Modèles partagés et infrastructures ouverte pour l'internet des objets de la ville Intelligente

Laurent Lemke

Communauté
UNIVERSITÉ Grenoble Alpes

THESE

Pour obtenir le grade de

DOCTEUR DE la Communauté UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Informatique - Internet des objets**

Arrêté ministériel : 25 Mai 2016

Presentée par

Laurent Lemke

Thèse dirigée par **Didier Donsez**
et co-encadrée par **Florence Maraninchi et Gilles Privat**

preparée au sein **LIG, Verimag et Orange Labs**
et de **l'École Doctorale Mathématiques, Sciences et Technnologies de l'Information, Informatique**

Modèles partagés et infrastructure ouverte pour l'internet des objets de la ville intelligente

Thèse soutenue publiquement le **06 Juin 2017**,
devant le jury composé de :

M. Michael MRISSA
Professeur, Université de Pau et des pays de l'Adour, President et Examinateur

M. Robert DE SIMONE
Directeur de recherche, INRIA Sophia-Antipolis, Rapporteur

M. Thierry MONTEIL
Maitre de conférences (HDR), INSA Toulouse, Rapporteur

Remerciements

En premier lieu, je souhaite remercier les différents membres du jury pour l'intérêt qu'ils portent à mes travaux ainsi que le temps qu'ils ont consacrés à relire mon manuscrit de thèse. Merci à Robert De Simone et à Thierry Monteil d'être les rapporteurs de ma thèse et merci à Michael Mrissa d'en être l'examinateur.

Je tiens à adresser de chaleureux remerciements à mes encadrants qui sont Florence Maraninchi, Didier Donsez et Gilles Privat. Les points de vue de tous n'avaient pas dans tout le temps une intersection commune mais c'est ce qui, à mon sens, a fait la richesse de cette thèse. Être entouré de personnes aussi compétentes a été une chance et cela m'a permis à la fois d'apprendre énormément de choses sur des sujets aussi vastes que variés mais également, et avant tout, de mener à bien mes travaux. Je tiens à souligner particulièrement leurs précieux petits coups de boost au moral lors des phases de doute et de démotivation, c'était vital !

J'ai une pensée à tous mes collègues d'Orange, de Verimag et de l'équipe ERODS qui ont tous contribué quelque part à cette thèse par l'ambiance de travail et les différentes discussions, de la plus anodine à la plus enrichissante, que nous avons pu avoir ensemble. En particulier merci à Quyet, Yuliia, Hamza et Denis avec qui j'ai partagé mon bureau et où l'ambiance a toujours été très agréable. Bon courage aux deux derniers pour la fin de votre thèse ! Merci également à mes collègues ERODS qui m'ont ré-accueillis dans leur équipe avec bienveillance et qui m'ont fait de nouveau goûter au joie du restaurant universitaire. En parlant d'ambiance, merci à mes collègues d'Orange et aux membres du bien nommé "G10" qui m'ont rappelés ô combien c'était important de devoir se détendre autour d'une bonne (ou plusieurs...) bière(s).

Ce manuscrit n'aurait pas pu voir le jour sans le soutien inconditionnel de mes amis dellois, belfortain, grenoblois, breton ou venant d'autres obscures contrées. Je ne suis pas certain que tous aient compris les tenants et les aboutissements de mon travail de trois années, mais leur soulagement suite à l'obtention de ce doctorat est à la hauteur du soutien moral qu'ils m'ont apportés. Un très grand merci à vous !

Une pensée émue à mon ami Rodmar : tu vois, j'ai fini cette "connerie de thèse" comme tu disais avec ton air amusé. Je me rappellerai toujours de nos innombrables discussions à propos de nos différentes cultures, de nos "conneries" communes qu'on pouvait sortir ainsi que de plein d'autres moments. Merci mon ami pour ton amitié.

Je ne remercierai jamais assez mes parents, mon frère et ma belle-soeur qui m'ont accompagnés à tout instant de la thèse. Ce que leur présence m'apporte au quotidien est inestimable.

Enfin, je remercie du fond du coeur la personne la plus importante à mes yeux, à savoir ma chérie. Outre le fait qu'il s'agisse de la personne ayant relu le plus de fois cette thèse, son soutien a été sans faille tout au long de mon travail. Je me rends compte de la chance que j'ai d'être à tes côtés au quotidien. Merci du fond du coeur "mi love", je t'aime. C'est la dernière soirée que je passe sur ce manuscrit ;-) !

Résumé

Les villes contemporaines font face à de nombreux enjeux : énergétiques, écologiques, démographiques ou encore économiques. Pour y répondre, des moyens technologiques sont mis en place dans les villes via l'utilisation de capteurs et d'actionneurs. Ces villes sont dites intelligentes.

Actuellement, les villes intelligentes sont opérées d'acteurs qui ne partagent ni leurs données de capteurs ni l'accès à leurs actionneurs. Cette situation est dite verticale : chaque opérateur déploie ses propres capteurs et actionneurs et possède sa propre infrastructure informatique hébergeant ses applications. Cela conduit à une redondance de l'infrastructure et à des applications ad-hoc pour superviser et contrôler un domaine de la ville.

La tendance est d'aller vers une situation dite horizontale via l'utilisation d'une plateforme de médiation ouverte et partagée. Les données de capteurs et les accès aux actionneurs sont mutualisés au sein de ce type de plateforme, permettant leur partage entre les différents acteurs. Les coûts d'infrastructure et de développement s'en trouvent alors réduits.

Cette thèse s'inscrit dans ce contexte d'horizontalisation, au sein d'une plateforme ouverte et partagée, dans laquelle nous proposons : 1) une couche d'abstraction pour le contrôle et la supervision de la ville, 2) un mécanisme de contrôle de concurrence gèrant les cas de conflits, 3) un mécanisme de coordination favorisant la réutilisation des actionneurs, 4) une implémentation de notre travail par une preuve de concept.

L'abstraction que nous proposons se base sur des modèles issus des systèmes réactifs. Ils ont pour objectif d'être génériques et représentent l'invariant de la ville intelligente : les éléments physiques. Ils permettent aux applications de contrôler et superviser la ville.

Pour faciliter le développement d'applications nous uniformisons l'interface de nos modèles. Ces applications pouvant avoir des contraintes temps réel, particulièrement celles qui ont des objectifs de contrôle, nous proposons de tirer parti de l'architecture distribuée de ce type de plateforme.

Compte-tenu du partage des actionneurs, nous avons identifié que des conflits peuvent survenir entre les applications. Nous proposons un mécanisme de contrôle de concurrence pour traiter ces cas de conflits.

Nous avons également identifié qu'un mécanisme de coordination doit être offert aux applications souhaitant effectuer atomiquement des opérations de contrôle. Un tel mécanisme favorise la réutilisation des actionneurs présents dans la ville.

Enfin, nous avons implémenté nos propositions autour d'une preuve de concept, comprenant plusieurs cas d'usages, permettant de démontrer notre travail.

Table des matières

Chapitre 1

Introduction

Enjeux dans la ville intelligente D'après l'United Nations Population Fund, plus de la moitié de la population mondiale vit actuellement dans des milieux urbains [1]. D'ici 2030, ce taux devrait augmenter et atteindre 60% de la population. Cette croissance amène les villes à faire face à de nouveaux enjeux démographiques.

Les milieux urbains doivent également prendre en compte les défis écologiques. Un rapport de l'Intergovernmental Panel on Climate Change [2] a estimé que les villes généraient environ 70% des émissions de CO2 dans le monde et consommaient plus 76% des énergies globales.

De surcroît, ces enjeux écologiques amènent de nouveaux enjeux énergétiques et économiques. A titre d'exemple, l'International Energy Agency [3] a évalué que l'éclairage public est responsable de 19% de l'électricité globale utilisée dans le monde et engendre environ 6% des émissions de gaz à effet de serre. De plus, l'éclairage public peut compter jusqu'à hauteur de 40% dans la facture d'électricité d'une ville.

Pour répondre à ces nombreux défis, des moyens technologiques sont mis en place dans les villes via l'utilisation de capteurs et d'actionneurs connectés en réseau. Compte-tenu de leurs capacités technologiques, ces villes sont qualifiées de villes intelligentes. Il a été proposé également de tenir compte d'autres aspects (éducation, gouvernance, etc.) [38, 116] pour qualifier une ville intelligente. Pour les besoins de notre travail nous nous concentrons sur l'aspect technologique.

D'une situation verticale à l'horizontalisation Les villes intelligentes sont actuellement opérées par des acteurs qui ne partagent ni leur données de capteurs, ni l'accès à leurs actionneurs. Chaque acteur possède sa propre infrastructure hébergeant ses applications et déploie ses propres capteurs et actionneurs pour ses besoins.

La figure 1.1 illustre ce constat actuel que nous appelons une situation verticale dans la ville intelligente. Quatre acteurs sont illustrés, à savoir l'acteur chargé de la gestion des déchets, du trafic routier, de l'éclairage public et de l'énergie électrique. Chacun d'eux déploie ses capteurs et actionneurs pour ses besoins spécifiques : superviser le taux de remplissage des poubelles, fluidifier le trafic routier, éclairer de manière efficiente la route et optimiser le stockage et la consommation électrique dans la ville.

1. http://www.unfpa.org
2. http://www.ipcc.ch
3. https://www.iea.org/

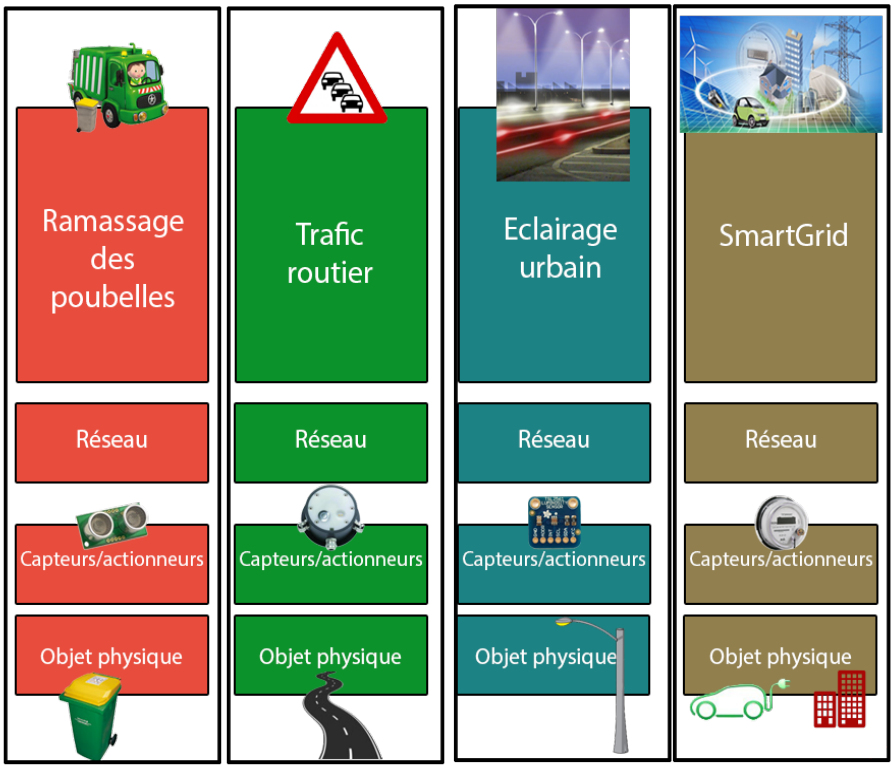


FIGURE 1.1 – Situation verticale dans la ville intelligente.

Pour mutualiser les capteurs et actionneurs des différents acteurs, une tendance est d'aller vers une situation horizontale. Une situation horizontale se traduit par l'utilisation d'une plateforme de médiation ouverte et partagée par les acteurs de la ville intelligente. Elle permet à ceux-ci de partager leurs données de capteurs ainsi que les accès à leurs actionneurs. Les coûts d'infrastructure, de développement et de déploiement des capteurs et actionneurs s'en trouvent réduits.

