_NL:=index_NL+1; -- increment the index_NL
       index_SL:=1;
       loop SL2 in
          index_SL:=index_SL+1; -- increment the index_SL
          if(neighbors_list[index_NL]==shuffle_list[index_SL]) then
           neighbors_list[index_NL]:=q_shuffle_list[index_SLQ];
             node_is_neighbor[NAT(shuffle_list[index_SL].ind)]:=false;
             node_is_neighbor[NAT(q_shuffle_list[index_SLQ].ind)]:=true;
             break SL2
          end if;
          if(index_SL >= SL) then
             break SL2
          end if
       end loop;
       if(index_NL >= nb_neighbors) then
          break NL2
```

```
            end if
         end loop
       end if
      end if
   end if;
   if(index_SLQ >= SL) then
      break SLQ
   end if
end loop
```

Modélisation du comportement d'un noeud récepteur dans l'itération de brassage

Si le noeud en question est un noeud récepteur, on commence par un rendez-vous sur la porte *MEMBERSHIP_INIT*. Durant cette communication, nous avons deux données échangées : le paramètre *index_1* qui est un paramètre reçu de la part du noeud initiateur exprimant son indice, et le paramètre *id_sub* est l'indice du noeud récepteur qui est envoyé aux autres noeuds. Cette communication s'effectue uniquement si *index_1* est différent de *my_id_GU*.

```
MEMBERSHIP_INIT("INIT", ?index_1,"REC", id_sub) where (index_1 <> id_sub);
```

Ensuite, le noeud récepteur prépare une liste de brassage de façon aléatoire à partir de sa liste de voisins. Cette liste est préparée de la même façon que le noeud initiateur prépare sa liste de brassage.

Une fois la liste *shuffle_list* est prête, on effectue une communication par rendez-vous sur la porte *SHUFFLING_TRANSFER* entre les unités de gouvernance du système. Quatre paramètres sont échangés durant cette communication : *index_1* l'indice du noeud initiateur P envoyé de P à Q, *q_shuffle_list* la liste de brassage de P envoyée de P à Q, *id_sub* l'indice du noeud récepteur Q envoyé de Q à P, *shuffle_list* la liste de brassage de Q envoyée de Q à P.

```
SHUFFLING_TRANSFER("INIT", ?index_1, ?q_shuffle_list, "REC", id_sub, shuffle_list)
                        where (index_1<>id_sub);
```

Le noeud récepteur met à jour la liste de ses voisins en prenant en considération la liste reçue de la part du noeud initiateur. Cette mise à jour est exécutée de la même manière que celle qui a été faite dans le noeud initiateur.

Modélisation du comportement d'un noeud non concerné par l'itération de brassage

Ce noeud est un noeud passif. Tous les noeuds restants du système (à l'exeption du noeud initiateur et du noeud récepteur) sont des noeuds non concernés. Un tel noeud communique sur les deux portes de communication *MEMBERSHIP_INIT* et *SHUFFLING_TRANSFER*. Les noeuds non concernés ont une connaissance sur la communication qui s'effectue ainsi que le transfert de données réalisé sans faire part de ce transfert.

```
MEMBERSHIP_INIT("INIT", ?index_1, "REC", ?index_2)
      where (index_1 <> id_sub) and (index_2 <> id_sub) ;

SHUFFLING_TRANSFER( "INIT", ?index_1, ?any SHUFFLE_LIST_T,
                    "REC", ?index_2, ?any SHUFFLE_LIST_T)
      where (index_1<>id_sub) and (index_2<>id_sub)
```

6.3 Modélisation du scénario d'ajout d'un noeud au système

Dans cette partie, nous allons modéliser le scénario d'ajout dynamique d'un noeud au système. Pour ce faire, nous proposons deux modèles :

- Dans le premier modèle, nous montrons que pour ajouter dynamiquement un noeud au système, il suffit de lui définir un seul voisin qui est déja actif dans le système.

- Dans le deuxième modèle, nous montrons que seule la définition du noeud ne suffit pas pour le faire intègrer au système.

Dans la suite nous allons nous focaliser sur le premier modèle.

6.3.1 Modèle formel du scénario d'ajout

Dans le premier modèle, on considère un système initialisé avec trois noeuds. L'objectif est de modéliser l'ajout d'un quatrième noeud au système et de prouver qu'il s'intègre bien au système, et que le comportement correct du système est maintenu. Pour répondre à cet objectif, nous apportons des modifications au modèle précédent. Tout d'abord, nous modifions l'initialisation des noeuds du système, la nouvelle itnitialisation en LNT est définie dans le processus *gouvernance_unit*.

```
 select
   only if (id_sub==index_sub_system(1)) then
      neighbors_list[1]:=NEIGHBOR(index_sub_system(2),0);
      -- Noeud 2 est voisin du noeud 1
      node_is_neighbor[2]:=true
   end if
[]
   only if (id_sub==index_sub_system(2)) then
      neighbors_list[1]:=NEIGHBOR(index_sub_system(3),0);
      -- Noeud 3 est voisin du noeud 2
      node_is_neighbor[3]:=true
   end if
[]
   only if (id_sub==index_sub_system(3)) then
      neighbors_list[1]:=NEIGHBOR(index_sub_system(1),0);
    -- Noeud 1 est voisin du noeud 3
      node_is_neighbor[1]:=true
   end if
[]
   only if (id_sub==index_sub_system(4)) then
        null
   end if
end select;
```

La figure 6.2 illustre l'initialisation des noeuds dans ce scénario d'ajout d'un noeud au système.

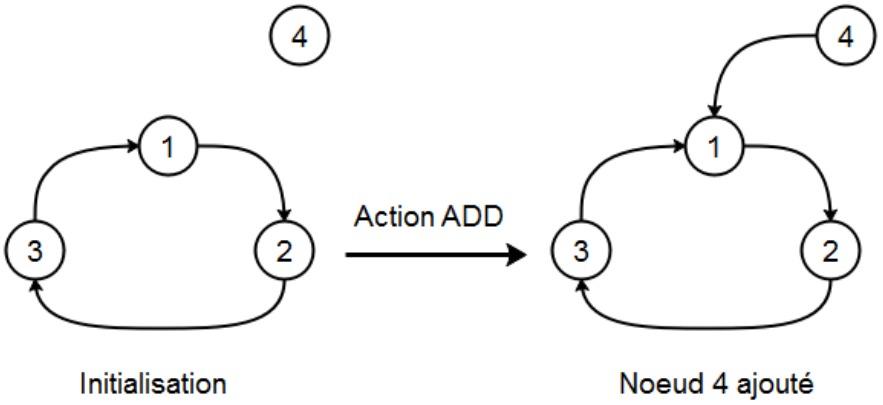


FIGURE 6.2 – Initialisation des noeuds du système lors du scénario d'ajout

Ensuite nous définissons une nouvelle porte de communication ADD qui s'ajoute aux paramètres de tous les processus.

```
process MAIN [MEMBERSHIP_INIT, SHUFFLING_TRANSFER, ADD : any]

process gouvernance_unit[MEMBERSHIP_INIT, SHUFFLING_TRANSFER, ADD : any]

process membership_management[MEMBERSHIP_INIT, SHUFFLING_TRANSFER, ADD:any]
```

Ensuite, nous apportons des modifications au processus *membership_management*. Nous définissons une variable *ajout* de type booléen qui est initialisée à faux est qui prend la valeur vrai lorsque l'ajout est effectué. Après nous ajoutons une branche dans le choix non déterministe défini au corps du processus *membership_management*. Cette branche donne la possibilité d'ajouter un quatrième noeud au système et d'assurer sa bonne intégration tout en maintenant le fonctionnement global du système.

```
if(ajout) then
       ADD("Ajouter 4");
       if (id_sub==index_sub_system(4)) then
            neighbors_list[1]:=NEIGHBOR(index_sub_system(1),0);
            -- Le noeud 1 est voisin du noeud 4
            node_is_neighbor[1]:=true;
            ajout:=false
       end if
```

```
end if
```

6.3.2 Génération de scénarios d'exécution

La figure 6.3 représente le LTS de quelques scénarios d'exécution du modèle d'jout d'un noeud au système. Pour obtenir ce graphe, nous utilisons l'outil OCIS qui permet la navigation pas à pas afin d'explorer différents scénarios d'exécution. Ensuite le résultat de cette navigation est enregistré en fichier BCG puis en fichier PS.

6.4 Modélisation du scénario de suppression d'un noeud du système

Dans cette partie, nous proposons une modélisation formelle du scénario de suppression d'un noeud du système. Pour ce faire, nous définissons un système composé initialement de quatre noeuds (voir figure 6.4).

Ensuite nous définnisons une nouvelle porte de communication *DELETE* qui intervient comme paramètre dans les processus *MAIN*, *gouvernance_unit* et *membership_management*.

```
process MAIN [MEMBERSHIP_INIT, SHUFFLING_TRANSFER, DELETE : any]

process gouvernance_unit[MEMBERSHIP_INIT, SHUFFLING_TRANSFER, DELETE : any]