Nous illustrons ce principe d'horizontalisation dans la figure 1.2 suivante. Dans cet exemple, l'utilisation d'une plateforme horizontale permettrait à l'acteur chargé de l'éclairage public de réutiliser les données des capteurs de l'acteur du trafic routier pour allumer ou éteindre les lampadaires en fonction de la présence ou non de voitures sur la chaussée.

Objectifs du travail Le but de notre travail est d'aboutir à une situation horizontale dans la ville intelligente en promouvant une plateforme ouverte et partagée.

Pour parvenir à cette horizontalisation, de nombreuses plateformes ont été proposées (cf. section 3.2). Nous avons identifié que ces plateformes sont principalement destinées à être utilisées par des applications de supervision. En effet, le partage des données des capteurs est une préoccupation majeure des plateformes actuelles. Cependant, très peu d'entre elles adressent la problématique du partage d'actionneurs.

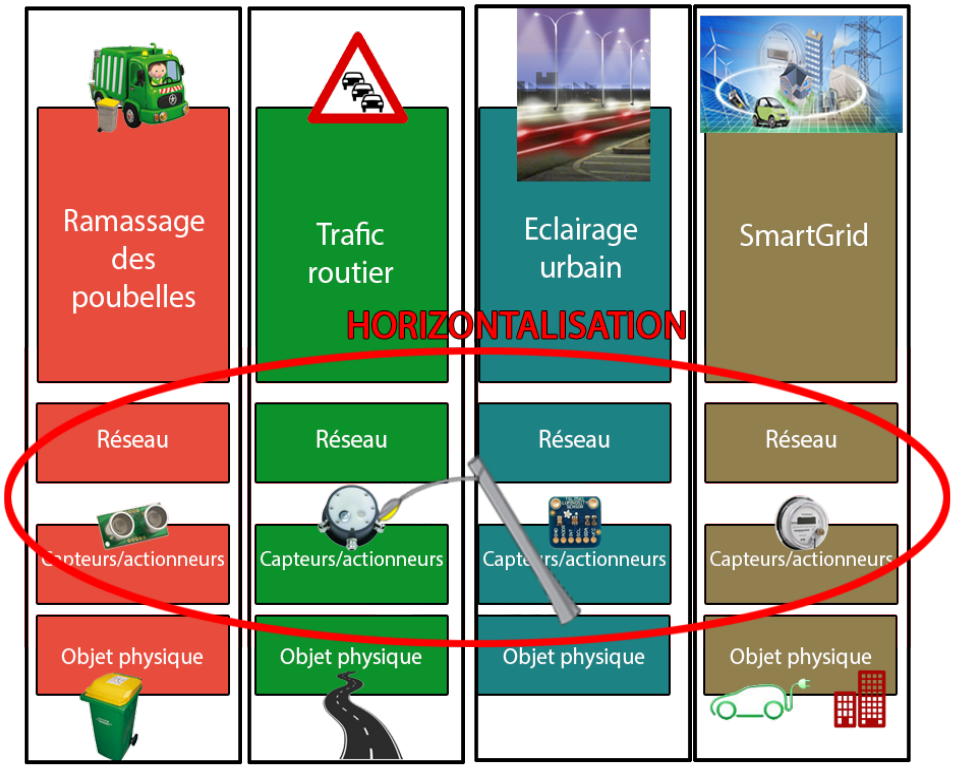


FIGURE 1.2 – Principe de l'horizontalisation.

Nous avons choisi d'adresser cette problématique dans notre travail afin qu'une plateforme ouverte et partagée soit utilisée par des applications de contrôle. Pour promouvoir l'utilisation de la plateforme par des applications de contrôle, nous avons identifié qu'un mécanisme de coordination favorisant la réutilisation des actionneurs et un mécanisme de contrôle de concurrence gérant les cas de conflits devaient exister. Par ailleurs, nous avons également identifié que ces applications de contrôle pouvaient avoir des contraintes temps-réel et qu'il fallait en tenir compte dans l'infrastructure de la plateforme.

Enfin, pour poursuivre cet objectif d'ouverture et de partage de la plateforme par les acteurs, nous avons également choisi de faciliter le développement d'applications et l'intégration de nouveaux équipementiers dans la ville intelligente. Dans ce contexte, nous avons identifié qu'une couche d'abstraction était nécessaire pour contrôler et superviser la ville.

Contributions La couche d'abstraction que nous proposons se base sur des modèles, issus des systèmes réactifs. Il s'agit d'automates à états finis décrits à l'aide du langage Argos permettant une représentation générique des éléments physiques présents dans la ville qui sont les invariants de la ville intelligente. A partir de ces modèles, nous avons effectué de la génération de code vers des ressources REST qui

sont utilisées par les applications de manière uniforme pour contrôler et superviser la ville.

Ces applications pouvant avoir des contraintes temps-réel, particulièrement celles qui ont des objectifs de contrôle, nous avons identifié qu'une infrastructure incluant du edge computing était nécessaire et qu'une infrastructure de cloud computing ne suffisait pas à elle seule.

Pour empêcher que des conflits surviennent entre les applications, engendrés par le partage des actionneurs, nous avons créé un mécanisme de contrôle de concurrence. Nous proposons également un mécanisme de coordination pour pouvoir effectuer atomiquement des opérations de contrôle. Nous avons validé le protocole de l'algorithme de coordination que nous avons inventé afin de détecter que celui-ci ne comportait pas de potentiel bug. Dans un soucis d'uniformisation des interactions avec les applications, nous avons choisi d'associer des ressources aux mécanismes de coordination et contrôle de concurrence.

Enfin, pour évaluer notre travail nous l'avons implémenté autour d'une preuve de concept comprenant plusieurs cas d'usage. Nous avons créé plusieurs scénarios d'applications sur la base d'un exemple miniature de ville intelligente. Un simulateur a été créé afin de simuler les capteurs et actionneurs ainsi que les latences réseaux et pannes potentielles.

Chapitre 2

Background

Ce chapitre vise à décrire les éléments nécessaires à la compréhension des concepts techniques présents dans notre travail.

La section 2.1 dresse un état des lieux des éléments à prendre en compte lors de la conception d'une plateforme partagée pour l'internet des objets de la ville intelligente. Nous détaillons le style d'architecture que nous avons retenu et expliquons son adoption dans l'internet des objets (section 2.1.1). Puis, nous décrivons les deux types d'infrastructures existantes en nous positionnant par rapport à celles-ci (section 2.1.2). Enfin, nous donnons des détails à propos des réseaux de capteurs existants dans l'internet des objets (section 2.1.3) permettant de mettre en oeuvre une plateforme partagée.

En section 2.2, nous présentons les modèles formels que nous avons utilisés. Ces modèles sont utilisés dans les systèmes réactifs (cf. section 2.2.1). Puis, nous décrivons les modèles asynchrones et l'outil SPIN (cf. section 2.2.2) que nous avons utilisés pour valider l'algorithme de gestion de conflits que nous avons proposé.

Enfin, nous décrivons en section 2.3 les modèles sémantiques sur lesquels nos travaux s'appuient. Après avoir rappelé les enjeux de ces modèles par rapport à notre travail (cf. section 2.3.1), nous détaillons la notion d'ontologies ainsi que la conception et l'utilisation de ces dernières (cf. section 2.3.2).

Sommaire

2.1 Éléments à prendre en compte dans la conception d'une plateforme partagée

2.1.1 Style d'architecture

Une architecture logicielle peut se définir comme un ensemble de composants logiciels qui interagissent ensemble afin de constituer un système informatique [132].

Un style d'architecture sert d'abstraction pour définir des architectures logicielles. Il contient des principes de conception et des contraintes appliquées aux composants logiciels et à leurs interactions [151] afin que le système remplisse certains critères (passage à l'échelle, fiabilité, performance, etc.). Un grand nombre de styles d'architecture existent dans la littérature [61] ("pipe and filter", "event bus", "blackboard", etc.).

Dans notre contexte, une plateforme pour la ville intelligente doit notamment posséder trois qualités : être fiable compte-tenu de la criticité de certaines applications l'utilisant, passer à l'échelle compte-tenu de l'envergure et de la complexité des domaines d'applications considérés, permettre que de nouvelles applications puissent être intégrées facilement en utilisant la plateforme.

En ce sens, le style d'architecture REST [51] est un choix idéal puisqu'il promeut ces trois critères d'où son utilisation dans notre travail.

2.1.1.1 Principes du style d'architecture REST

REST est un style d'architecture à couplage faible conçu pour les systèmes distribués. Les critères énumérés dans ce style architectural sont issus d'une généralisation de l'architecture du web. Ainsi, cela permet à des systèmes de passer à l'échelle et d'être fiables tout en étant faiblement couplés avec les clients qui les utilisent.

REST propose notamment trois contraintes architecturales :

— Les interactions suivent un modèle client/serveur où clients et serveurs sont découplés, ce qui leur permet d'être plus scalables.

— Les interactions sont sans état : chaque requête envoyée par un client contient toutes les informations nécessaires pour être comprise par le serveur. Un serveur n'a ainsi pas besoin de stocker le contexte de chaque client (exemple : une session d'un client). Cela permet un passage à l'échelle plus facile des systèmes et améliore leur fiabilité en facilitant la reprise après panne.

— Les interactions ont lieu au travers d'une interace uniforme : REST définit la notion de *ressources* qui sont manipulées de manière uniforme par les clients. Cela simplifie les interactions effectuées par les clients et réduit leur couplage avec le serveur. Typiquement, l'interface uniforme se reposera sur le protocole utilisé pour les interactions, par exemple les verbes GET, POST, PUT, DELETE dans HTTP [50].

Une ressource est n'importe quoi pouvant être nommé et stocké par un ordinateur (un document, un objet virtuel, etc.), il s'agit de l'abstraction prépondérante

promulguée dans les contraintes REST. Une ressource possède un état qui correspond à sa représentation, appelée *représentation d'une ressource*. Lorsqu'un client interagit avec une ressource stockée sur un serveur (exemple : ressource "Compte Bancaire"'), il obtient sa représentation (exemple : montant et numéro de compte du compte) qui peut varier dans le temps. Il est à noter que cette interaction se fera de la même manière pour l'accès à n'importe quelle autre ressource grâce au principe d'interface uniforme.

Les ressources REST doivent être identifiées de manière unique afin que les clients puissent interagir avec celles-ci, notamment grâce à un Uniform Resource Identifier (URI).

Les ressources REST doivent également être liées les unes aux autres, où les liens entre les ressources représentent une relation entre elles. Par exemple, une ressource "Compte bancaire" serait liée à une ressource "Personne" pour indiquer qu'il s'agit du propriétaire du compte bancaire en question. Ces liens sont relatifs à la propriété HATEOAS (Hypermedia As The Engine Of Application) dans REST. Elle permet à des clients de naviguer de ressource en ressource (au travers des liens qui les unissent) sans devoir interroger le serveur et par conséquent de manière faiblement couplée.

Il est à noter que le style architectural REST ne se limite pas uniquement à des interactions basées sur le protocole de communication HTTP. Ces interactions peuvent se faire, par exemple, via le protocole MQTT [39]. A titre d'exemple, l'organisme de standardisation oneM2M [11] qui propose une architecture pour le M2M et l'internet des objets, prévoit que les ressources REST décrites au sein de son architecture puissent être interrogées au travers des protocoles MQTT [126], CoAP [124] ou HTTP [125].

Dans notre travail, nous avons choisi d'adhérer aux principes REST pour les raisons que nous avons évoquées précédemment. Nous proposons une architecture orientée ressources (cf. section 4.4) qui est une déclinaison des principes REST se focalisant sur les ressources et leur modélisation [145]. Dans notre implémentation, nous avons choisi le protocole CoAP pour les interactions entre les applications et les ressources (cf. section 7.2.1).

2.1.1.2 Adoption de REST dans l'internet des objets

L'utilisation du style architectural REST [51] est devenue un standard de fait dans le domaine de l'Internet des objets. Par exemple, l'organisme de standardisation oneM2M promeut son utilisation dans leur proposition d'architecture pour le M2M et l'internet des objets. De même, l'IETF via son groupe de travail "Constrained RESTful Environments" (CoRE) contribue à l'adoption de ce style architectural dans l'IoT en proposant le protocole CoAP [152] pour les environnements très contraints. En effet, CoAP a été conçu pour des devices, c'est-à-dire le matériel comprenant un ou plusieurs capteurs et/ou actionneurs, contraints en énergie et ayant des ressources matérielles (RAM, CPU, etc.) limitées. Ce protocole est similaire à HTTP puisqu'il utilise les mêmes verbes (GET, PUT, POST, etc). Il comporte également des nouveautés par rapport à celui-ci : possibilité de s'abonner à des

ressources (verbe OBSERVE), messages de taille faible, etc.

En lien avec l'utilisation de REST dans l'internet des objets, la notion de "Web of Things" [71] est apparue en 2011. Le Web of Things se définit par l'utilisation de mini-serveurs web embarqués au sein de devices intégrant des capteurs et actionneurs. Ainsi, Les devices peuvent être interrogés au travers des ressources REST qu'ils exposent afin de collecter des données de capteurs et/ou d'envoyer des ordres aux actionneurs.

D'autre part, comme le montre cette étude [110], la très grande majorité des plateformes pour l'IoT expose des ressources REST aux applications puisque sur les 39 plateformes évaluées par l'auteur seules quatre d'entre elles n'exposent pas de ressources REST.

L'un des éléments de réponse au choix de REST est que les développeurs d'applications de l'Internet des objets considèrent plus aisée la conception d'applications basées sur l'utilisation de services web REST plutôt que sur des services web WS [70], bien que ces derniers possèdent d'autres atouts notamment en terme de sécurité [130].

2.1.2 Infrastructures orientées Cloud et "Edge computing"

Le cloud computing peut se définir comme le regroupement de services sur internet fournis par du matériel et des logiciels hébergés au sein de datacenters [16]. Ces services peuvent être utilisés pour divers objectifs : stocker des données, utiliser des services applicatifs, etc. Les datacenters hébergeant ces services sont situés dans le "cloud" qui peut être privé ou public. Un cloud public permet de fournir des services qui sont partagés par tous, au contraire d'un cloud privé qui n'est dédié qu'à certains utilisateurs.

Compte-tenu du matériel hébergé en son sein, une infrastructure de cloud computing donne l'illusion que les ressources matérielles sont infinies. Ainsi, l'élasticité est l'une des caractéristiques principale du cloud computing permettant le passage à l'échelle de système utilisant ce type d'infrastructure. Nous avons pu noter que de nombreuses plateformes pour l'internet des objets et la ville intelligente utilisent une infrastructure cloud (cf. section 3.2).

Cependant, l'élasticité d'une infrastructure cloud a un prix, puisqu'elle cache la localisation de ces serveurs. Ces derniers sont hébergés au sein de centres de données pouvant être éloignés physiquement des consommateurs de services, à savoir les applications. De ce fait, des latences réseaux variables peuvent apparaître lors des interactions entre les applications et les services hébergés au sein du cloud.

Une infrastructure de "edge computing" permet de maîtriser ces latences fluctuantes. Le edge computing ne se substitue pas au cloud computing mais est complémentaire de celui-ci. Le edge computing vise notamment à fournir des unités de calcul au plus proche des applications consommatrices de services, permettant de réduire et maîtriser les latences de ces communications [59].

Nous illustrons ce principe en figure 2.1 en décrivant une vue d'ensemble d'une infrastructure de cloud computing avec (partie droite) et sans edge computing (partie

gauche). Cette figure nous permet de montrer qu'une infrastructure de edge computing est complémentaire d'une infrastructure cloud puisque nous retrouvons dans les deux parties de la figure des serveurs hébergés dans le cloud. Les communications, dans une infrastructure edge computing, ne remontent pas nécessairement jusqu'à l'infrastructure permettant ainsi de maîtriser les latences de communications entre les applications et les services.

Par ailleurs, une infrastructure incluant du edge computing est tolérante aux pannes puisqu'elle est intrinsèquement distribuée et peut supporter des pannes matérielles.

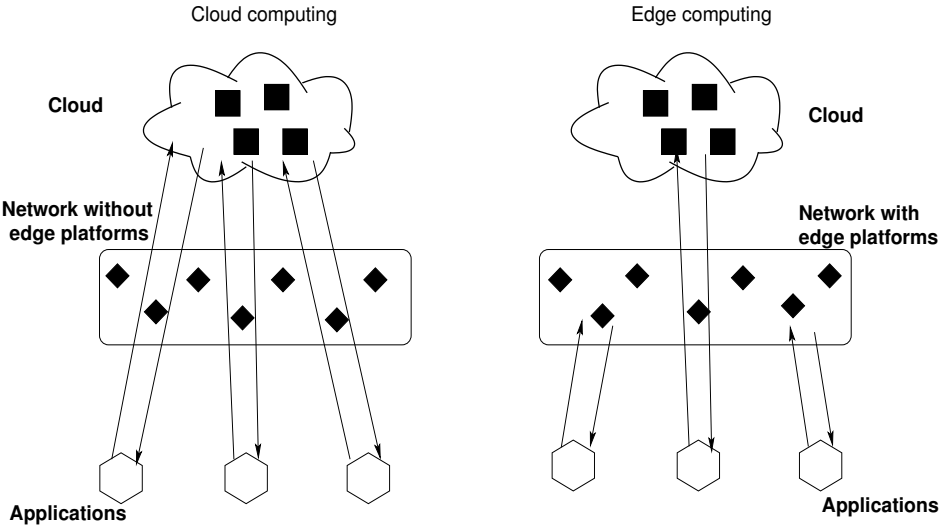


FIGURE 2.1 – Vue d'ensemble d'une infrastructure de cloud computing (partie gauche) et de edge computing (partie droite). Les flèches représentent des communications réseaux. Les losanges représentent les ressources matérielles situées en bord de réseau pouvant constituer l'infrastructure edge. Les grands carrés noirs représentent une multitude de serveurs formant le cloud. Les hexagones font références à des applications communiquant avec des services.

Dans le contexte d'une plateforme partagée pour la ville intelligente, une infrastructure edge computing nous semble requise pour satisfaire les contraintes des applications temps-réel. Nous verrons plus en détails dans les sections 4.1 et 4.3 comment se positionne notre travail par rapport à ce type d'infrastructure ainsi que la qualité de service que nous pouvons espérer par rapport à celle-ci.

2.1.3 Réseaux de capteurs et d'actionneurs

Un ensemble de capteurs et d'actionneurs transférant et recevant des données au travers d'un réseau est appelé un réseau de capteurs et d'actionneurs.

Il s'agit d'un domaine de recherche actif où il existe notamment deux catégories d'architecture : multi-sauts et en étoile. Les réseaux de capteurs multi-sauts occupent une place prépondérante dans la littérature de ce domaine de recherche. Cependant

de par leur complexité ils ne sont pas industrialisés et n'ont vu le jour qu'au travers d'expérimentations et de prototypes. Au contraire, les réseaux en étoile sont ceux qui sont actuellement déployés compte tenu de leur simplicité de mise en oeuvre. Parmi les différents réseaux suivant cette architecture, nous pouvons citer les réseaux Low-Power Wide-Area (LPWA) tel que LoRa [10], Sigfox [157] ou plus traditionnellement les réseaux cellulaires ainsi que leur évolution tel que LTE-M [89].

Afin de transférer leurs données, les capteurs utilisent des protocoles réseaux qui varient d'un réseau de capteurs à un autre en raison de la multitude de protocoles existants. De même, suivant les contraintes en énergie des capteurs, le protocole réseau utilisé peut varier. Par exemple, un protocole incluant une pile IP (CoAP [78], MQTT [121], etc.) sera adopté par des devices moins contraints en énergie que d'autres protocoles (enOCean [9], LoRa [10], etc.). De nombreux standards relatifs à ces protocoles réseaux existent dans l'internet des objets (Bluetooth Low Energy [64], 6LoWPAN [153], Zigbee [12], CoAP [78], etc.). Cependant, aucun de ces standards ne prédomine.

Au sein de la ville intelligente, il faut tenir compte du fait qu'il existe à la fois des protocoles réseaux filaires et sans-fil dans les réseaux de capteurs. Typiquement, les protocoles filaires (BACnet [112], LonTalk [102], etc.) sont utilisés par certains types d'équipements de contrôle, tels que les feux de circulation. Cependant, leurs portées sont très limitées et ils sont plus coûteux et plus difficiles à déployer. Au contraire, les protocoles sans-fil [5] ont une portée allant de quelques mètres à plusieurs kilomètres et sont plus simples et rapides à déployer.

Dans le contexte de notre travail qui s'insère dans une plateforme partagée pour le contrôle et la supervision de la ville intelligente, les principaux critères relatifs aux réseaux de capteurs que nous devons considérer sont les suivants :

— Garantir que des réseaux de capteurs offrent une latence faible et bornée, notamment pour permettre l'émergence des applications de contrôle temps-réel (cf. section 4.3).

— Fiabilité notamment dans le cadre de supervision ou de contrôle temps-réel.

Ainsi, les réseaux de capteurs qui nous semblent les plus appropriés dans le contexte de nos travaux sont ceux reposant sur des protocoles utilisant des bandes de fréquence réglementées telles que la LTE [3] ou LTE-Advanced [2] du 3GPP. En effet, ils offrent une meilleure garantie de latence réseaux que les protocoles opérant sur des bandes non-réglementées (LoRa, Sigfox, etc.) qui communiquent sur des bandes de fréquence non-licenciées (ISM) et donc partagées par tous les opérateurs qu'ils soient publics ou privés. De même, les protocoles filaires nous semblent appropriés au regard des deux critères cités. Nous les considérons et les faisons apparaître dans notre exemple conducteur (cf. section 4.2.1.2).

2.2 Modèles formels du temps et de la concurrence

Les modèles formels sont utilisés dans cette thèse pour deux objectifs : à la fois pour représenter les entités génériques dans la ville (cf. section 5.1.2) en tenant

compte de leurs comportements et de leurs propriétés temporelles, mais également pour valider l'algorithme de gestion de conflits que nous proposons (cf. section 6.4.3).

2.2.1 Modèle utilisé dans les systèmes réactifs

2.2.1.1 Les systèmes réactifs

On peut classifier les systèmes informatiques en trois catégories [77]. Les systèmes *transformationnels* produisent des sorties en fonction d'entrées puis se terminent. Un compilateur est un exemple de système transformationnel, il transforme du code haut niveau en code bas niveau puis se termine.

Les systèmes *interactifs* produisent des sorties en fonction d'entrées reçues tout au long de leur fonctionnement et ne se terminent pas forcément. Le rythme de l'interaction est déterminé par le système lui-même. Par exemple, un shell BASH qui reçoit des commandes et affiche le résultat de celles-ci est un exemple de système interactif. Dans cet exemple, si le résultat d'une commande prend du temps à s'afficher, l'utilisateur devra attendre avant de lancer sa prochaine commande et qu'elle soit prise en compte ; c'est donc bel et bien le système qui impose son rythme à l'utilisateur.

Les systèmes *réactifs* réagissent également en permanence aux sollicitations de l'environnement en effectuant des actions sur celui-ci [75]. Nous illustrons ce principe dans la figure 2.2 où les actions sont dites des sorties du système et les sollicitations sont appelées des entrées du système.

A la différence des systèmes interactifs, le rythme est déterminé par l'environnement et non le système. Les systèmes réactifs sont dits temps-réel. Par exemple, le système d'airbag présent dans une voiture est un système réactif : il prend en compte en permanence les différentes données des capteurs présents dans la voiture (vitesse d'impact, par exemple) afin de se déclencher lorsque nécessaire. Le rythme du système imposé par l'environnement doit capter une information de choc assez rapidement afin de déclencher le gonflement de l'airbag.

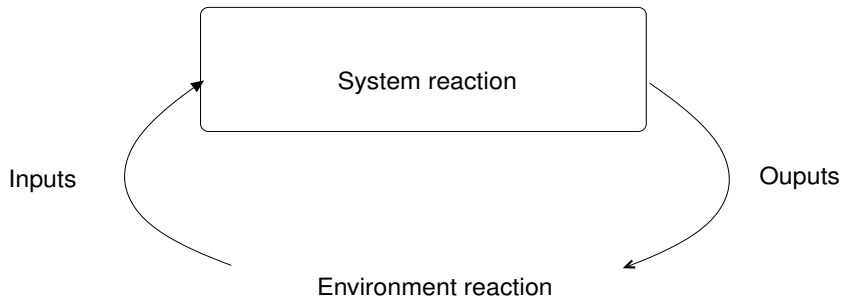


FIGURE 2.2 – Principe général d'un système réactif.