process membership_management[MEMBERSHIP_INIT, SHUFFLING_TRANSFER, DELETE:any]
```

Ensuite, nous apportons des modifications au processus *membership_management*. Nous définissons une variable *delete* de type booléen qui est initialisée à vrai et qui prend la valeur faux lorsque la suppression est effectuée. Après nous ajoutons deux branches dans le choix non déterministe défini au corps du processus *membership_management*. La première branche concerne le noeud concerné par la suppression : sa liste de voisin est vidée. La deuxième branche concerne les autres noeuds du système : ils ne prennent plus en considération le noeud supprimé. Ces deux branches donnent la possibilité de supprimer un noeud du système et d'assurer que le fonctionnement global du système est bien maintenu.

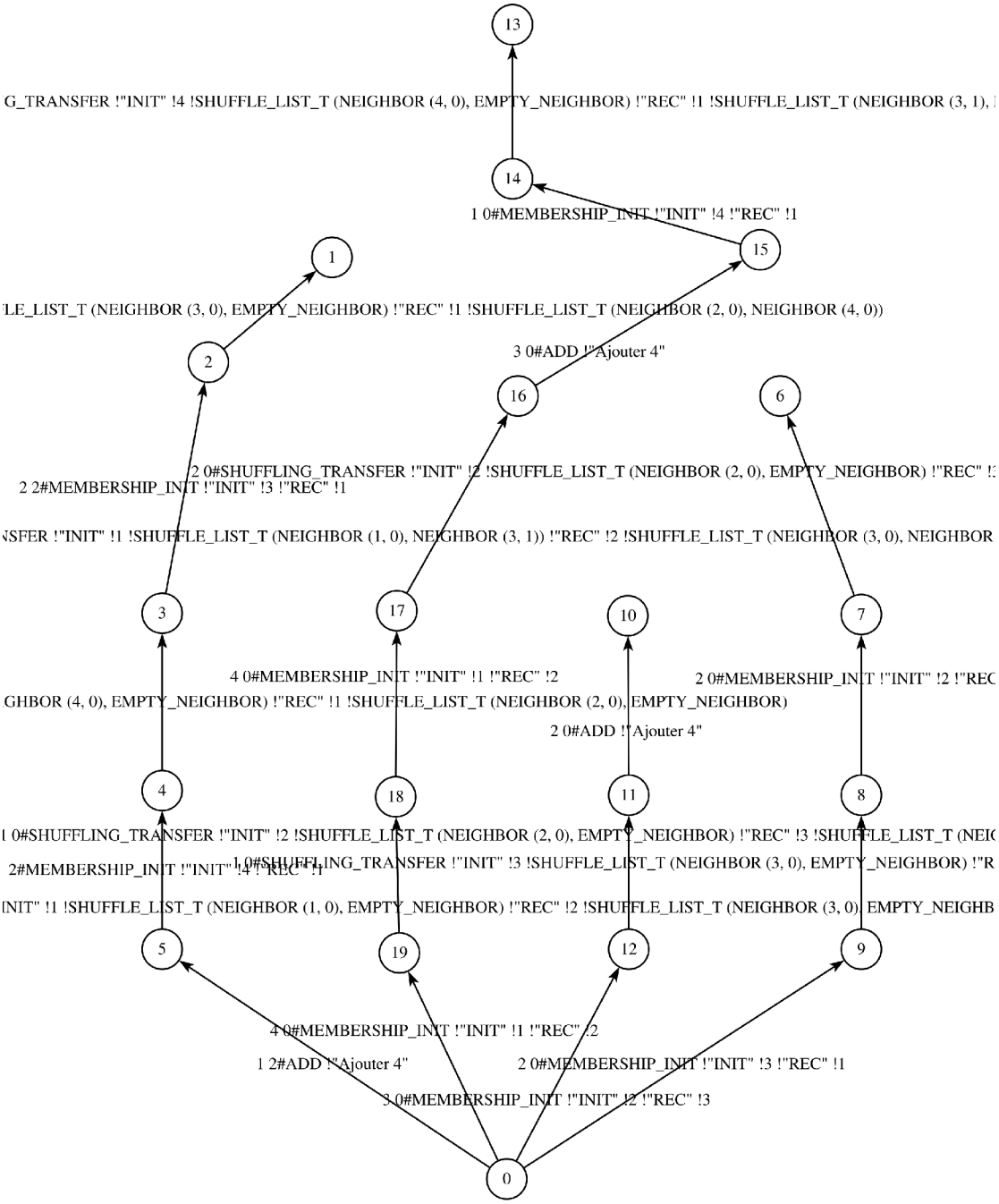


FIGURE 6.3 – LTS de quelques scénarios d'exécution du modèle d'ajout d'un noeud au système

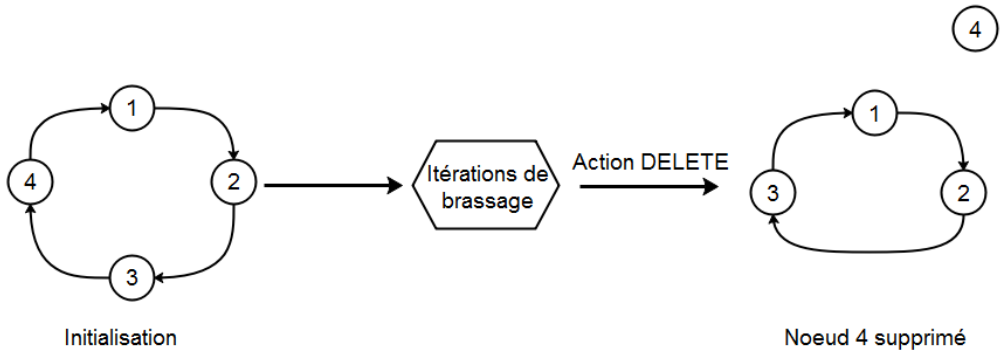


FIGURE 6.4 – Scénario de suppression d'un noeud du système