Les systèmes réactifs sont souvent utilisés dans des contextes critiques où un dysfonctionnement peut avoir de très graves conséquences. Dans ce contexte, un critère de qualité est de pouvoir s'assurer que le système est bien déterministe et

donc prévisible. Le déterminisme signifie que la même séquence d'entrées donne toujours la même séquence de sorties.

La famille des langages synchrones [22] a été introduite pour offrir des langages de programmation qui aident à garantir le déterminisme des systèmes réactifs. Nous utilisons des langages de cette famille, basés sur des automates, à la section 5.2.

2.2.1.2 Machines de Mealy

Les machines de Moore et Mealy sont des transducteurs, elles ont des entrées et produisent des sorties. Nous avons utilisé des machines de Mealy dans nos travaux. Les machines de Mealy sont des automates à états finis composées d'un tuple $(S, S_0, \Sigma, \Lambda, T, G)$ où :

- S est l'ensemble fini d'états
- S_0 est l'état initial parmi l'ensemble d'états S
- Σ est l'ensemble fini d'entrées
- Λ est l'ensemble fini de sorties
- T est la fonction de transition $S \times \Sigma \rightarrow S$
- G est la fonction de sortie $S \times \Sigma \rightarrow \Lambda$

Une machine de Mealy peut être utilisée pour représenter un système réactif. Dans la syntaxe graphique d'automates, on représente les états par des ronds et les transitions par des flèches. Chaque transition porte une entrée et une sortie. Dans la définition de base, Σ et Λ sont des ensembles de noms. Dans les systèmes réactifs on autorise les transitions à porter des étiquettes de la forme *cond/act‘* où *cond* est une condition booléenne sur Σ et *act* un ensemble de noms de Λ. A titre informatif, il s'agit d'une différence par rapport aux machines de Moore, les sorties étant portées par des états et non des transitions.

A titre d'exemple, nous illustrons une machine de Mealy en figure 2.3 qui représente le double clic de souris. La machine de Mealy possède deux entrées notées c pour un clic de souris et t pour une unité de temps écoulée pouvant symboliser n'importe quelle métrique de temps (secondes, millisecondes, etc.). La sortie notée d correspond à la détection d'un double clic, la sortie s correspond à la détection d'un simple clic.

Un clic de souris (entrée c) conduit l'automate à passer de l'état W à X. Lorsqu'une unité de temps s'écoule sans qu'il y ait de clic (entrée t “vraie” et l'entrée c “fausse”), l'automate passe de l'état X à Y. Si un second clic se produit avant l'écoulement des trois unités de temps, alors la sortie d sera déclenchée, sinon la sortie s sera effectuée. On note que dans l'état Z, l'entrée t est le troisième top depuis le clic qui a permis de quitter l'état W. Il se peut que cette entrée arrive en même temps que le clic. On règle la priorité en écrivant t.-c (top sans clic) et c/d (clic avec ou sans top) sur deux transitions qui retournent à l'état W.

L'exemple donné ici est simple, cependant lorsque le système réactif est complexe, la machine de Mealy devient également complexe et peut comporter des

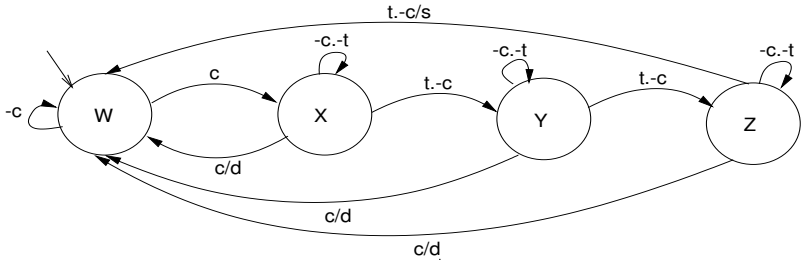


FIGURE 2.3 – Machine de Mealy représentant un double clic de souris (sortie d), si ceux-ci sont séparés par plus de 3 tops (entrée t), alors il s'agit d'un simple clic (sortie s). L'entrée c correspond à un clic. On note "." le ET et "-" le NOT.

milliers d'états. La composition de machines de Mealy permet de réduire cette complexité et de modulariser la représentation d'un système réactif. La modularité est l'un des objectifs de tous les langages synchrones dont celui que nous allons voir en section suivante.

2.2.1.3 Langages synchrones

Les langages synchrones sont des langages de haut niveau qui permettent d'étendre les concepts vus dans les machines de Mealy. Ils sont notamment utilisés dans le domaine où des systèmes de contrôle temps-réel sont nécessaires (avionique, ferroviaire, etc.). Les premiers langages synchrones qui ont eu du succès dans l'industrie sont les langages Esterel [25], Lustre [74] et Signal [95]. Dans cette thèse, nous avons choisi d'utiliser le langage Argos [104] pour la simplicité de ses automates dont nous ré-utilisons certains des éléments de syntaxe.

Le langage synchrone Argos apporte une syntaxe étendue aux machines de Mealy et permet, entre autres, de définir au sein de ceux-ci des variables possédant des valeurs numériques. Les variables sont une façon simplifiée de représenter un ensemble d'états au sein d'un automate. Il est possible de définir des conditions appliquées à ces variables, au sein d'une transition, afin de passer d'un état à un autre. De même, il est possible d'assigner une valeur à ces variables en tant qu'effet d'une transition. En reprenant l'exemple précédent du double clic, nous représentons celui-ci en figure 2.4 avec le formalisme Argos et sa syntaxe étendue comprenant des variables.

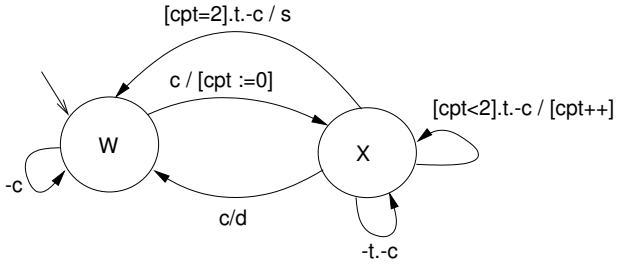


FIGURE 2.4 – Double clic représenté dans le langage Argos à l'aide d'une variable.

Au lieu des états de comptage X, Y, Z de la figure 2.3, une variable notée cpt est définie au sein de l'automate. Les conditions et affectations de cpt sont notées entre crochets. Les sorties d et s sont toujours présentes. La variable cpt est utilisée ici pour compter le nombre d'unités de temps écoulées et permet d'éviter la présence des états Y et Z.

Le langage Argos apporte la possibilité de définir des états temporisés qui sont également une manière simplifiée de représenter un ensemble d'états. De la même façon qu'un timer, un état est associé à une durée temporelle à laquelle il est soumis : soit il reçoit une entrée le faisant sortir de son état avant expiration du timer, soit le timer associé expire. Nous illustrons en figure 2.5 ce principe en prenant le même exemple du double clic et en utilisant un état temporisé.

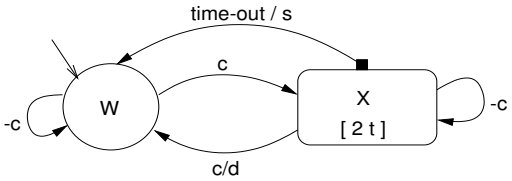


FIGURE 2.5 – Double clic représenté dans le langage Argos à l'aide d'un état temporisé.

Un état temporisé possède une forme rectangulaire qui le permet d'être identifiable, ici l'état X. Le timer qui lui est associé est entre "[" "]" et est de la forme suivante "nombre-d-occurrences nom-entrée" ce qui définit la durée de vie du timer en question. Ainsi, l'état X comptera de manière interne le nombre d'occurrences de t, soit deux, lorsqu'il est l'état courant et qu'il reçoit l'entrée t. Si ce timer expire, c'est-à-dire que l'entrée t est reçue une troisième fois, alors une transition spéciale de timeout est déclenchée. Cette transition spéciale est matérialisée dans le formalisme Argos par un carré noir et porte le label time-out qui définit la transition prise lorsque le timer d'un état a expiré. Ici, il s'agira de déclencher la sortie s correspondant à un simple clic.

Enfin, le langage Argos permet de définir de manière modulaire le produit de machines de Mealy par ce qui est appelée une mise en parallèle [105]. Nous en illustrons un exemple dans la figure 2.6 où deux automates sont présents : le premier comportant deux états W et X, le second comportant l'état Q.

La mise en parallèle est dénotée par la ligne en pointillé. Les deux automates sont exécutés de manière concurrente et peuvent partager des signaux qui sont une sortie de l'un et une entrée de l'autre. Ici, le premier automate sert à effectuer un clic ou un double clic alors que le second automate sert à compter des unités de temps. Une fois que le second automate a compté trois unités de temps, il émet une sortie m afin de le signaler au premier qui décide alors que c'est un simple clic.

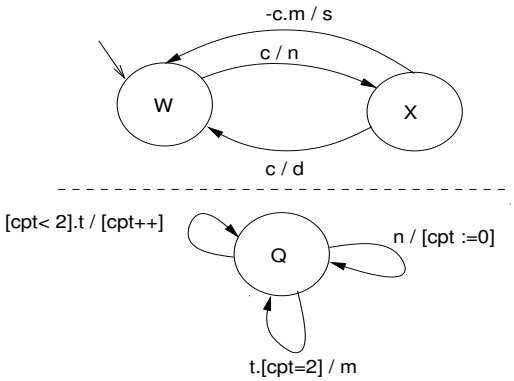


FIGURE 2.6 – Double clic représenté dans le langage Argos à l'aide d'une mise en parallèle.

2.2.2 Modèles asynchrones et outil SPIN

SPIN [79] est l'outil de vérification que nous avons utilisé pour la validation de l'algorithme de gestion de conflits que nous proposons (cf. chapitre 6). Il s'agit d'un outil basé sur la famille des modèles asynchrones que nous présentons dans la section suivante.

2.2.2.1 Système asynchrone

Dans un système synchrone, les différentes parties partagent le temps, ce qui conduit à ce que le produit parallèle des automates amène à effectuer deux transitions simultanément.

Dans un système asynchrone, au contraire, les différentes parties ne partagent pas de notion de temps commune. C'est le cas du système de machines sur internet, d'un réseau de capteurs, etc. Pour réfléchir au comportement des systèmes asynchrones, on utilise des automates et des produits d'automates qui représentent la mise en parallèle asynchrone par de l'entrelacement de transition.

De très nombreux outils sont basés sur l'idée d'entrelacement, tel que l'outil de vérification SPIN.

2.2.2.2 Automates simples pour systèmes asynchrones

Les parties d'un système asynchrone sont représentées par des automates comme ceux de la figure 2.7.

Le produit asynchrone est donné par la figure 2.8.

On voit qu'une transition du produit fait intervenir un seul automate. Par exemple, pour la transition de l'état AB à BC comportant l'action a, seul le premier automate réagit alors que le second ne bouge pas. C'est le principe d'entrelacement des transitions de l'un avec les transitions de l'autre.

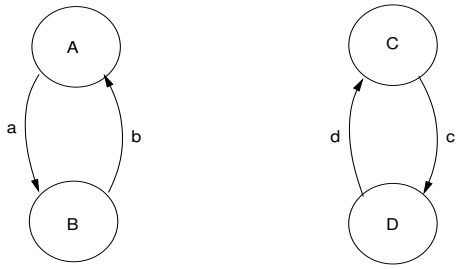


FIGURE 2.7 – Exemple d'automates évoluant au sein d'un système asynchrone. Les noms a, b, c, d représentent des actions abstraites.