```
      only if (delete==true) and (id_sub==index_sub_system(4)) then
         DELETE("supprimer 4");
         node_is_neighbor:= TAB_IS_NEIGHBOR(false);
         neighbors_list := NEIGHBORS_LIST_T(empty_neighbor);
         delete:=false
      end if
[]
      only if (delete==true) and (id_sub<>index_sub_system(4)) then
         DELETE("supprimer 4");
         index_NL:=0;
         loop DEL in
            index_NL:=index_NL+1; -- increment the index_NL
            if( neighbors_list[index_NL]<>empty_neighbor) then
               if( neighbors_list[index_NL].ind==index_sub_system(4)) then
                   node_is_neighbor[4]:=false;
                   neighbors_list[index_NL]:=empty_neighbor
               end if
            end if;
            if(index_NL >= nb_neighbors) then
               break DEL
            end if
         end loop;
         delete:=false
      end if
```

6.5 Conclusion

Dans ce chapitre, nous avons modélisé formellement la gestion de l'adhésion des composants au système en utilisant l'algorithme de brassage avec le langage de spécification formelle LNT. En effet, la modélisation concerne un système composé de quatre noeuds et nous nous intéressons au comportement des différents noeuds du système lors d'une itération de brassage. Dans une itération, un noeud peut être soit initiateur, soit récepteur, soit non concerné par l'itération de brassage. Puis nous avons validé formellement les sénarios utiles d'ajout et de suppression dynamique d'un sous-système qui est représenté dans cette partie par son unité de gouvernance. La communication entre les différents noeuds s'effectue par rendez-vous sur des portes de communication définies. Nous avons validé des propriétés d'absence d'inter-blocage qui prouvent que le système ne se bloque pas. Le travail réalisé dans ce chapitre permet de prouver l'aspect dynamique de notre architecture ReDy et aussi de renforcer sa fiabilité.

Chapitre 7

Application de notre architecture ReDy au Smart Grids

7.1 Introduction

Après avoir présenté dans les chapitres précédents notre approche de modélisation formelle de l'architecture ReDy proposée destinée aux applications IoT, nous allons à présent appliquer notre approche dans le cas d'étude des Smart Grids. Dans ce chapitre, nous allons commencer par présenter la modélisation en archirtecture ReDy d'un système Smart Grid générique en explicitant la modélisation de chaque composant. Ensuite, nous allons présenter les micro Smart Grids qui sont des Smart Grids de petite taille, conçus pour fournir un approvisionnement électrique fiable et de meilleure qualité à un petit nombre de consommateurs. Les micro Smart Grids sont principalement conçus pour alimenter des zones géographiquement isolées et permettent une gestion optimisée de la production d'électricité tout en intégrant les énergies de sources renouvelables disponibles à l'échelle locale. Notre cas d'étude est un micro Smart Grid composé de quatre noeuds principaux. L'objectif est d'assurer un équilibre entre la consommation et la production en intégrant les compteurs intelligents. Pour ce faire, nous proposons une modélisation formelle du micro Smart Grid étudié, puis nous appliquons les techniques de model checking pour vérifier la validité du modèle proposé.

7.2 Modélisation d'un Smart Grid par l'architecture ReDy

Dans cette partie, nous proposons une modélisation des Smart Grids par l'architecture ReDy. Nous proposons de se focaliser sur les aspects fonctionnels et distribués des Smart Grids. L'objectif est de permettre d'analyser la fiabilité et l'aspect dynamique de ces derniers. C'est pourquoi nous modélisons un Smart Grid par une architecture ReDy où chaque noeud du réseau électrique représente un sous-système de l'architecture ReDy.

Dans un Smart Grid chaque noeud est doté d'un dispositif de gestion de l'énergie qui est responsable de contrôler le fonctionnement des appareils électroniques et de décider de les mettre en marche ou de les arrêter selon des critères et des règles spécifiées lors de l'implémentation du dispositif. Ce dispositif est en communication à la fois avec les éléments internes au noeud et avec le monde extérieur à savoir les autres noeuds du Smart Grid. Ce dispositif central du noeud est modélisé dans l'architecture ReDy par l'unité de gouvernance du sous-système.

Chaque noeud est composé de différents éléments producteurs et/ou consommateurs d'énergie électrique. Il est d'autre part équipé d'un ensemble de capteurs pour le suivi de la production ou de la consommation d'énergie. Ces capteurs sont modélisés dans l'architecture ReDy par des unités de détection.

Un noeud peut être équipé d'actionneurs pour lancer ou arrêter la production d'un producteur ou la consommation d'un consommateur. Ces actionneurs sont modélisés dans l'architecture ReDy par des unités d'action.

TABLE 7.1 – Eléments de modélisation des Smart Grids par l'architecture ReDy

Eléments du Smart Grid	**Eléments de l'architecture ReDy**
Smart Grid	Système ReDy
Noeud du Smart Grid	Sous-système du système ReDy
Dispositif de gestion de l'énergie	Unité de gouvernance
Capteurs pour le suivi de la consommation/production de l'énergie	Unités de détection
Actionneur pour lancer/arrêter la production d'un producteur ou la consommation d'un consommateur	Unités d'action

L'unité de gouvernance, les unités de détection et d'action d'un sous-système diffèrent

selon la nature du noeud concerné. Un noeud peut être une maison intelligente, une usine, un bâtiment intelligent, une centrale de production de l'électricité ou autres.

Le réseau électrique est un réseau à configuration plutôt stable et nous essayons de garder cette stabilité au fil du temps. Nous travaillons également à rendre le réseau électrique tolérent à l'ajout et suppression de noeuds ainsi qu'à la défaillance de certains équipements en assurant un service tant qu'un chemin physique le permet. Le réseau électrique est régi par les règles de la physique qui stipulent par exemple que l'électron produit est toujours consommé par le consommateur le plus proche.

D'autre part, nous nous intéressons dans notre modélisation au réseau de données qui doit superviser et gérer l'aspect distribué et dynamique du Smart Grid. L'utilisation de l'architecture ReDy nous permet de réaliser cet objectif vu qu'elle propose un modèle de gestion décentralisé des différents noeuds du réseau (sous-systèmes de notre architecture ReDy). Chaque noeud garde une liste de voisins et échange une partie de ces voisins à chaque tour pour découvrir automatiquement de nouveaux voisins et éliminer automatiquement les voisins qui n'interagissent plus (soit parce qu'ils sont défaillants ou parce qu'ils ont quitté le réseau). Le noeud va toujours communiquer les informations relatives à sa production et à sa consommation d'électricité à ses voisins et communiquera aussi ses possibilités d'augmentation et de diminution de production ainsi que la disponibilité de consommateur flexible (voiture électrique à recharger sans urgence, ballon de chauffe-eau électrique, batterie, etc.) à actioner au cas où une production gratuite ou à faible coût est disponible.

7.3 Etude fonctionnelle des Smart Grids

Dans cette partie, nous allons citer les fonctionnalités principales qui doivent être réalisées globalement dans les Smart Grids. Ces fonctionnalités sont divisées en deux modes : le mode local et le mode distant. Le mode local concerne les fonctionnalités qui doivent être remplies par un noeud du réseau, vu qu'il représente les communications qui se font entre les entités faisant partie du même noeud. Le mode distant concerne les échanges entre les différents noeuds du réseau.

La figure 7.1 illustre un exemple d'un Smart Grid composé de trois noeuds (sous-systèmes). Chaque noeud est composé de trois types d'entités :

- Compteur intelligent : le dispositif intelligent est représenté dans ce schéma par un

compteur intelligent noté CI.

- Actionneur de consommation ou de production.

- Détecteur de consommation ou de production.

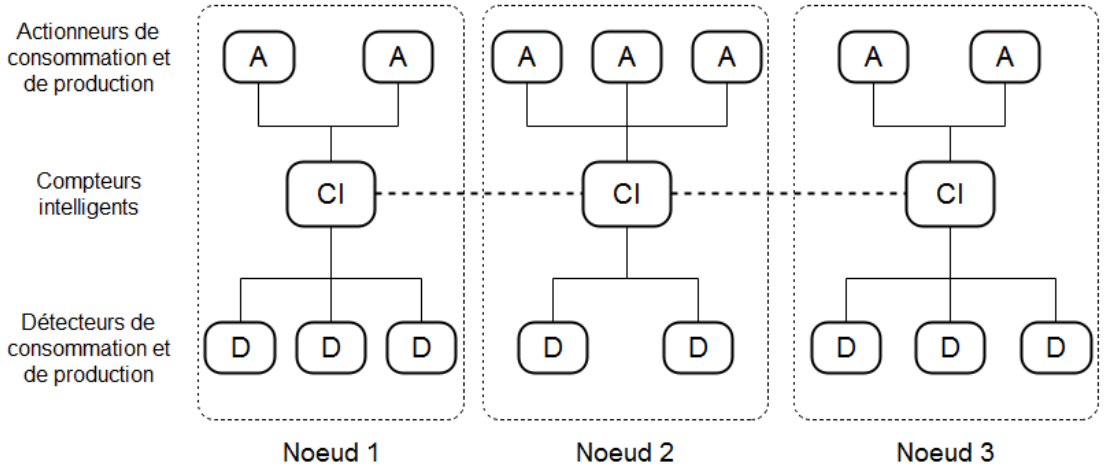


FIGURE 7.1 – Modélisation d'un Smart Grid par l'architecture ReDy

7.3.1 Mode local

Dans le mode local nous traitons les communications au sein du même noeud représenté par le sous-système. Le but est de garantir une communication centralisée entre le dispositif intelligent représenté par l'unité de gouvernance et les différents capteurs et compteurs représentés par les unités de détection et les actionneurs représentés par les unités d'action.

L'objectif est d'avoir une donnée temps réel au niveau du dispositif intelligent du noeud (compteur intelligent) de l'état de production et/ou consommation d'énergie électrique et d'avoir une vue sur les capacités de production présentes ainsi que les capacités de consommation présentes. Ce dispositif est le décideur et le porte parole de tous les éléments du noeud vers l'extérieur du noeud.

7.3.2 Mode distant

Les fonctionnalités requises dans le mode distant concernent les communications qui se font entre les noeuds du réseau.

Un noeud principalement consommateur communique la quantité d'énergie utilisée grâce au compteur intelligent. Il communique aussi ses consommations flexibles prêtes à être activées ou celles en cours prêtes à être désactivées.

Un noeud principalement producteur communique la quantité d'énergie injectée dans le réseau. Il communique aussi ses capacités de production non utilisées et son plan de production dans la période à venir. Une centrale d'énergie renouvelable peut communiquer ses prévisions de production relatives à ses ressources renouvelables intermitantes. Pour une énergie renouvelable avec une importante capacité de stockage comme l'énergie solaire thermique, l'état des stocks d'énergie est aussi à communiquer.

Au cas d'un manque de production ou d'un besoin de consommation, les données des disponibilités de ressources additionnelles ou la possibilité de désactiver de la consommation non urgente permet de mieux gérer les ressources disponibles et surtout de mieux gérer le coût de l'électricité. De même, au cas d'un excès de production les producteurs les plus chers, polluants ou flexibles peuvent être avertis pour réduire leur production. Dans ce dernier cas nous pouvons aussi lancer des consommations plus flexibles.

C'est une intelligence globale du réseau qui peut être garantie de façon distribuée et décentralisée.

7.4 Présentation du micro Smart Grid à modéliser formellement

Les micro Smart Grids sont des Smart Grids de petite taille, conçus pour fournir un approvisionnement électrique fiable et de meilleure qualité à un petit nombre de consommateurs. Ils sont principalement conçus pour alimenter des zones géographiquement isolées et permettent ainsi une gestion optimisée de la production d'électricité tout en intégrant les énergies de sources renouvelables disponibles à l'échelle locale. Dans la suite, nous allons présenter le cas d'étude relatif au micro Smart Grids.

Le micro Smart Grid peut être modélisé selon l'architecture ReDy. Dans ce qui suit, nous présentons le modèle du micro Smart Grid selon l'architecture ReDy qui sera utilisé à des fins de validation formelle.

L'architecture modélisée est composée de quatre nœuds (figure 7.2). Chaque nœud est présenté comme suit :

- Noeud 1 : un noeud principalement consommateur. Il s'agit d'une maison intelligente intégrant une petite source de production d'énergie (panneaux solaires). Ce nœud contient un compteur intelligent (SM), deux capteurs de consommation (CS), deux actionneurs de consommation (CA), un capteur de production (PS) et un actionneur de production (PA).

- Nœud 2 et Nœud 3 : un nœud consommateur. Il s'agit d'une maison normale qui consomme de l'énergie sans aucune source de production. Un tel nœud contient un compteur intelligent (SM), deux capteurs de consommation (CS) et deux actionneurs de consommation (CA).

- Noeud 4 : un nœud principalement producteur. Il s'agit d'une centrale électrique produisant de l'énergie (solaire et combustible). Ce nœud contient un compteur intelligent (SM), un capteur de consommation (CS), un actionneur de consommation (CA), deux capteurs de production (PS) et deux actionneurs de production (PA).

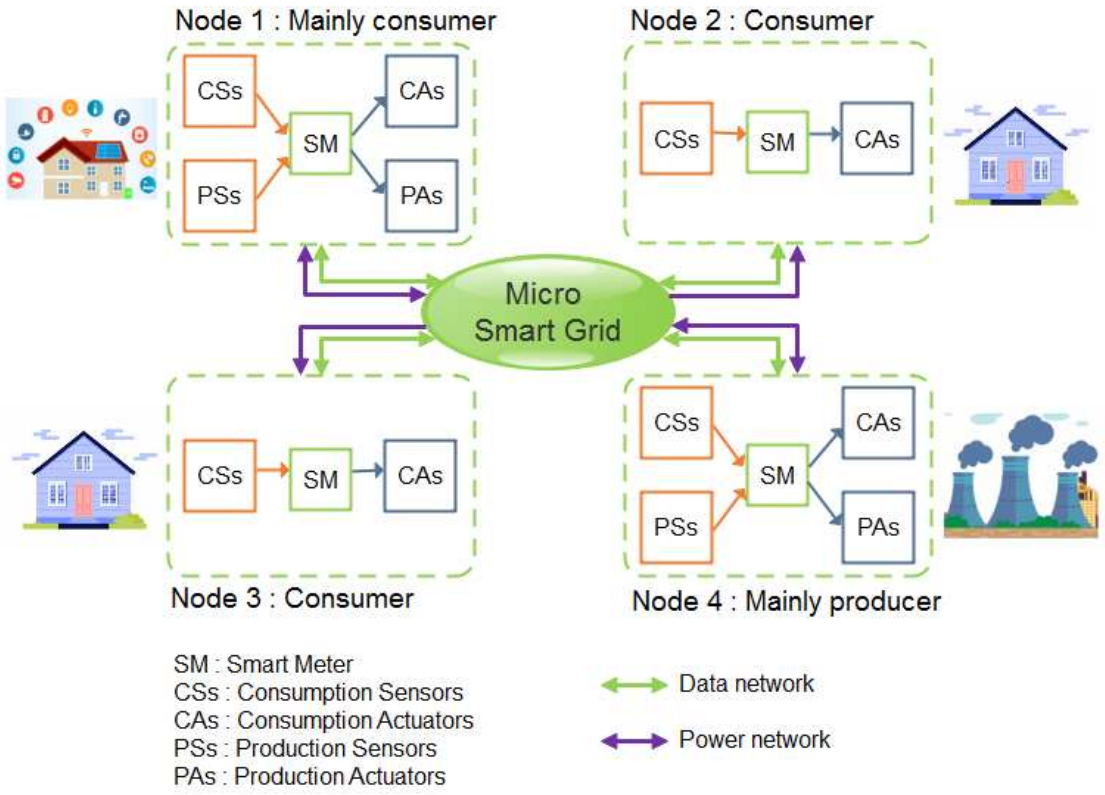


FIGURE 7.2 – Modélisation du micro Smart Grid étudié selon l'architecture ReDy

Il est à noter que les différentes entités de chaque noeud sont implémentés de façon à satisfaire des besoins précis relatifs au cas d'application en question. Dans notre travail,

nous nous intéressons aux fonctionnalités suivantes :

- Un détecteur de consommation (CS) déclenche un signal d'excès de consommation lorsque la consommation dépasse la production disponible. Ceci est détecté notamment dans un réseau à courant alternatif (AC) par la diminution de la fréquence à moins de 50 Hz et atteint un seuil par exemple de 49,9 Hz, ce qui nécessite de lancer une production flexible (telle que le diesel). Une telle décision est prise par le compteur intelligent et exécutée par l'actionneur de production (PA).

- Un détecteur de production (PS) déclenche un signal de surplus de production lorsque la production dépasse la consommation globale du réseau. Dans un réseau électrique à courant alternatif (AC), ceci est détecté par l'augmentation de la fréquence à plus de 50 Hz en atteignant un seuil de 50,1 Hz par exemple, ce qui permet de lancer une consommation flexible (telle que les batteries à recharger, voiture électrique). Une telle décision est prise par le compteur intelligent et exécutée par l'actionneur de consommation (CA).

7.5 Modélisation formelle du Micro Smart Grid

Dans cette partie, nous présentons la modélisation formelle de notre cas d'étude de Micro Smart Grid en décrivant la modélisation des différents éléments modélisés. En effet, nous présentons la modélisation en processus LNT des entités suivantes : le capteur de consommation, le capteur de production, l'actionneur de consommation, l'actionneur de production et le compteur intelligent. Ensuite, nous présentons la modélisation d'un noeud du micro Smart Grid pour enfin arriver au modèle formel global de notre cas d'étude.

En utilisant le langage formelle LNT, nous proposons un modèle formel de l'architecture du micro Smart Grid étudié. Chaque composant de l'architecture est modélisé par un processus LNT. La structure globale de l'architecture est modélisée par des communications entre des processus LNT sur des portes de communication. Dans l'architecture étudiée, nous avons deux modes :

- Mode local : consiste en des interactions entre les composants d'un même nœud, c'est-à-dire entre le compteur intelligent et différents capteurs et actionneurs.

- Mode distant : consiste en des interactions entre des composants appartenant à différents

nœuds, c'est-à-dire entre les différents compteurs intelligents du micro Smart Grid.

Dans la suite, nous présentons le modèle d'un nœud parmi le micro Smart Grid puis nous présentons la composition parallèle utilisée pour obtenir le modèle du micro Smart Grid étudié.

7.5.1 Modélisation formelle d'un noeud du micro Smart grid

Afin de modéliser un nœud, nous commençons par modéliser les différents composants de l'architecture par des processus LNT définis dans des modules LNT. Nous définissons cinq modules LNT : détecteur de consommation (*ConsumptionSensor*), actionneur de consommation (*ConsumptionActuator*), détecteur de production (*ProductionSensor*), actionneur de production (*ProductionActuator*), compteur intelligent (*SmartMeter*).

Dans le module ConsumptionSensor, nous définissons un processus du même nom qui est paramétré par une porte de communication ConsS et par deux variables : idNode représentant l'indice du noeud auquel appartient le capteur, et idCS représentant l'indice du capteur. Le corps du processus est défini par une boucle sur la porte de communication ConsS. ConsS est exécuté, dans notre cas, lorsque la consommation est supérieure à la production. C'est un seuil à définir plus en détail selon le besoin.