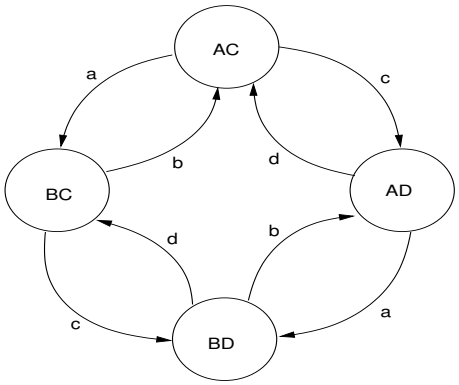


FIGURE 2.8 – Exemple du produit asynchrone des automates de la figure 2.7.

2.2.2.3 Langage Promela

Promela [80] (Process Meta Language) est un langage haut niveau utilisé dans le domaine de la vérification au sein de l'outil SPIN (Simple Promela Interpreter). Il permet de décrire des processus et leurs interactions au travers de l'utilisation de trois éléments principaux du langage : les canaux, les processus et les variables. Un canal permet d'assurer le passage de messages entre deux processus, il s'agit d'une file FIFO qui stocke des messages entre deux processus ou de manière interne, au sein d'un processus même. Ainsi, un canal sert à représenter un transfert de données, par exemple un réseau séparant un émetteur d'un récepteur.

De la même manière que dans tout langage de programmation, des variables peuvent être définies (numériques, booléennes, etc.). Celles-ci représentent des états internes à un processus ou des états globaux. Les variables peuvent également être utilisées pour représenter un message transitant par un canal.

Enfin, les processus utilisent les variables et les canaux. La description d'un processus correspond à un automate représentant son comportement. Les processus s'exécutent de manière indépendante les uns des autres. Un modèle SPIN/Promela est basé sur la notion d'entrelacement.

Nous illustrons dans la figure 2.9 un programme Promela composé de deux processus : un receveur (ligne 4) et un émetteur (ligne 15). La communication entre l'émetteur et le receveur a lieu au travers d'un canal qui peut stocker conq messages

valant chacun un octet (ligne 1). La syntaxe " !" indique l'envoi d'un message dans un canal. Ici, l'émetteur envoie de manière atomique des messages dans le canal contenant la valeur 1 ou 2 (ligne 18 et 19). A l'inverse, la syntaxe " ?" indique la réception d'un message contenu dans un canal, consommé si des "[" "]" sont présents. Ici, peu importe le message reçu par l'émetteur (ligne 8 à 11), celui-ci le consomme et change son état interne (state).

```
 1 chan toR = [5] of {byte};
 2
 3 active proctype Receiver () {
 4   byte state = 0 ;
 5   byte msg_rcv;
 6
 7   do
 8     :: atomic {(state == 0) && (toR?[1]) -> toR?msg_rcv; state=1;}
 9     :: atomic {(state == 0) && (toR?[2]) -> toR?msg_rcv; state=1;}
10     :: atomic {(state == 1) && (toR?[1]) -> toR?msg_rcv; state=0;}
11     :: atomic {(state == 1) && (toR?[2]) -> toR?msg_rcv; state=0;}
12   od
13  }
14
15  active proctype Emitter () {
16   byte msg_sent = 1;
17   do
18     :: atomic {toR!msg_sent; msg_sent=2;}
19     :: atomic {toR!msg_sent; msg_sent=1;}
20   od
21 }
```

FIGURE 2.9 – Exemple d'un programme Promela comprenant deux processus où un émetteur envoie des messages à un récepteur qui modifie son état interne en fonction du nombre de messages reçus.

Nous illustrons l'automate du processus émetteur et récepteur dans la figure 2.10 pour décrire l'échange entre ces deux processus. A gauche de la figure, est représenté l'émetteur envoyant la valeur de msg_sent et pouvant la positionner à 1 ou 2 (ligne 18 ou 19). A droite de la figure est décrit le récepteur qui reçoit n'importe qu'elle valeur et change d'état en modifiant la valeur de state à 1 (ligne 8 si la valeur reçue est 1 ou ligne 9 si c'est la valeur 2) ou 0 (ligne 10 pour la valeur reçue 1 ou ligne 11 si la valeur reçue est 2).

L'outil SPIN permet de simuler toutes les combinaisons d'états possibles des automates. Dans cet exemple, cela consisterait à simuler l'envoi d'un message contenant la valeur 1 ainsi que la réception de celui-ci lorsque le récepteur a pour état interne 0 ou 1. Idem pour l'envoi et la réception du message ayant la valeur 2 avec les

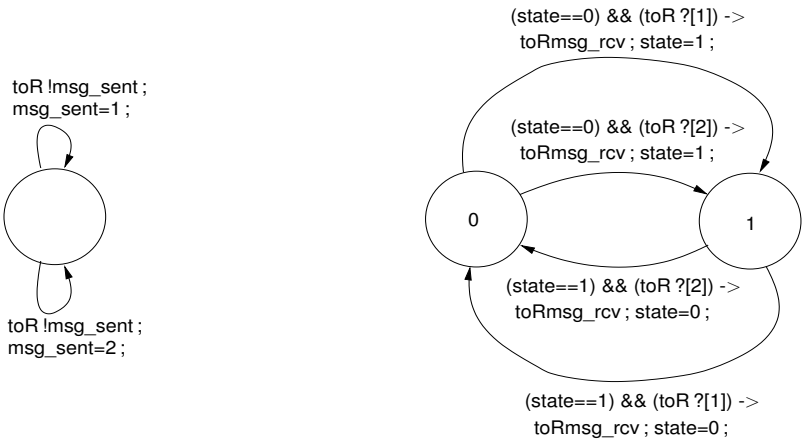


FIGURE 2.10 – Représentation des processus du programme Promela de la figure 2.9 par des automates. L'automate de gauche correspond au processus de l'émetteur. L'automate de droite correspond au processus du récepteur.

mêmes états internes du récepteur. L'outil graphique iSPIN est l'interface graphique de SPIN qui permet d'effectuer la simulation. Il s'agit de l'outil utilisé dans cette thèse pour valider notre algorithme de gestion de conflits (cf. section 6.4.3).

La figure 2.11 illustre une capture d'écran de iSPIN où le programme Promela précédent a été chargé. Ici, la simulation est exécutée en mode interactif (cf. partie 1 de la figure, case "interactive" cochée). Cela permet de choisir le comportement à effectuer par les processus : envoyer une valeur ou la recevoir. Le pop-up de la capture d'écran portant le numéro 2 illustre ce choix à faire : envoyer une valeur et positionner l'envoi de la prochaine valeur à "1" ou "2" (lignes 16 et 17 de la figure 2.9) ou recevoir un précédent message envoyé (ligne 9 de la figure 2.9).

ISpin représente graphiquement les séquences d'envoi et de réception des messages des différents processus (numéro 3 dans la figure). Ici les deux processus sont représentés où l'on voit que l'émetteur a d'abord envoyé un message de valeur 1, puis le récepteur l'a réceptionné, la flèche indiquant à quel moment le message émis a été consommé. Ensuite, l'émetteur a envoyé un message de valeur 1 et un second de valeur 2, le premier message ayant été consommé par le récepteur.

L'outil affiche également l'exécution de chacune des instructions des processus (numéro 4) ce qui permet une analyse exhaustive du fonctionnement de chacun des processus pour trouver de potentiel bug. Un résumé des variables utilisées par chaque processus ainsi que leurs valeurs est également présent (numéro 5). A noter que le chiffre qui suit le nom du processus est un identifiant unique qui permet de différencier les processus les uns des autres lorsque ceux-ci ont le même nom.

Un cas typique de bug qui pourrait être détecté serait lorsque le processus émetteur ne consomme et ne réceptionne pas un message de valeur 2 alors qu'il est dans l'état 1 (ligne 11 du programme n'existant pas). Dans le volet de droite (numéro 3), un message de valeur 2 du côté de l'émetteur n'aurait jamais de flèche vers le récepteur ce qui nous conduirait à conclure que ce message n'est jamais réceptionné. Par

conséquent, le canal deviendrait saturé et l'on pourrait remarquer dans les autres volets (numéros 4 et 5) que le récepteur ne consomme jamais de message ayant la valeur 2 lorsqu'il est dans l'état 1.

Il est à noter que iSPIN permet d'effectuer des simulations en laissant l'outil effectuer lui-même des choix aléatoires contrairement au mode interactif que nous présentons ici. Cette option est présente dans la partie numérotée 1 de la figure. La différence par rapport au mode interactif est qu'au lieu de simuler soi-même des chemins dans les automates (processus), des chemins aléatoires sont générés. Dans ce cas, on cherche alors à vérifier que l'on est bien passer par tel ou tel chemin au travers de l'écriture de contraintes de Logique Temporelle Linéaire (LTL).

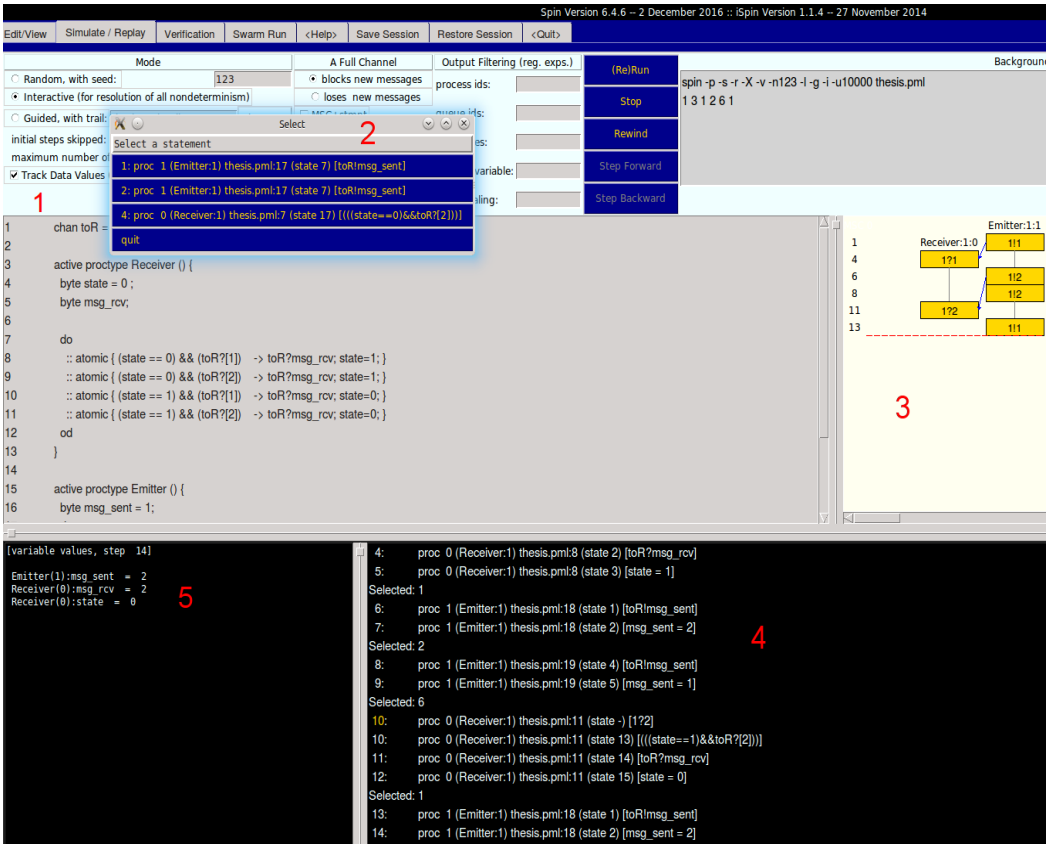


FIGURE 2.11 – Capture d'écran de l'outil graphique iSPIN contenant le programme Promela présenté en figure 2.9.

2.3 Modèles sémantiques

2.3.1 Enjeux liés à notre travail

Afin de pouvoir partager des informations entre les différents acteurs de la ville (données des capteurs, objets physiques, etc.), une nomenclature doit être adoptée par tous pour que les acteurs se comprennent.

Prenons l'exemple de capteurs de température qui sont déployés par deux acteurs différents dans la ville. Tout deux ont leurs propres applications qui utilisent leurs mesures des capteurs. Pour décrire les mesures de ses capteurs, le premier acteur utilise un attribut "temp = xx c" (xx = un nombre quelconque), au sein de l'API utilisée par ses applications, alors que le second a défini un attribut "heat = xx Celsius".