```
module ConsumptionSensor (types) is

process ConsumptionSensor[ConsS : any]
    (idNode:index_Node,idCS:index_CS)
is
   loop
          ConsS(idNode,idCS)

   end loop
end process

end module
```

De la même façon, nous définissons le module ProductionSensor. Le corps du processus est défini par une boucle sur la porte de communication ProdS qui est exécutée lorsque

la production est supérieure à la consommation. C'est un seuil à définir plus en détail selon le besoin.

```
module ProductionSensor (types) is

process ProductionSensor[ProdS : any]
    (idNode:index_Node,idPS:index_PS)
is
   loop
          ProdS(idNode,idPS)

   end loop
end process

end module
```

Le module ConsumptionActuator est défini également par un processus avec une boucle permanente. L'action ConsA est exécutée lorsqu'on désire lancer un consommateur. Cette action doit dépendre des détections réalisées par les capteurs ainsi que la décision du compteur intelligent.

```
module ConsumptionActuator (types) is

process ConsumptionActuator[ConsA : any]
    (idNode:index_Node,idCA:index_CA)
is
   loop
          ConsA(idNode,idCA)

   end loop
end process

end module
```

De même, nous définissons le module ProductionActuator qui est défini également par

un processus avec une boucle permanente avec l'action ProdA qui est exécutée lorsqu'on désire lancer un producteur.

```
module ProductionActuator (types) is

process ProductionActuator[ProdA : any]
    (idNode:index_Node,idPA:index_PA)
is
   loop
          ProdA(idNode,idPA)

   end loop
end process

end module
```

Le module *SmartMeter* est modélisé similairement au module gouvernance_unit d'une architecture ReDy. Il est composé de quatre processus. Le processus *SmartMeter* initialise et appelle le processus *membership_management*. Ce processus est responsable de la construction du réseau des composants. En particulier, ce processus est responsable de l'ajout et de la suppression des noeuds du micro Smart Grid. Le processus *membership_management* appelle le processus *local_communication* qui est responsable de la communication à l'intérieur du même noeud : la communication entre le SmartMeter et les processus ConsumptionActuator, ProductionActuator, ConsumptionSensor et ProductionSensor.

Le processus *local_communication* appelle le processus *distant_communication*. Ce dernier assure la communication entre le noeud actuel et les autres noeuds, en particulier il assure la communication entre les SmartMeters à travers la porte DisComm.

Le processus distant_communication appelle recursivement le processus *membership_management*.

Le processus SmartMeter est paramétré par cinq portes : ConsS, ProdS, ConsA, ProdA et DisComm qui représentent des portes de communication entre les différentes unités du système. Ce processus est paramétré également par la variable idNode qui est un indice de type index_node.

```
process SmartMeter[ConsS, ProdS, ConsA, ProdA, DisComm : any]
   (idNode: index_node)
is
   var
   new_ConsS : bool,
   new_ProdS : bool
   in
      new_ConsS := false;
      new_ProdS := false;

-- Nodes initialization here --

      membership_management[ConsS, ProdS, ConsA, ProdA, DisComm]
         (idNode, new_ConsS, new_ProdS, action_to_do,  idNode)

   end var
end process
```

L'étape suivante consiste à exécuter une itération de brassage en définissant le processus membership_management. Ce processus est paramétré par cinq portes de communication et par quatre variables.

Ce processus appelle le processus local_communication en lui passant en paramètre les variables déclarées.

```
process membership_management[ConsS, ProdS, ConsA, ProdA, DisComm : any]
   (idNode:index_node,
    new_ConsS : bool,
    new_ProdS : bool,
    idNode_other :index_node
    )
is
   -- shuffling algorithm here  --
```

```
   local_communication [ConsS, ProdS, ConsA, ProdA, DisComm]
         (idNode, new_ConsS, new_ProdS, action_to_do,  idNode_other)
end process
```

Le processus local_communication assure l'exécution d'une communication en mode local, ce qui correspond à une communication entre :

- SmartMeter et ConsumptionSensor sur la porte ConsS.

- SmartMeter et ConsumptionActuator sur la porte ConsA.

- SmartMeter et ProductionSensor sur la porte ProdS.

- SmartMeter et ProductionActuator sur la porte ProdA.

```
process local_communication [ConsS, ProdS, ConsA, ProdA, DisComm : any]
   (idNode : index_node,
    new_ConsS : bool,
    new_ProdS : bool,
    idNode_other : index_node)
is
   var id_ConsS_unit: index_ConsS,
       id_ProdS_unit: index_ProdS,
       id_ConsA_unit: index_ConsA,
       id_ProdA_unit: index_ProdA
   in

   select
      null
   []
      only if not(new_ConsS) and not (new_ProdS) then
         ConsS(idNode, ?id_ConsS_unit);
         new_ConsS :=true
      end if
```

```
   []
      only if not(new_ConsS) and not (new_ProdS) then
         ProdS(idNode, ?id_ProdS_unit);
         new_ProdS :=true
      end if
   []
      only if (new_ConsS) then
         ProdA(idNode_other, idNode, ?id_ProdA_unit);
         new_ConsS :=false
      end if
   []
      only if (new_ProdS) then
         ConsA(idNode_other, idNode, ?id_ConsA_unit);
         new_ProdS :=false
      end if

   end select;
   distant_communication [ConsS, ProdS, ConsA, ProdA, DisComm]
         (idNode, new_ConsS, new_ProdS, action_to_do,  idNode_other)
   end var
end process
```

Le processus distant_communication assure l'exécution d'une communication en mode distant, ce qui correspond à une communication entre les SmartMeters.

```
process distant_communication [ConsS, ProdS, ConsA, ProdA, DisComm : any]
   (idNode : index_node,
    new_ConsS : bool,
    new_ProdS : bool,
    idNode_other : index_node)
is

    select
      null
```

```
   []
      only if (new_ConsS) then
         DisComm(idNode, "ConsS");
         new_ConsS := false
      end if
   []
      only if (new_ProdS) then
         DisComm(idNode, "ProdS");
         new_ConsS := false
      end if
   []
      DisComm(?idNode_other, "ConsS") where (idNode_other<>idNode);
      new_ConsS:=true
   []
      DisComm(?idNode_other, "ProdS") where (idNode_other<>idNode);
      new_ProdS:=true
   end select;

   membership_management [ConsS, ProdS, ConsA, ProdA, DisComm]
         (idNode, new_ConsS, new_ProdS, action_to_do,  idNode_other)
end process
```

Dans un nœud du micro Smart Grid, la communication est assurée à l'aide de portes de communication. Nous définissons quatre portes de communication utilisées en mode local (c'est-à-dire au sein du même nœud). La figure 7.3 illustre différents processus et portes de communication définis dans notre modèle formel. ConsA est la porte de communication entre les processus SmartMeter et ConsumptionActuator. ProdA est la porte de communication entre SmartMeter et ProductionActuator. ConsS est la porte de communication entre SmartMeter et ConsumptionSensor. ProdS est la porte de communication entre SmartMeter et ProductionSensor.

Chaque nœud du micro Smart Grid est défini par un module appelé MicroSmartGridNode. Ce module utilise cinq autres modules : SmartMeter, ConsumptionSensor, ProductionSensor, ConsumptionActuator, ProductionActuator. Ensuite, nous définissons le processus MicroSmartGridNode par une composition parallèle entre le processus SmartMeter et

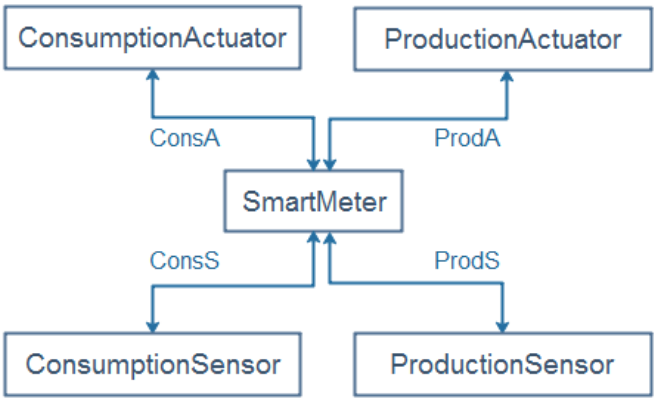


FIGURE 7.3 – Portes de communication en mode local

tous les autres processus sur les portes de communication définies (ConsS, ProdS, ConsA, ProdA). Ensuite, nous définissons la composition du nœud, c'est-à-dire le nombre de chaque capteur / actionneur dans le nœud. Dans notre exemple, nous pouvons définir un nœud avec un ou deux capteurs de consommation (ConsumptionSensor). En utilisant la même méthode, nous pouvons définir le nombre requis de chaque capteur / actionneur dans le nœud du micro Smart Grid en donnant différentes valeurs aux variables concernées (idNode, nbCS, nbPS, nbCA, nbPA).