Lorsque ces deux applications veulent partager les données de leurs capteurs et ainsi permettre à leurs applications d'utiliser les données de l'autre, cette description unilatérale des mesures de capteurs est handicapante. les applications devront être ré-écrites afin qu'elles tiennent compte de la nomenclature adoptée par l'autre. Ces acteurs doivent ainsi se mettre d'accord sur une nomenclature commune pour décrire les mesures de leurs capteurs : c'est le rôle principal des modèles sémantiques, promouvoir l'interopérabilité.

Dans le contexte d'une plateforme partagée, cette nomenclature permet ainsi aux acteurs présents de pouvoir aisément utiliser les capteurs et actionneurs des autres. De plus, cela permet à un nouvel acteur de la ville de pouvoir s'intégrer rapidement à la plateforme puisqu'il n'a qu'à utiliser cette nomenclature pour pouvoir utiliser les différents capteurs et actionneurs présents.

Les ontologies permettent de décrire cette nomenclature comme nous allons le voir dans la section suivante.

2.3.2 Ontologies

2.3.2.1 Définition

Les ontologies permettent de définir des modèles sémantiques qui, dans le contexte de l'informatique, représentent les connaissances d'un domaine. Une ontologie est définie [68] comme un ensemble de termes décrivant la signification des concepts d'un domaine particulier ainsi que leurs relations. L'ensemble de ces concepts décrits est appelé un *vocabulaire*.

Il existe des ontologies dans différents domaines, par exemple en biologie pour décrire des génomes [17] ou dans le domaine de la géographie pour décrire des espaces géographiques [55]. Les ontologies permettent de décrire à différents niveaux de granularité les connaissances d'un domaine. Par exemple, il peut exister plusieurs ontologies visant à décrire des espaces physiques où chacune d'entre elles définit à un certain niveau de détails ce qu'est un espace physique (une pièce, une ville, une maison, etc.).

Un domaine de recherche appelé l'alignement d'ontologies s'intéresse à cette problématique liée à la multiplication des ontologies décrivant le même domaine. L'alignement d'ontologies vise à lier des ontologies entre elles en fonction de leur point commun [154].

Des ontologies existent également dans le domaine de l'internet des objets, par exemple, pour décrire les devices, et à un niveau de granularité plus fin les capteurs et actionneurs ainsi que les mesures qu'ils effectuent. Nous verrons que certaines

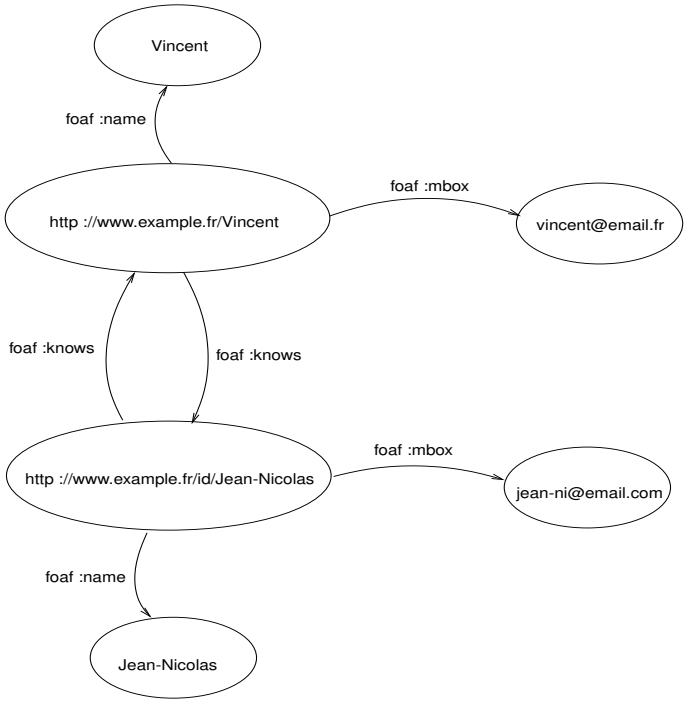


FIGURE 2.12 – Exemple d'une ontologie appliquée à deux personnes qui se connaissent décrivant également leur adresse mail et leur nom.

plateformes que nous avons étudiées (cf. section 3.2) s'appuient sur des ontologies existantes dans l'internet des objets pour ajouter des méta-données au modèle qu'elles proposent.

2.3.2.2 Conception et utilisation des ontologies

Les ontologies sont exprimées sous la forme de triplets $<$ *sujet, prédicat, objet* $>$. Nous décrivons un exemple d'utilisation d'une ontologie en figure 2.12. Il s'agit de l'application d'une ontologie pour représenter les caractéristiques (email et nom) de deux personnes qui se connaissent. Les nœuds qui ont une flèche sortante sont des sujets, les flèches sont des prédicats et les objets sont les nœuds qui possèdent une flèche entrante (un nœud peut être sujet et objet).

L'ontologie utilisée est FOAF [54] qui comporte comme vocabulaire "knows", "mbox", "name" et bien d'autres noms que nous n'utilisons pas ici. L'application de cette ontologie permet de décrire des liens existants entre des personnes. Ici, les deux personnes ont un URI qui permet de les identifier et qui sert de base pour décrire leurs caractéristiques.

Tout comme il existe des URIs et URLs pour identifier et localiser des documents sur le web, il existe également des URIs pour identifier des concepts présents au sein des ontologies (ici, "example.com/..."). Il s'agit d'un des fondements de ce qui est appelé le web sémantique [24] où les notions présentes dans le web sont transposées au domaine des ontologies.

La conception et l'utilisation d'ontologies sont des domaines de recherche à part entière. Il existe des langages plus ou moins expressifs (RDF [103], OWL [21], etc.) pour décrire des ontologies. Divers organismes de standardisation (IEEE, W3C, etc.) tendent à proposer des ontologies (SUMO [119], SSN [40], etc.). au travers de groupes de travail qu'ils ont créés. De même, il existe une multitude de formats de sérialisation pour utiliser des ontologies (XML [31], Turtle [36], N3 [23]) ainsi que des outils permettant de les stocker (triple-store) et de les exploiter (moteur d'inférence).

Nous verrons que les choix que nous avons effectués dans notre travail permettent l'utilisation d'ontologies pour décrire la sémantique de nos modèles (cf. section 5.5.3).

Chapitre 3

État de l'art

Comme nous l'avons décrit dans le premier chapitre, il existe de multiples définitions de la ville intelligente.

Selon nous, une ville n'est pas considérée intelligente lorsque les différents acteurs de la ville ont le monopole exclusif de leurs capteurs et actionneurs, qui se traduit par ce que nous appelons des solutions verticales dans la ville. Après avoir choisi un échantillon des domaines d'applications existants dans la ville, nous décrivons en section 3.1 des solutions verticales proposées dans la ville.

De notre point de vue, une ville est intelligente lorsque les différents acteurs de la ville partagent les données de leurs capteurs l'accès à leurs actionneurs. Ce partage peut s'effectuer au travers d'une plateforme. Ainsi, nous décrivons ce qu'est une plateforme ce qu'elle apporte (cf. section 3.2). Puis, en utilisant des points de comparaison que nous définissons, nous décrivons un échantillon de plateformes qui sont opérationnelles ainsi que celles qui, à un stade moins avancé, ont été proposées mais ne sont pas encore déployées.

A l'aide ces divers points de comparaison, nous dressons en section 3.3 un résumé de ce qui manque aux plateformes actuelles. Ainsi, cela nous permet d'introduire le travail réalisé dans cette thèse.

Sommaire

3.1 Solutions verticales proposées dans la ville

Les solutions présentes dans la ville "pré-intelligente" sont destinées à des domaines d'applications spécifiques (éclairage public, trafic routier, etc.). Des solutions ont été proposées pour traiter spécifiquement les problématiques de ces domaines, nous en décrivons un sous-ensemble en section 3.1. En tenant compte d'un déploiement préalable de capteurs et d'actionneurs, ces solutions permettent l'émergence d'applications dans la ville qui sont verticalement intégrées.

Ainsi, nous montrons que les applications reposant sur ces solutions pourraient être enrichies si elles utilisaient une plateforme horizontale telle que nous le visons dans notre travail (cf. section 3.2). Ce bénéfice est une conséquence du partage, par le biais de la plateforme horizontale, des données de capteurs et des accès aux actionneurs entre les différents acteurs de la ville et leurs domaines d'application respectifs.

3.1.1 Domaines d'application liés à ces solutions verticales

Smart Road Des solutions pour traiter les problématiques liées au trafic routier ("Smart Road") dans la ville ont été proposées [163, 20, 90]. Elles visent à informer les automobilistes des conditions actuelles de circulation sur les axes routiers (densité, météo, etc.), fluidifier le trafic et permettre une intervention rapide des services d'urgence lorsqu'un accident est détecté.

Ces solutions reposent sur l'utilisation exclusive de capteurs déployés dans l'environnement pour leurs propres besoins. Il s'agit de capteurs déployés au sein des infrastructures routières (feux tricolores, routes, etc.) qui permettent d'obtenir des données relatives à la densité du trafic routier et aux conditions météorologiques. De plus, des capteurs (caméra, GPS, etc.) sont également déployés dans les voitures afin de connaître leur localisation et leur permettre de détecter les accidents. Enfin, les feux tricolores sont utilisés en tant qu'actionneurs.

Smart Parking Les différentes propositions autour du domaine du "Smart Parking" visent à guider les automobilistes, de manière optimale, afin qu'ils trouvent une place de parking disponible [62, 169, 87]. Il s'agit de systèmes de réservation de places de parking qui tiennent compte de la localisation des automobilistes, en utilisant un capteur GPS au sein des véhicules, ainsi que des disponibilités des places de parking, collectées grâce à des capteurs présents sous celles-ci. Une fois les données des capteurs collectées, un algorithme d'allocation des places de parking permet à des automobilistes de pouvoir se garer rapidement [62] une fois qu'ils se sont acquittés du paiement de ce stationnement via un système qui le facilite tel que décrit en [84].

Smart lighting Des propositions relatives au "Smart Lighting" ont été formulées pour contrôler les lampadaires de la ville et optimiser l'éclairage public. Parmi elles, certaines s'intéressent au contrôle des lampadaires à destination des piétons circulant

dans des quartiers isolés de la ville [115]. Les lampadaires à proximité du piéton, identifiés via le capteur GPS intégré au smartphone de celui-ci, sont contrôlés et leur intensité d'éclairage est réglée en fonction des préférences du piéton enregistrées dans une application smartphone dédiée et en cours d'exécution.

D'autres propositions visent à éclairer les routes de la ville de façon moins énergivore qu'actuellement [108, 137, 113], c'est-à-dire en ne les éclairant qu'au passage des véhicules qui sont détectés par des capteurs de présence intégrés à la route et/ou aux lampadaires. Certaines solutions proposent, en plus, de pouvoir superviser l'état de fonctionnement des lampadaires à distance [137].

Smart space Afin de superviser les différents lieux publics de la ville ("Smart space"), une plateforme a été proposée au sein du projet européen SOFIA [52]. Contrairement aux précédentes solutions décrites, il s'agit d'une plateforme qui n'est pas restreinte à un type d'application mais à un ensemble d'applications (environnement, sécurité, surveillance, etc.) ayant pour point commun la supervision des lieux publics.

Cette plateforme possède une architecture événementielle utilisant les données de capteurs hétérogènes présents dans les lieux publics. Ces données de capteurs sont agrégées et consolidées sous-forme d'événements qui contiennent des méta-données sémantiques. Ces événements peuvent être consommés par des applications externes et/ou mener à la création d'autres événements (exemple : événement lié à une mesure d'un seuil de CO2 amenant à créer un événement lié à une alerte d'incendie).

Contrairement à ce que nous proposons dans notre travail, la plateforme du projet SOFIA se focalise uniquement sur la remontée de données de capteurs. Par conséquent, seules les applications de supervision peuvent utiliser cette plateforme. D'autre part, cette plateforme n'est pas complètement horizontale puisqu'elle ne partage que les données des capteurs présents dans les lieux publics. Nous verrons dans la sous-section suivante les bénéfices qu'ont les applications à utiliser une plateforme horizontale plutôt que des solutions verticales.