```
module MicroSmartGridNode(SmartMeter, ConsumptionSensor, ProductionSensor,
                          ConsumptionActuator, ProductionActuator) is

process MicroSmartGridNode [ConsS, ProdS, ConsA, ProdA, DisComm : any]
 (idNode:index_Node, nbCS:index_CS, nbPS:index_PS, nbCA:index_CA, nbPA:index_PA)

is

   par ConsS, ProdS, ConsA, ProdA in
   --Smart Meter
     SmartMeter[ConsS, ProdS, ConsA, ProdA, DisComm] (idNode)
     ||
       par
       -- Consumption Sensors
          if (nbCS==index_CS(1)) then ConsumptionSensor[ConsS](idNode,index_CS(1))
          elsif (nbCS==index_CS(2)) then
          par
```

```
            ConsumptionSensor[ConsS](idNode,index_CS(1))
            ||
            ConsumptionSensor[ConsS](idNode,index_CS(2))
            end par
          end if
         ||
            -- Production Sensors
  if (nbPS==index_PS(1)) then ProductionSensor[ProdS](idNode,index_PS(1))
          elsif (nbPS==index_PS(2)) then
          par
            ProductionSensor[ProdS](idNode,index_PS(1))
            ||
            ProductionSensor[ProdS](idNode,index_PS(2))
            end par
          end if

         ||
            -- Consumption Actuators
  if (nbCA==index_CS(1)) then ConsumptionActuator[ConsA](idNode,index_CA(1))
          elsif (nbCA==index_CA(2)) then
          par
            ConsumptionActuator[ConsA](idNode,index_CA(1))
            ||
            ConsumptionActuator[ConsA](idNode,index_CA(2))
            end par
          end if
         ||
            -- Production Actuators
  if (nbPA==index_PA(1)) then ProductionActuator[ProdA](idNode,index_PA(1))
          elsif (nbPA==index_PA(2)) then
          par
            ProductionActuator[ProdA](idNode,index_PA(1))
            ||
            ProductionActuator[ProdA](idNode,index_PA(2))
            end par
          end if
```

```
        end par
      end par
end process

end module
```

7.5.2 Modélisation formelle du micro Smart Grid

Une fois que nous avons modélisé le nœud du micro Smart Grid étudié, nous pouvons maintenant modéliser l'ensemble du micro Smart Grid en exécutant une composition parallèle entre quatre nœuds sur une porte de communication nommée DisComm (Figure 7.4).

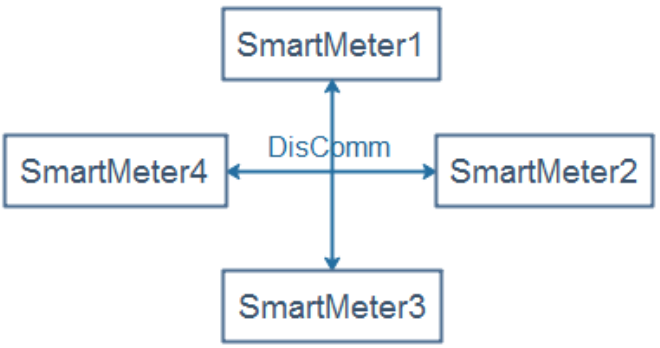


FIGURE 7.4 – Portes de communication dans le mode distant

Le modèle du micro Smart Grid est défini par une composition parallèle entre quatre processus. Chaque processus représente un nœud du micro Smart Grid. Pour chaque processus, nous définissons des portes de communication et des valeurs de variables. Le premier processus (MicroSmartGridNode d'indice 1) contient deux ConsumptionSensor (capteurs de consommation), un seul ProductionSensor (capteur de production), deux ConsumptionActuator (actionneurs de consommation) et un seul ProductionActuator (actionneur de production). Le deuxième / troisième processus (MicroSmartGridNode d'indice 2/3) contient deux ConsumptionSensor (capteurs de consommation) et deux ConsumptionActuator (actionneurs de consommation). Le quatrième processus (MicroSmartGridNode d'indice 4) contient un seul ConsumptionSensor (capteurs de consommation), deux ProductionSensor (capteur de production), un seul ConsumptionActuator (actionneurs de consommation) et deux ProductionActuator (actionneur de production).

```
module main(MicroSmartGridNode) is
```

```
process MAIN [ConsS, ProdS, ConsA, ProdA, DisComm : any]
is

 par DisComm in

 MicroSmartGridNode [ConsS, ProdS, ConsA, ProdA, DisComm]
 (index_Node(1), index_CS(2), index_PS(1), index_CA(2), index_PA(1))
 -- Node number 1 with 2 CSs 1 PS 2 CAs 1 PA

 ||
 MicroSmartGridNode [ConsS, ProdS, ConsA, ProdA, DisComm]
 (index_Node(2), index_CS(2), index_PS(0), index_CA(2), index_PA(0))
 -- Node number 2 with 2 CSs 2 CAs

 ||
 MicroSmartGridNode [ConsS, ProdS, ConsA, ProdA, DisComm]
 (index_Node(3), index_CS(2), index_PS(0), index_CA(2), index_PA(0))
 -- Node number 3 with 2 CSs 2 CAs

 ||
 MicroSmartGridNode [ConsS, ProdS, ConsA, ProdA, DisComm]
 (index_Node(4), index_CS(1), index_PS(2), index_CA(1), index_PA(2))
 -- Node number 4 with 1 CS 2 PSs 1 CA 2 PAs

 end par

end process

end module
```

7.6 Validation formelle

Après avoir modélisé formellement un micro Smart Grid, nous proposons des propriétés à vérifier grâce aux techniques de vérification de modèle (model checking). Dans la suite

nous décrivons deux propriétés, qui assure chacune la bonne terminaison des transactions du modèle, c'est à dire l'absence d'inter-blocage (deadlock) ou de blocage opérationnel (livelock). Les propriétés à vérifier sont exprimées en MCL. Le langage MCL traite les actions et pas les états, considère une logique avec branchement et manipule les données c'est-à-dire peut enregistrer une donnée au moment d'une action et vérifier dans la suite qu'une autre action arrive manipulant la même donnée.

Les deux propriétés suivantes utilisent la sémantique de l'inévitabilité, c'est à dire qu'une fois une action A arrive, alors inévitablement une action B arrivera. Dans une logique branchée telle que le MCL, cela signifie que quelle que soit la branche prise l'action B va être vraie à un moment futur.

Nous exprimons ici en MCL la macro *inevitable*, déjà utilisée dans la validation de l'architecture ReDy générique :

```
macro inevitable (L) =
   mu X . ( < true > true and [ not L ] X )
end_macro
```

La première propriété MCL vérifie que chaque fois qu'un excès de consommation est détecté par l'étiquette ConsS au niveau du noeud d'indice IdNode, inévitablement il existe au moins une production flexible qui est activée avec l'étiquette ProdA :

```
[ true * . {ConsS ?IdNode:Nat ...} ]
inevitable ( {ProdA !IdNode ...} )
```

La figure 7.5 illustre cette propriété.

La deuxième propriété MCL vérifie que chaque fois qu'un surplus de production est détecté par l'étiquette ProdS au niveau du noeud d'indice IdNode, inévitablement une consommation flexible est activée avec l'étiquette ConsA :

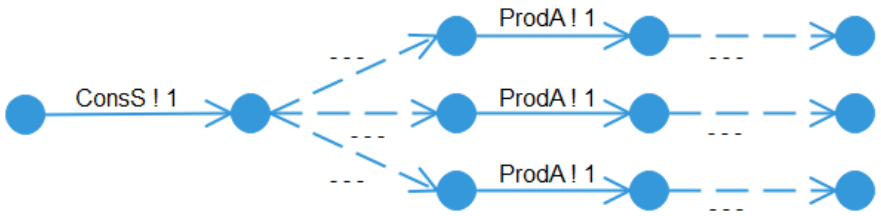


FIGURE 7.5 – Propriété MCL ordre ConsS ProdA