3.1.2 Des solutions verticales à une plateforme horizontale

Les applications utilisant des solutions verticales pourraient être plus élaborées si elles utilisaient une plateforme horizontale qui mutualisent les accès aux actionneurs ainsi que les données des capteurs des différents domaines d'application. Nous présentons quelques exemples de ces applications dans les paragraphes suivants.

L'état d'occupation des places de parking corrélé à l'horaire de la journée amènent de potentiels véhicules à circuler. En accédant aux données des capteurs mesurant cet état d'occupation, les applications de "Smart Road" pourraient affiner leurs prévisions des conditions de circulation aux automobilistes. Réciproquement, les applications de "Smart Parking" pourraient réutiliser les données collectées par les capteurs présents sur la route et/ou les feux tricolores afin de prendre en compte les conditions de circulation pour guider les automobilistes.

Puisque les données des capteurs et accès aux actionneurs sont partagés au sein d'une plateforme horizontale, les applications de "Smart Lighting" pourraient, par exemple, réutiliser les capteurs présents sur la route pour détecter la densité de trafic et ajuster l'éclairage public. De même, d'autres applications seraient susceptibles de contrôler les lampadaires de la ville, telle qu'une application gérant les catastrophes naturelles guidant l'évacuation des piétons ou des véhicules via l'éclairage public.

Du moment qu'une plateforme horizontale peut garantir une certaine qualité de service pour satisfaire les exigences des applications de contrôle (cf. section 4.3), il est profitable pour des applications d'utiliser une plateforme horizontale pour superviser comme pour contrôler la ville.

3.2 Plateformes pour l'internet des objets et la ville intelligente

La terme plateforme permet de désigner un ensemble de composants ou de services assurant la médiation des informations entre les différents utilisateurs de la plateforme. Ce terme peut être pris au sens riche en le comparant avec le terme "plateforme multi-faces" qui a été défini par les économistes pour décrire un modèle économique [73].

Les plateformes multi-faces, telles que Airbnb [1] ou Blablacar [2], créent de la valeur par la contribution des différentes catégories d'utilisateurs de la plateforme, appelées faces. Par exemple, la plateforme Blablacar a deux faces, à savoir les conducteurs de véhicules et les passagers qui créent de la valeur à la plateforme. Dans le contexte d'une plateforme pour l'internet des objets, les catégories d'utilisateurs peuvent être, par exemple : les fabricants de capteurs/actionneurs, les fournisseurs de connectivité réseau, les propriétaires des données des capteurs ou encore les développeurs d'applications.

Cette médiation effectuée par la plateforme entre les différentes catégories d'utilisateurs amène à créer une synergie et un véritable éco-système. Par exemple, plus il y a de propriétaires de données de capteurs utilisant une plateforme pour l'internet des objets et plus il y a de développeurs d'applications qui utiliseront cette plateforme et inversement.

Nous avons choisi un échantillon de plateformes proposées et un échantillon des principales plateformes opérationnelles. Des études [19, 111, 6] présentent un panorama de différentes plateformes où sont évalués les domaines d'applications qu'elles adressent, les technologies que celles-ci utilisent ainsi que les problématiques restantes à résoudre selon les auteurs. Le même type d'étude existe pour présenter les diverses architectures logicielles utilisées par ces plateformes [69]. Cependant, nous considérons qu'il manque certains critères dans la description des plateformes de ces études pour positionner le travail effectué dans cette thèse.

1. https://www.airbnb.fr/
2. https://www.blablacar.fr/

Ainsi, nous avons retenu les critères suivants pour décrire les plateformes choisies :

— Le niveau d'abstraction proposé aux applications ainsi que la présence de méta-données. Le niveau d'abstraction peut être : du relais de données brutes de capteurs et/ou de commandes aux actionneurs, du relais de données structurées de capteurs et/ou commandes aux actionneurs ou une consolidation des données et des commandes sous forme d'*entités*. A propos des méta-données, il est possible que les plateformes en fournissent ou non. Lorsqu'elles en fournissent, nous distinguons les méta-données "informelles" et "formelles". Les méta-données "informelles" sont des informations obtenues par les développeurs d'applications après interprétations de celles-ci. Par exemple un développeur saura que les mesures d'un capteur ont pour unité le Celsius s'il connaît le type de capteur, son modèle ou si une documentation le décrit. A l'inverse, les méta-données "formelles" sont décrites au moyen d'ontologies et de langages (RDF, OWL, etc.) et peuvent être interprétées par des machines.

— Le périmètre de couverture de la plateforme, en terme de domaines d'applications adressés et de connectivité réseau utilisée par la plateforme pour communiquer avec les capteurs et actionneurs.

— La prise en compte de la sécurité et de la confidentialité par la plateforme. La sécurité fait référence à l'utilisation de moyens tel que le chiffrement des communications avec les applications et/ou les capteurs/actionneurs ainsi que l'authentification et l'autorisation des applications pour qu'elles accèdent à la plateforme. Nous désignons la confidentialité comme étant relative à la protection des données, pouvant être potentiellement sensibles (exemple : impact sur la vie privée). Il existe des moyens de garantir la confidentialité des données [156], par exemple en contrôlant l'utilisation des données (exemple : anonymisation des données) ou effectuant du contrôle d'accès de manière fine afin d'empêcher certaines actions de la part des applications (exemple : impossible d'accéder à des données de géolocalisation).

— Du partage des actionneurs et de la prise en compte du temps réel. Le premier élément désigne le fait que la plateforme le partage les accès aux actionneurs entre les applications utilisant la plateforme. Le second élément correspond au fait que la plateforme comporte des mécanismes permettant à des applications de contrôle temps-réel d'utiliser la plateforme.

— L'infrastructure utilisée ou préconisée pour le déploiement de la plateforme. Il peut s'agir d'une infrastructure uniquement cloud où les capteurs et actionneurs doivent posséder une connectivité IP pour communiquer avec la plateforme. L'hébergement de la plateforme peut être distribuée en mêlant des passerelles et une infrastructure cloud pour ne pas que cette limitation relative à la connectivité des capteurs et actionneurs existe, les passerelles assurant le passage des données. Enfin, la plateforme peut être déployée dans une infrastructure incluant du "edge computing", où les passerelles ne sont pas utilisées exclusivement pour relayer les données mais également en tant que

puissance de calcul à part entière où des applications peuvent, par exemple, s'interfacer avec ce matériel.

— L'interface exposée aux applications, c'est-à-dire l'API qui va être utilisée par les applications pour interagir avec la plateforme.

— De la possibilité pour les applications de gérer et d'administrer les *devices*, c'est-à-dire le matériel physique composé d'un ou plusieurs capteurs et/ou actionneurs. Ces actions d'administration peuvent être, par exemple, de redémarrer des devices ou de reconfigurer leur firmware. Par ailleurs, le second critère est relatif à la configuration/reconfiguration automatique des modèles présents dans la plateforme.

— L'existence de validation sous-forme de simulations, expérimentations ou prototypages qui visent à démontrer la viabilité de la plateforme.

3.2.1 Plateformes opérationnelles

3.2.1.1 Plateformes de mise en commun des capteurs et actionneurs

Nous présentons quelques plateformes pour l'internet des objets qui sont actuellement commercialisées et opérationnelles. Elles ont principalement pour but de mettre en commun les données des capteurs entre les différents utilisateurs de la plateforme.

Xively Xively [172] est une plateforme pour l'internet des objets, hébergée au sein d'une infrastructure de cloud public. Elle permet aux utilisateurs de partager les données de capteurs ainsi que l'accès aux actionneurs, bien que l'accent soit mis sur le partage des données. En effet, le principal but de cette plateforme est de stocker les données des capteurs et de les mettre en commun afin que les utilisateurs puissent les interroger.

Les utilisateurs enregistrent leurs devices afin de partager ce qui est appelé des "flux de données" qui sont un ensemble de données des capteurs et/ou d'ordres émis aux actionneurs. Par exemple, une station météo NetAtmo peut être enregistrée pour que ses mesures de capteurs (humidité, température, etc.) soient partagées. Un flux de donnée n'est associé qu'à un capteur ou actionneur du device à la fois. Ici, le détenteur de la station météo NetAtmo pourra partager le flux de données relatif au capteur de température de la station par exemple. Des SDK sont disponibles pour diverses plateformes (Arduino [14], ARM Mbed [15], etc.) et différents langages (Java, Python, etc.) afin que les devices puissent envoyer les mesures de leurs capteurs à la plateforme Xively ; les utilisateurs doivent avoir, au préalable, enregistré les devices au sein de la plateforme.

Une API REST et MQTT est exposée aux applications pour leur permettre d'obtenir les caractéristiques des devices (modèle du produit, numéro de série, etc.), leur géolocalisation (latitude, longitude) ainsi que les flux de données des capteurs/actionneurs des devices. Quelques méta-données (timestamp, unité de mesure), ainsi qu'une faible agrégation de celles-ci (moyenne, minimum et maximum

des mesures des capteurs) sont ajoutées. Les devices publient leurs données dans la plateforme via également une API REST et MQTT.

La plateforme Xively permet aux utilisateurs de définir des autorisations afin de restreindre l'accès aux flux de données à certaines applications, via des API keys ; l'authentification des applications est effectuée avec le protocole d'authentification OAuth2 [76]. De plus, la plateforme supporte les communications chiffrées au travers du protocole sécurisé HTTPS.

Enfin, en terme de configuration, la plateforme permet aux applications d'administrer les devices (device provisioning) en fournissant, par exemple, des moyens pour mettre à jour le firmware ou modifier les paramètres de certains devices. Cependant, aucun moyen n'est fourni aux applications pour qu'elles se configurent automatiquement et soient autonomes.

Critère / Plateforme	Xively
Niveau d'abstraction / Présence de métadonnées	Relais de données structurées / Métadonnées informelles
Périmètre de couverture	Tout domaine d'application. Réseaux de capteurs et actionneurs avec connectivité IP.
Sécurité / confidentialité	Oui / Non
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Oui / Non
Infrastructure de déploiement	Uniquement Cloud
Interface exposée aux applications	REST
Device management / Auto-configuration	Uniquement device management
Simulations, expérimentations, prototypage	Plateforme opérationnelle.

Thingspeak Thingspeak [164] est une plateforme pour l'internet des objets, hébergée dans une infrastructure de cloud public, qui est comparable à la plateforme Xively puisqu'elle suit les mêmes objectifs c'est-à-dire le partage des données.

Les utilisateurs de la plateforme enregistrent leurs devices en tant que "channel", un seul "channel" est associé à un device, et complète des informations relatives à la géolocalisation (latitude, longitude) de leurs devices. Lors de l'enregistrement d'un device, les capteurs et/ou actionneurs intégrés à celui-ci (appelés "channel feed") ainsi que les informations relatives à l'unité de mesure de ces capteurs/actionneurs doivent être renseignées (Celsius pour un capteur de température, par exemple). Enfin, les utilisateurs peuvent choisir de partager ou non le channel qu'ils viennent de créer, ayant pour conséquence de partager ou non les "channel feed" relatifs au channel. Lorsqu'un utilisateur choisit de partager un channel, il peut générer des "APIs keys" pour pouvoir lire et/ou écrire le channel et donc les channels feeds.

De même que dans la plateforme Xively, une API REST et MQTT est exposée aux applications pour leur permettre d'interroger les différents "channel" et "channel feed" des utilisateurs de la plateforme. Un channel contient des informations à propos du device auquel il est attaché (nom, identifiant, etc.) ainsi que quelques méta-données (latitude, longitude) qui ne suivent aucun standard particulier. Un channel feed permet d'obtenir les différentes données du capteur/actionneur auquel celui-ci est associé ainsi qu'une faible agrégation des mesures du capteur/actionneur (moyenne, minimum, maxium).

Les requêtes envoyées par les applications sont en lecture et/ou écriture suivant l'API key qui est fournie par ces dernières. L'authentification des applications est effectuée grâce au protocole OAuth2 [76]. Le protocole HTTPS est supporté afin que les communications soient chiffrées vers la plateforme.

Enfin, à l'inverse de la plateforme Xively, cette plateforme ne permet pas d'effectuer de la gestion de device, ni même de l'auto-configuration.