```
[ true * . {ProdS ?IdNode:Nat ...} ]
inevitable ( {ConsA !IdNode ...} )
```

La figure 7.6 illustre cette propriété.

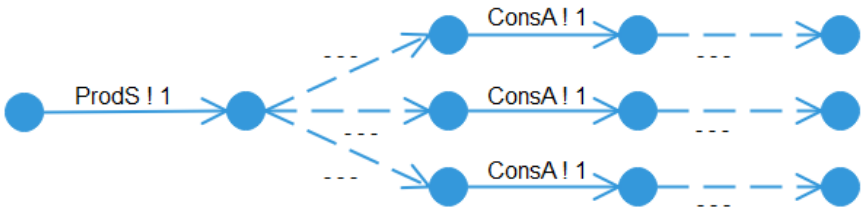


FIGURE 7.6 – Propriété MCL ordre ProdS ConsA

Si le système contient des inter-blocages ou des blocages opérationnels alors au moins l'une de ces deux propriétés ne sera pas vérifiée.

Grâce à l'utilisation des techniques de vérification de modèle, nous avons pu détecter des deadlocks et livelocks dans des versions intermédiaires de notre modèle formel. Tous les inter-blocages et blocages opérationnels ont été corrigés dans la dernière version du modèle que nous avons présentée dans ce rapport.

7.7 Conclusion

Dans ce chapitre, nous avons présenté l'application de notre approche dans le cas d'étude des micro Smart Grids. En effet, le système étudié est composé de quatre noeuds tels que chaque noeud comporte un compteur intelligent, des capteurs de consommation

et/ou de production ainsi que des actionneurs de consommation et/ou de production. Nous avons présenté la modélisation formelle en langage LNT des différents composants de chaque noeud, puis celle de chaque noeud du micro Smart Grid. En appliquant une composition parallèle entre les quatre noeuds, nous obtenons le modèle formel du système étudié. Ensuite nous somme passés à l'exploitation du modèle en vérifiant tout d'abord l'absence d'inter-blocages et de blocages opérationnels, puis en vérifiant des propriétés de logique temporelle écrites en langage MCL afin de prouver le comportement désiré. Ce travail de modélisation et de validation formelle nous permet d'assurer l'équilibre entre la consommation et la production au niveau du micro Smart Grid étudié. Cette analyse peut être utile pour mieux gérer l'intelligence des dispositifs utilisés dans les Smart Grids de demain : le but est d'avoir une gestion plus dynamique des composants d'un Smart Grid, tout en gardant un niveau suffisant de fiabilité.

Chapitre 8

Conclusion et perspectives

8.1 Résumé des contributions

En terme de synthèse, les systèmes IoT sont en pleine voie de recherche et d'investigation afin d'acquérir de plus en plus de maturité pour être intégrés dans plusieurs domaines d'applications. Notre travail contribue aux travaux de recherche dans le domaine en proposant une architecture ReDy qui permet de concevoir un système IoT fiable, dynamique et extensible. Nos propos sont validés formellement en utilisant les techniques de model checking et en s'outillant de la boite à outils CADP qui offre une large panoplie de scénarios de validation formelle.

Dans cette thèse nous avons illustré différentes contributions que nous résumons dans ce qui suit :

Etat de l'Art de l'Internet des Objets (IoT) : les systèmes IoT sont au croisement de nombreux domaines et applications technologiques. Afin de traiter un tel paradigme nous devrions avoir une vision globale de plusieurs aspects liés aux systèmes IoT. C'est pourquoi nous avons mené une étude sur différentes architectures IoT existantes qui permettent de mettre en œuvre des systèmes IoT à différentes couches. A partir des architectures étudiées, nous avons pu proposer une architecture IoT de bout en bout générale composée de cinq couches principales : dispositifs, réseaux, middleware, application et business. Nous avons résumé les principales technologies pouvant être utilisées pour implémenter les systèmes IoT en les organisant selon les couches d'architecture IoT de bout en bout proposée. Nous avons présenté les principaux défis rencontrés lors de la conception et de la

mise en œuvre des systèmes IoT. Par ce travail, nous avons pu dresser une vision globale des systèmes IoT d'un point de vue académique afin d'être initié à leur conception et mise en œuvre. Notre proposition peut être utilisée dans de nombreux domaines d'application des systèmes IoT tels que les maisons intelligentes, la surveillance de la santé, le transport intelligent, l'agriculture intelligente, etc. Les domaines d'application concernés par notre proposition sont ceux qui demandent un niveau d'exigence envers les trois défis, objets de nos préoccupations, et qui sont : l'extensibilité, le dynamisme et la fiabilité.

Proposition de l'Architecture ReDy : Dans cette thèse, nous avons proposé l'architecture ReDy (Reliable and Dynamic) déstinée à concevoir des systèmes IoT fiables, dynamiques et extensibles. Il s'agit d'une architecture hybride vu qu'elle intègre des solutions centralisées (client/serveur) et décentralisées (peer to peer) pour réaliser l'ensemble des besoins fonctionnels et non fonctionnels du système IoT en question. Nous avons d'abords présenté l'architecture ReDy utilisée comme support lors de la conception d'un ensemble de systèmes IoT partageant des exigences communes. Il s'agit d'une solution réutilisable qui assure trois exigences principales. La première est une forte dynamicité du système résultant permettant aux nœuds de quitter et de rejoindre le système sans altérer son fonctionnement normal. La seconde fournit la fiabilité vis-à-vis d'une défaillance. La troisième consiste à assurer l'extensibilité du système.

Afin d'établir la conception de notre architecture de base, nous avons effectué une analyse des différentes phases nécessaires à la conception d'un système IoT. Nous avons présenté des solutions pour la construction du système, ainsi que des solutions pour le mode de communication, tout en tenant compte que les exigences globales soient satisfaites. De plus, nous présentons l'architecture logicielle adaptée à l'organisation des modules logiciels des composants du système IoT tout en couvrant toutes les phases de conception.

Formalisation de l'algorithme de brassage : Afin d'assurer les critères de l'architecture ReDy, nous avons proposé d'utiliser un algorithme de brassage pour la gestion de la construction de graphe des noeuds qui composent notre système. Les éléments de base de cet algorithme de brassage ont été décrits dans la littérature en langue naturelle. Nous avons proposé alors une fomalisation algorithmique de l'algorithme amélioré utilisé pour la gestion de l'adhésion des noeuds au système ReDy. Cette formalisation permet d'assurer les critères de fiabilité et de dynamisme de l'architecture ReDy.

Modélisation Formelle de l'Architecture ReDy : Pour des fins de validation de notre proposition, nous avons utilisé les méthodes formelles pour prouver les critères recherchés

par notre architecture. Nous avons eu recours à une modélisation formelle par le language de spécification formelle LNT qui est basé sur le formalisme des algèbres des processus. En effet, nous avons proposé différentes versions d'un modèle formel de l'architecture ReDy et de l'algorithme de brassage. La modélisation formelle nous a permis d'être précis et rigoureux dans la spécification de la structure et du comportement de notre architecture ReDy. Le modèle a permis aussi de simuler différents comportements possibles et de générer des graphes états/transitions de type systèmes à transitions étiquetés, dites LTS, pour certaines configurations simples.

Validation formelle : Grâce aux techniques de model checking et à la boite à outils CADP qui permet l'exploitation des modèles formels LNT, nous avons pu prouver la fiabilité, le dynamisme et l'extensibilité de notre architecture. En effet, nous nous sommes concentrés sur la validation formelle de la partie de gestion d'adhésion des noeuds du système selon l'algorithme de brassage amélioré. A cause du problème d'explosion combinatoire de l'espace d'état, la validation formelle est limitée à de petites configurations des systèmes (c'est-à-dire, nombre de sous-systèmes, nombre d'unités de gouvernance/action/détection). Notre méthode fournit une validation exhaustive de ces différentes configurations. En conséquence, nous pouvons éliminer un grand nombre d'erreurs de spécification et détecter très tôt des défaillances du système.

Comme résultat, nous avons pu valider formellement le fonctionnement des systèmes ReDy jusqu'à quatre sous-systèmes sans interblocage avec une bonne terminaison des transactions entamées. Nous avons validé également deux scénarios intéressants : le système fonctionne correctement à l'ajout dynamique d'un nouveau sous système ainsi qu'à la suppression d'un sous système déja existant.

Notre méthode de travail a adopté un processus incrémental et itératif basé sur la validation formelle. Ce processus nous a permis d'évoluer lors de la conception de systèmes IoT selon l'architecture ReDy afin de pouvoir valider plusieurs configurations possibles. Le but est d'aboutir à la configuration désirée à travers les différentes itérations et incrémentations effectuées.

Application aux Smart Grids : Comme cas d'étude, nous avons proposé d'appliquer notre approche dans le contexte des micro Smart Grids. À cette fin, nous avons utilisé l'architecture ReDy garantissant l'évolutivité, la fiabilité et la gestion dynamique de l'adhésion des noeuds au système, ce qui permet aux noeuds du micro Smart Grid d'optimiser dynamiquement la production et la consommation d'énergie électrique.

8.2 Perspectives

Ce travail ouvre des perspectives intéressantes concernant différents aspects, tels que :

- Validation formelle de configurations plus grandes de l'architecture ReDy que celles réalisées dans ce travail de thèse. Un tel objectif peut être réalisé en utilisant des grids de calcul puissant pouvant supporter la taille du modèle formel généré. Nous pouvons penser également à utiliser des outils de la boite à outils CADP basés sur des algorithmes parallèles afin de diminuer le temps de calcul.

- Dans le cas d'étude des micro Smart Grids, il est possible de modéliser formellement des scénarios plus compliqués en implémentant des algorithmes intelligents dans les unités intelligentes afin de mieux gérer l'équilibre de l'énergie électrique entre la production et la consommation.

- Prévoir un mécanisme de réplication afin d'assurer un degré de fiabilité plus élevé. En effet, cela peut être réalisé en dupliquant l'élément le plus intelligent dans notre architecture qui est l'unité de gouvernance.

- Une autre amélioration possible consiste à ajouter d'autres exigences importantes à l'architecture conçue, telles que la sécurité. En effet, dans notre modèle, un noeud qui tombe en panne est modélisé par un processus qui ne répond plus. Dans la réalité, on peut avoir affaire à des noeuds ayant des intentions malveillantes qui continuent d'interagir dans le système. Ce type de noeud peut être modélisé par un processus de panne arbitraire et de prendre en compte un tel comportement lors de la modélisation de notre système afin d'assurer sa sécurité.

- Nous pouvons penser également à intégrer et prouver le critère de mobilité dans notre architecture ReDy afin de pouvoir maintenir le bon fonctionnement du système malgré la mobilité des dispositifs IoT. Ceci permettra une ouverture de notre architecture à d'autres domaines d'application de l'IoT ayant des dispositifs IoT mobiles.

- Implémenter un prototype pour les micro Smart Grids en utilisant les dispositifs électroniques adéquats. Ce prototype permettra de tester la solution et de proposer une configuration optimale de l'algorithme de brassage dans le cas des micro Smart Grids.

- Application de notre approche dans d'autre cas d'étude : l'agriculture intelligente par exemple.

Enfin, notre travail permet d'ouvrir de grands chantiers de recherche et de recherche-développement au point où même notre architecture ReDy peut être revue pour intégrer de nouvelles couches répondant aux nouveaux besoins et défis de certains domaines spécifiques.

8.3 Publications scientifiques

Notre thèse a donné lieu à plusieurs publications dans des conférences et journaux internationaux.

- Kaoutar Hafdi and Abdelaziz Kriouile. Designing ReDy distributed systems. In International Conference on Autonomic Computing (ICAC), 2015, pages : 331-336. IEEE, 2015.

- Kaoutar Hafdi, Abdelaziz Kriouile, and Abderahman Kriouile. Formal modeling and validation of ReDy architecture intended for IoT applications. In International Journal of Innovative Research in Computer Science and Technology (IJIRCST), Volume 5, Issue 4 , pages : 339-349, 2017.

- Kaoutar Hafdi, Abderahman Kriouile, and Abdelaziz Kriouile. IoT ReDy architecture for Smart Grid management. In Computer and Information Science (CIS), Volume 11, Number 4, pages : 36-44, 2018.

- Kaoutar Hafdi, Abderahman Kriouile, and Abdelaziz Kriouile. Overview on Internet of Things (IoT) Architectures, Enabling Technologies and Challenges. In : Journal of Computers (JCP), Volume 14, Number 9, pages : 557-570, September 2019.

- Kaoutar Hafdi and Abderahman Kriouile. Formal Modeling and Validation of Micro Smart Grids Based on IoT ReDy Architecture. In International Conference on Cloud Computing and Artificial Intelligence : Technologies and Applications (CloudTech), 2020, pages : 270-276. IEEE, 2015

Bibliographie

[A+10] M Aigner et al. Bridge—building radio frequency identification for the global environment. report on first part of the security wp : Tag security (d4. 2.1), 2010.

[AB05] Ali Ziya Alkar and Umit Buhur. An internet based wireless home automation system for multifunctional devices. *IEEE Transactions on Consumer Electronics*, 51(4) :1169–1174, 2005.

[ACCM10] Federico Andreini, Flavio Crisciani, Claudio Cicconetti, and Raffaella Mambrini. Context-aware location in the internet of things. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 300–304. IEEE, 2010.

[AFGM+15] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things : A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4) :2347–2376, 2015.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things : A survey. *Computer networks*, 54(15) :2787–2805, 2010.

[AK15] C Alexakos and Athanasios P Kalogeras. Internet of things integration to a multi agent system based manufacturing environment. In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*, pages 1–8. IEEE, 2015.

[AMC07] Ian F Akyildiz, Tommaso Melodia, and Kaushik R Chowdhury. A survey on wireless multimedia sensor networks. *Computer networks*, 51(4) :921–960, 2007.

[AMI] Huawei AMI. . http://support.huawei.com/enterprise/docinforeader!loadDocument1.action?contentId=DOC1000106039&partNo=10052#adc_iot_pd_0012/. [Online ; accessed 23-november-2019].

[Ash09] Kevin Ashton. That internet of things thing. *RFiD Journal*, 22 :97–114, 2009.

[ASSC02] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks : a survey. *Computer networks*, 38(4) :393–422, 2002.

[ASW+99] Ken Arnold, Robert Scheifler, Jim Waldo, Bryan O'Sullivan, and Ann Wollrath. *Jini specification*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[ATR16] Mohammed Riyadh Abdmeziem, Djamel Tandjaoui, and Imed Romdhani. Architecting the internet of things : state of the art. In *Robots and Sensor Clouds*, pages 55–75. Springer, 2016.

[AUSA19] Bilal Afzal, Muhammad Umair, Ghalib Asadullah Shah, and Ejaz Ahmed. Enabling iot platforms for social iot applications : vision, feature mapping, and challenges. *Future Generation Computer Systems*, 92 :718–731, 2019.

[Azi16] Benjamin Aziz. A formal model and analysis of an iot protocol. *Ad Hoc Networks*, 36 :49–57, 2016.

[Bae05] Jos CM Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3) :131–146, 2005.

[BAPM83] Mordechai Ben-Ari, Amir Pnueli, and Zohar Manna. The temporal logic of branching time. *Acta informatica*, 20(3) :207–226, 1983.

[BCFD16] Ramazan Bayindir, I Colak, G Fulli, and K Demirtas. Smart grid technologies and applications. *Renewable and Sustainable Energy Reviews*, 66 :499–516, 2016.

[BPP98] Kamel Barkaoui and J-F Pradat-Peyre. Verification in concurrent programming with petri nets structural techniques. In *High-Assurance Systems Engineering Symposium, 1998. Proceedings. Third IEEE International*, pages 124–133. IEEE, 1998.

[BV17] Irena Bojanova and Jeffrey Voas. Trusting the internet of things. *IT Professional*, (5) :16–19, 2017.

[CAMD16] Massimo Condoluci, Giuseppe Araniti, Toktam Mahmoodi, and Mischa Dohler. Enabling the iot machine age with 5g : Machine-type multicast services for innovative real-time applications. *IEEE Access*, 4 :5555–5569, 2016.

[CBC+10] Angelo P Castellani, Nicola Bui, Paolo Casari, Michele Rossi, Zach Shelby, and Michele Zorzi. Architecture and protocols for the internet of things : A case study. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 678–683. IEEE, 2010.

[CCG+14] David Champelovier, Xavier Clerc, Hubert Garavel, Yves Guerte, Christine McKinty, Vincent Powazny, Frédéric Lang, Wendelin Serwe, and Gideon Smeding. Reference manual of the lnt to lotos translator (version 6.1). *Inria/Vasy and Inria/Convecs*, 131, 2014.

[CCG+17] David Champelovier, Xavier Clerc, Hubert Garavel, Yves Guerte, Frédéric Lang, Christine McKinty, Vincent Powazny, Wendelin Serwe, and Gideon Smeding. Reference manual of the lnt to lotos translator, 2017.

[CDDM+15] Luca Catarinucci, Danilo De Donno, Luca Mainetti, Luca Palano, Luigi Patrono, Maria Laura Stefanizzi, and Luciano Tarricone. An iot-aware architecture for smart healthcare systems. *IEEE Internet of Things Journal*, 2(6) :515–526, 2015.

[CGB+11] Angelo P Castellani, Mattia Gheda, Nicola Bui, Michele Rossi, and Michele Zorzi. Web services for the internet of things through coap and exi. In *Communications Workshops (ICC), 2011 IEEE International Conference on*, pages 1–6. IEEE, 2011.

[CGB16] Ray Chen, Jia Guo, and Fenye Bao. Trust management for soa-based iot and its application to service composition. *IEEE Transactions on Services Computing*, 9(3) :482–495, 2016.

[CGR11] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to reliable and secure distributed programming.* Springer, 2011.

[ČH] SAMIR ČAUŠEVIĆ and ADISA HASKOVIĆ. The model of transport monitoring application based on internet of things.

[ČH18] Alem Čolaković and Mesud Hadžialić. Internet of things (iot) : A review of enabling technologies, challenges, and open research issues. *Computer Networks*, 2018.

[CN16] Hon Fong Chong and Danny Wee Kiat Ng. Development of iot device for traffic management system. In *2016 IEEE Student Conference on Research and Development (SCOReD)*, pages 1–6. IEEE, 2016.

[CXL+14] Shanzhi Chen, Hui Xu, Dake Liu, Bo Hu, and Hucheng Wang. A vision of iot : Applications, challenges, and opportunities with china perspective. *IEEE Internet of Things journal*, 1(4) :349–359, 2014.

[Dil20] G Dileep. A survey on smart grid technologies and applications. *Renew. Energy*, 146 :2589–2625, 2020.

[DM08] Mohsen Darianian and Martin Peter Michael. Smart home mobile rfid-based internet-of-things systems and services. In *Advanced Computer Theory and Engineering, 2008. ICACTE'08. International Conference on*, pages 116–120. IEEE, 2008.

[DMOD+10] Angelika Dohr, Robert Modre-Opsrian, Mario Drobics, Dieter Hayn, and Günter Schreier. The internet of things for ambient assisted living. In *Information Technology : New Generations (ITNG), 2010 Seventh International Conference on*, pages 804–809. Ieee, 2010.

[EK08] Florian Echtler and Gudrun Klinker. A multitouch software architecture. In *Proceedings of the 5th Nordic conference on Human-computer interaction : building bridges*, pages 463–466. ACM, 2008.

[FN08] Kary Främling and Jan Nyman. Information architecture for intelligent products in the internet of things. *Beyond Business Logistics proceedings of NOFOMA*, pages 224–229, 2008.

[FPVC14] Romano Fantacci, Tommaso Pecorella, Roberto Viti, and Camillo Carlini. A network architecture solution for efficient iot wsn backhauling : challenges and opportunities. *IEEE Wireless Communications*, 21(4) :113–119, 2014.

[FT00] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Doctoral dissertation, 2000.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot) : A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7) :1645–1660, 2013.

[GC13] O. Gogliano and C. Eduardo Cugnasca. An overview of the epcglobal network. *IEEE Latin America Transactions*, 11(4) :1053–1059, June 2013.

[GCFP10] Pau Giner, Carlos Cetina, Joan Fons, and Vicente Pelechano. Developing mobile workflow support in the internet of things. *IEEE Pervasive Computing*, 9(2) :18–26, 2010.

[GGMT08] Stefano Gallotti, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. Quality prediction of service compositions through probabilistic model checking. In *International Conference on the Quality of Software Architectures*, pages 119–134. Springer, 2008.

[GLMS13] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. Cadp 2011 : a toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, 15(2) :89–107, 2013.

[GOS98] Rachid Guerraoui, Rui Oliveira, and André Schiper. Stubborn communication channels. Technical report, 1998.

[Grø08] Inge Grønbæk. Architecture for the internet of things (iot) : Api and interconnect. In *Sensor Technologies and Applications, 2008. SENSORCOMM'08. Second International Conference on*, pages 802–807. IEEE, 2008.

[GTK+10] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio. Interacting with the soa-based internet of things : Discovery, query, selection, and on-demand provisioning of web services. *IEEE transactions on Services Computing*, (3) :223–235, 2010.

[GTMW11] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. From the internet of things to the web of things : Resource-oriented architecture and best practices. *Architecting the Internet of things*, pages 97–129, 2011.

[Hay05] Simon Haykin. Cognitive radio : brain-empowered wireless communications. *IEEE journal on selected areas in communications*, 23(2) :201–220, 2005.

[HBGS07] Fei He, Luciano Baresi, Carlo Ghezzi, and Paola Spoletini. Formal analysis of publish-subscribe systems by probabilistic timed automata. In *International Conference on Formal Techniques for Networked and Distributed Systems*, pages 247–262. Springer, 2007.

[HK15] Kaoutar Hafdi and Abdelaziz Kriouile. Designing redy distributed systems. In *International Conference on Autonomic Computing (ICAC), 2015 IEEE*, pages 331–336. IEEE, 2015.

[HKK17] Kaoutar Hafdi, Abdelaziz Kriouile, and Abderahman Kriouile. Formal modeling and validation of redy architecture intended for iot applications. *International Journal of Innovative Research in Computer Science and Technology*, 5 :339–349, 07 2017.

[HKK18] Kaoutar Hafdi, Abderahman Kriouile, and Abdelaziz Kriouile. Iot redy architecture for smart grid management. *Computer and Information Science*, 11(4) :36–44, 2018.

[HKK19] Kaoutar Hafdi, Abderahman Kriouile, and Abdelaziz Kriouile. Overview on internet of things (iot) architectures, enabling technologies and challenges. *JCP*, 14(9) :557–570, 2019.

[HKP+10] Sungmin Hong, Daeyoung Kim, Sang Jun Park, Minkeun Ha, Sungho Bae, Wooyoung Jung, and Jae-Eon Kim. Snail : an ip-based wireless sensor network approach to the internet of things. *IEEE Wireless Communications*, 17(6), 2010.

[HL84] Patrick M Hodge and Raymond J Lipan. Vehicle presence loop detector, September 18 1984. US Patent 4,472,706.

[HM11] Hisakazu Hada and Jin Mitsugi. Epc based internet of things architecture. In *RFID-Technologies and Applications (RFID-TA), 2011 IEEE International Conference on*, pages 527–532. IEEE, 2011.

[KAC+16] Omprakash Kaiwartya, Abdul Hanan Abdullah, Yue Cao, Ayman Altameem, Mukesh Prasad, Chin-Teng Lin, and Xiulei Liu. Internet of vehicles : Motivation, layered architecture, network model, challenges, and future aspects. *IEEE Access*, 4 :5356–5373, 2016.

[KAK16] Meesun Kim, Hyun Ahn, and Kwanghoon Pio Kim. Process-aware internet of things : A conceptual extension of the internet of things framework and architecture. *TIIS*, 10(8) :4008–4022, 2016.

[KCL+17] Moez Krichen, Omar Cheikhrouhou, Mariam Lahami, Roobaea Alroobaea, and Afef Jmal Maâlej. Towards a model-based testing framework for the security of internet of things for smart city applications. In *International Conference on Smart Cities, Infrastructure, Technologies and Applications*, pages 360–365. Springer, 2017.

[KKA10] Fahim Kawsar, Gerd Kortuem, and Bashar Altakrouri. Supporting interaction with the internet of things across objects, time and space. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010.

[KKSF10] Gerd Kortuem, Fahim Kawsar, Vasughi Sundramoorthy, and Daniel Fitton. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1) :44–51, 2010.

[LCP+05] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, Steven Lim, et al. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7(1-4) :72–93, 2005.

[LL15] In Lee and Kyoochun Lee. The internet of things (iot) : Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4) :431–440, 2015.

[LLC+11] Jianming Liu, Xiangzhen Li, Xi Chen, Yan Zhen, and Lingkang Zeng. Applications of internet of things on smart grid in china. In *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, pages 13–17. IEEE, 2011.

[LO82] Leslie Lamport and Susan Owicki. Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems*, 4, 1982.

[LSL+11] Zhao Liqiang, Yin Shouyi, Liu Leibo, Zhang Zhen, and Wei Shaojun. A crop monitoring system based on wireless sensor network. *Procedia Environmental Sciences*, 11 :558–565, 2011.

[MEAV+18] Mirella Martínez, Anna I Esparcia-Alcázar, Tanja EJ Vos, Pekka Aho, and Joan Fons i Cors. Towards automated testing of the internet of things : Results obtained with the testar tool. In *International Symposium on Leveraging Applications of Formal Methods*, pages 375–385. Springer, 2018.

[MSDPC12] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things : Vision, applications and research challenges. *Ad Hoc Networks*, 10(7) :1497–1516, 2012.

[MT08a] Radu Mateescu and Damien Thivolle. A model checking language for concurrent value-passing systems. In *International Symposium on Formal Methods*, pages 148–164. Springer, 2008.

[MT08b] Radu Mateescu and Damien Thivolle. A model checking language for concurrent value-passing systems. In *International Symposium on Formal Methods*, pages 148–164. Springer, 2008.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[Puj06] Guy Pujolle. An autonomic-oriented architecture for the internet of things. In *Modern Computing, 2006. JVA'06. IEEE John Vincent Atanasoff 2006 International Symposium on*, pages 163–168. IEEE, 2006.

[PYC+18] Samira Pouyanfar, Yimin Yang, Shu-Ching Chen, Mei-Ling Shyu, and SS Iyengar. Multimedia big data analytics : A survey. *ACM Computing Surveys (CSUR)*, 51(1) :1–34, 2018.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on programming*, pages 337–351. Springer, 1982.

[Ray18] Partha Pratim Ray. A survey on internet of things architectures. *Journal of King Saud University-Computer and Information Sciences*, 30(3) :291–319, 2018.

[REC15] Karen Rose, Scott Eldridge, and Lyman Chapin. The internet of things : An overview. *The Internet Society (ISOC)*, pages 1–50, 2015.

[RLSS10] Ragunathan Raj Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems : the next computing revolution. In *Proceedings of the 47th Design Automation Conference*, pages 731–736. ACM, 2010.

[RMJPC15] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke. Middleware for internet of things : a survey. *IEEE Internet of things journal*, 3(1) :70–95, 2015.

[RSL17] Maryam Rahmani, Junsup Song, and Moonkun Lee. Prism : A knowledge engineering tool to model collective behaviors of real-time iot systems. In *PrOse@ PoEM*, 2017.

[RSMCP16] Alessandra Rizzardi, Sabrina Sicari, Daniele Miorandi, and Alberto Coen-Porisini. Aups : An open source authenticated publish/subscribe system for the internet of things. *Information Systems*, 62 :29–41, 2016.

[SDG13] Andrey Somov, Corentin Dupont, and Raffaele Giaffreda. Supporting smart-city mobility with cognitive internet of things. In *2013 Future Network & Mobile Summit*, pages 1–10. IEEE, 2013.

[SDS19] Amber A Smith-Ditizio and Alan D Smith. Using rfid and barcode technologies to improve operations efficiency within the supply chain. In *Advanced Methodologies and Technologies in Business Operations and Management*, pages 1277–1288. IGI Global, 2019.

[SF00] Jon Siegel and Dan Frantz. *CORBA 3 fundamentals and programming*, volume 2. John Wiley & Sons New York, NY, USA :, 2000.

[SGFW10] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. Vision and challenges for realising the internet of things. *Cluster of European*

Research Projects on the Internet of Things, European Commision, 3(3) :34–36, 2010.

[SHB14] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). Technical report, 2014.

[She12] Zach Shelby. Constrained restful environments (core) link format. Technical report, 2012.

[SKG+09] Patrik Spiess, Stamatis Karnouskos, Dominique Guinard, Domnic Savio, Oliver Baecker, Luciana Moreira Sá De Souza, and Vlad Trifa. Soa-based integration of the internet of things in enterprise services. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 968–975. IEEE, 2009.

[SKP+11] Hans Schaffers, Nicos Komninos, Marc Pallot, Brigitte Trousse, Michael Nilsson, and Alvaro Oliveira. Smart cities and the future internet : Towards cooperation frameworks for open innovation. In *The future internet assembly*, pages 431–446. Springer, 2011.

[SKS17] Snehal R Shinde, AH Karode, and Dr SR Suralkar. Review on iot based environment monitoring system. *International Journal of Electronics and Communication Engineering and Technology*, 8(2), 2017.

[SRS+16] João Santos, Joel JPC Rodrigues, Bruno MC Silva, João Casal, Kashif Saleem, and Victor Denisov. An iot-based mobile gateway for intelligent personal assistants on mobile health environments. *Journal of Network and Computer Applications*, 71 :194–204, 2016.

[SSP+15] Chayan Sarkar, Akshay Uttama Nambi SN, R Venkatesha Prasad, Abdur Rahim, Ricardo Neisse, and Gianmarco Baldini. Diat : A scalable distributed architecture for iot. *IEEE Internet of Things journal*, 2(3) :230–239, 2015.

[ST17] Biljana L Risteska Stojkoska and Kire V Trivodaliev. A review of internet of things for smart home : Challenges and solutions. *Journal of Cleaner Production*, 140 :1454–1464, 2017.

[Sta14] John A Stankovic. Research directions for the internet of things. *IEEE Internet of Things Journal*, 1(1) :3–9, 2014.

[TATM15] H Tschofenig, J Arkko, D Thaler, and D McPherson. Architectural considerations in smart object networking. Technical report, 2015.

[TM17] Antero Taivalsaari and Tommi Mikkonen. A roadmap to the programmable world : software challenges in the iot era. *IEEE Software*, (1) :72–80, 2017.

[TMD09] Richard N Taylor, Nenad Medvidovic, and Eric M Dashofy. *Software architecture : foundations, theory, and practice*. Wiley Publishing, 2009.

[TVS07] Andrew Tanenbaum and Maarten Van Steen. *Distributed systems*. Pearson Prentice Hall, 2007.

[VAAV18] M Vardhana, N Arunkumar, Enas Abdulhay, and PV Vishnuprasad. Iot based real time trafic control using cloud computing. *Cluster Computing*, pages 1–10, 2018.

[VGVS05] Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen. Cyclon : Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2) :197–217, 2005.

[VSB12] Sauro Vicini, Alberto Sanna, and Sara Bellini. A living lab for internet of things vending machines. In *The impact of virtual, remote, and real logistics labs*, pages 35–43. Springer, 2012.

[VZS10] Antonio J Jara Valera, Miguel A Zamora, and Antonio FG Skarmeta. An architecture based on internet of things to support mobility and security in medical environments. In *2010 7th IEEE Consumer Communications and Networking Conference*, pages 1–5. IEEE, 2010.

[WADX15] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. The internet of things—a survey of topics and trends. *Information Systems Frontiers*, 17(2) :261–274, 2015.

[WBC+09] Evan Welbourne, Leilani Battle, Garrett Cole, Kayla Gould, Kyle Rector, Samuel Raymer, Magdalena Balazinska, and Gaetano Borriello. Building the internet of things using rfid : the rfid ecosystem experience. *IEEE Internet computing*, 13(3), 2009.

[WGB99] Mark Weiser, Rich Gold, and John Seely Brown. The origins of ubiquitous computing research at parc in the late 1980s. *IBM systems journal*, 38(4) :693–696, 1999.

[WLL+10] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun, and Hui-Ying Du. Research on the architecture of internet of things. In *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, volume 5, pages V5–484. IEEE, 2010.

[WWZ+16] Shiyong Wang, Jiafu Wan, Daqiang Zhang, Di Li, and Chunhua Zhang. Towards smart factory for industry 4.0 : a self-organized multi-agent system with big data based feedback and coordination. *Computer Networks*, 101 :158–168, 2016.

[XXL18] Li Da Xu, Eric L Xu, and Ling Li. Industry 4.0 : state of the art and future trends. *International Journal of Production Research*, 56(8) :2941–2962, 2018.

[YAA17] Kumar Yelamarthi, Md Sayedul Aman, and Ahmed Abdelgawad. An application-driven modular iot architecture. *Wireless Communications and Mobile Computing*, 2017, 2017.

[YTN16] Shingo Yamaguchi, Shoki Tsugawa, and Kazuya Nakahori. An analysis system of iot services based on agent-oriented petri net pn2. In *2016 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, pages 1–2. IEEE, 2016.

[YXM+14] Geng Yang, Li Xie, Matti Mäntysalo, Xiaolin Zhou, Zhibo Pang, Li Da Xu, Sharon Kao-Walter, Qiang Chen, and Li-Rong Zheng. A health-iot platform based on the integration of intelligent packaging, unobtrusive bio-sensor, and intelligent medicine box. *IEEE transactions on industrial informatics*, 10(4) :2180–2191, 2014.