Critère / Plateforme	Thingspeak
Niveau d'abstraction / Présence de métadonnées	Relais bruts de données / Métadonnées informelles
Périmètre de couverture	Tout domaine d'application. Réseaux de capteurs et actionneurs avec connectivité IP.
Sécurité / confidentialité	Oui / Non
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Oui / Non
Infrastructure de déploiement	Uniquement Cloud
Interface exposée aux applications	REST et MQTT
Device management / Auto-configuration	Non / Non
Simulations, expérimentations, prototypage	Plateforme opérationnelle.

3.2.1.2 Plateformes des fabricants d'équipements

Les fabricants d'équipements, c'est-à-dire ceux qui commercialisent des capteurs et des actionneurs, tels que le fabricant Philips et sa lampe Philips Hue [136] ou encore FitBit et son bracelet connecté [53], interviennent encore peu dans la ville intelligente. A notre connaissance, seules les stations météo du fabricant NetAtmo [118] sont destinées à des cas d'usages relatifs à la ville intelligente puisque ces équipements doivent être déployés à l'extérieur. Cependant, nous jugeons nécessaire de tenir compte de ces plateformes puisqu'elles seront potentiellement amenées à se développer.

Les détenteurs d'une station météo doivent enregistrer leur produit sur la plateforme Netatmo afin de pouvoir obtenir les données des capteurs présents dans

l'équipement. Cette plateforme est hébergée dans une infrastructure cloud qui permet aux applications d'interroger les différentes mesures. Le niveau d'abstraction proposé par cette plateforme se base sur des données des capteurs de la station météo enregistrée (CO2, température, humidité, vitesse du vent, etc.) dans la plateforme. Très peu de métadonnées sont associées à ces mesures qui ne sont pas décrites de manière formelle. Elles servent simplement à décrire les unités de mesure des diverses données de capteurs.

Les données des capteurs ne sont pas nécessairement cantonnées à un seul domaine d'application. Elles peuvent être utilisées différemment suivant les applications, par exemple une application de trafic routier peut les utiliser pour avertir les automobilistes en les corrélant avec les conditions de circulation, alors qu'une application gérant l'irrigation des parcs publics utilisera ces données à des fins différentes.

Enfin, l'accès aux données des capteurs peut se faire par le protocole HTTPS. De plus, un mécanisme d'authentification est intégré à la plateforme, qui se base sur le protocole OAuth2 (via des tokens d'accès) et qui permet aux diverses applications d'être authentifiées pour pouvoir accéder aux données des capteurs.

Critère / Plateforme	NetAtmo
Niveau d'abstraction / Présence de métadonnées	Relais de données structurées. Métadonnées informelles
Périmètre de couverture	Domaine d'application variable. Capteurs uniquement propriétaires
Sécurité / confidentialité	Oui / Pas d'information
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Non / Non
Infrastructure de déploiement	Cloud
Interface exposée aux applications	REST
Device management / Auto-configuration	Non / Non
Simulations, expérimentations, prototypage	Plateforme opérationnelle

3.2.1.3 Plateformes des opérateurs de télécommunications

Datavenue La plateforme pour l'internet des objets Datavenue [44] est actuellement commercialisée et appartient à l'opérateur de télécommunication Orange. Elle comporte trois principaux composants : "Live Objects", "Flexible Data" et "Flux Vision". "Flexible Data" est dédié à l'analyse prédictive de données collectée par la plateforme, "Flux Vision" se focalise sur le traitement des données mobiles pour mesurer, par exemple, la fréquentation d'une zone géographique. Dans ce paragraphe, nous nous focaliserons sur "Live Objects" qui est l'élément principal de la plateforme Datavenue car il permet d'interconnecter les applications et les capteurs afin que ces derniers soient partagés.

Cette plateforme vise à permettre aux applications d'accéder aux données des capteurs en les mutualisant au sein d'une infrastructure cloud. Les devices pourvus d'une connectivité IP publient leurs données au sein de la plateforme via le protocole HTTP et/ou MQTT. De même les devices LPWA, par exemple ceux pourvus de la technologie LoRA, publient également leurs données mais en utilisant des serveurs appelés "network server" qui sont les intermédiaires entre la plateforme et les devices LPWA.

En terme de niveau d'abstraction, une structure des données de capteurs sous la forme de "stream" est proposée. Un stream peut être obtenu au travers d'une API REST et/ou MQTT. Il s'agit d'un ensemble de clé-valeur qui représente les mesures des capteurs associés à un device, par exemple un device possèdant un capteur de température et d'humidité sera associé à un seul stream. Quelques méta-données sont incluses dans un stream en plus des données de capteurs présents dans un stream (timestamp, modèle du device, longitude, latitude). Ces méta-données sont identifiés comme telles via le nom de la clé ("longitude", "timestamp", etc.). Elles ne suivent aucun formalisme particulier.

Par ailleurs, puisque DataVenue utilise le moteur de recherche et d'indexation ElasticSearch [3] disposant lui même d'une API REST, il est donné la possibilité aux applications d'agréger les données d'un stream particulier. Ainsi, diverses opérations telles que la moyenne, la médiane, la somme, le minimum ou encore le maximum peuvent être appliquées à une ou plusieurs mesures contenues dans un stream.

DataVenue intègre des fonctionnalités de device management, pour permettre de savoir si un device est connecté ou non à la plateforme. De même, il est possible de redémarrer un device en particulier ou encore de re-configurer le firmware de celui-ci. Peu d'informations sont néanmoins données à propos de ces diverses fonctionnalités.

Le protocole sécurisé HTTPS est supporté entre la plateforme et les applications/devices. Un système de permission sur la base d'API Key est présent et permet de définir, pour un stream particulier, les APIs Keys autorisées à accéder à celui-ci. Ainsi, chaque requête envoyée par une application vers un stream particulier doit obligatoirement comporter une API Key afin que l'application soit authentifiée. Une API Key peut avoir une période de validité limitée. Cependant ce mécanisme d'authentification reste assez basique puisqu'il n'intègre pas un système de permission permettant, par exemple, de restreindre l'accès à certaines données de capteurs d'un stream.

3. https://www.elastic.co/fr/products/elasticsearch

Critère / Plateforme	Datavenue
Niveau d'abstraction / Présence de métadonnées	Relais de données structurées / Quelques méta-données informelles
Périmètre de couverture	Tous domaines d'applications. Capteurs connectivité IP et LPWA
Sécurité / confidentialité.	Oui / Pas d'information
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Non / Non
Infrastructure de déploiement	Cloud
Interface exposée aux applications	REST et MQTT
Device management / Auto-configuration.	Oui / Pas d'information
Simulations, expérimentations, prototypage	Plateforme opérationnelle

Sigfox L'opérateur de télécommunication Sigfox [4] possède sa propre plateforme qui est hébergée dans une infrastructure de cloud computing et qui permet de gérer uniquement les devices embarquant sa propre technologie réseau LPWA (Low Power Wide Area). Cette plateforme a déjà été déployée dans plusieurs pays européens, comme à Moscou [171] ou à Barcelone [170] afin, par exemple, de superviser le nombre de places de parking disponibles/occupées dans la ville.

Les données de capteurs collectées ne sont pas restreintes à un domaine d'application, il peut s'agir de données liées à la qualité de l'air, au niveau de remplissage d'une poubelle, à la disponibilité d'une place de parking, etc. Cependant, la plateforme Sigfox est uniquement compatible avec les devices comprenant la technologie réseau LPWA de Sigfox même, ce qui limite le nombre de domaines d'applications couvert par la plateforme.

La figure 3.1 illustre l'architecture de déploiement de cette plateforme. La plateforme Sigfox est principalement destinée à la collecte des données de capteurs. Les capteurs envoie de manière sécurisée des données vers des passerelles, appelées "base stations", via un chiffrement effectué par une clé privée contenue dans chaque device embarquant la technologie Sigfox. Les "base stations" transfèrent ces données via un tunnel VPN à la plateforme Sigfox.

Enfin, les applications communiquent avec la plateforme Sigfox au travers d'une API REST. Il est également possible de définir dans la plateforme un "callback" afin que les applications reçoivent des données d'un capteur en particulier. Un "callback" est une URL vers laquelle sont envoyées les données d'un capteur reçues par la plateforme qui permet aux applications de ne pas devoir l'interroger.

Les réponses reçues par les applications sont des données de capteurs brutes et plus exactement un tableau d'octets. Quelques méta-données (timestamp, bruit

4. https://www.sigfox.com

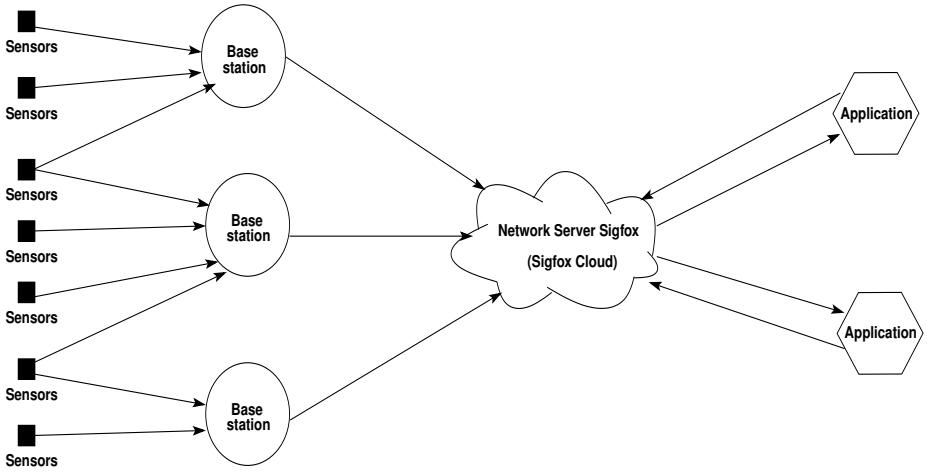


FIGURE 3.1 – Vue d'ensemble de l'architecture de déploiement de la plateforme Sigfox.

du signal, identifiant du device) sont présentes dans les réponses. Elles peuvent être identifiées en tant que telles par les applications grâce à l'attribut désignant la méta-donnée en question.

La plateforme Sigfox permet de configurer l'accès, via des groupes, aux données à un ou plusieurs capteurs. Ces groupes sont constitués d'un couple login/mot de passe pour authentifier des applications afin de les autoriser ou non à obtenir des données de capteurs après qu'elles aient envoyées une requête à l'API REST de la plateforme.

Critère / Plateforme	Sigfox
Niveau d'abstraction / Présence de métadonnées.	Relais de données brutes / Quelques méta-données informelles (timestamp, bruit du signal, identifiant du device)
Périmètre de couverture	Domaine d'application variable. Capteurs et actionneurs uniquement propriétaire.
Sécurité / confidentialité.	Oui / Pas d'information
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Non / Non
Infrastructure de déploiement	Cloud et passerelles
Interface exposée aux applications	REST
Device management / Auto-configuration.	Pas d'information / Pas d'information
Simulations, expérimentations, prototypage	Plateforme opérationnelle

3.2.2 Propositions de plateformes fédératrices partageant les données des capteurs

Des plateformes horizontales se focalisant sur le partage des données de capteurs ont été proposées. Il s'agit d'une grande partie des plateformes horizontales. Nous présentons un échantillon de ces plateformes en décrivant leurs objectifs et en les comparant au travers des critères que nous avons définis précédemment (cf. section 3.2).

Plateforme "CityHub" La plateforme CityHub [96] a été proposée pour concentrer dans une infrastructure cloud les données de capteurs issues des sous-systèmes de la ville intelligente (transport public, trafic routier, portail de données ouvertes, etc.) et les partager.

Le premier objectif de cette plateforme est de fournir une infrastructure cloud hybride, mêlant à la fois cloud privé et public, afin de prendre en considération les infrastructures privées existantes et de les intégrer. La présence de passerelles est également mentionnée afin de servir d'intermédiaire pour relayer les données vers l'infrastructure cloud. L'infrastructure proposée vise à être utilisée par des applications de supervision mais, puisque ce n'est pas mentionné, n'adresse pas les applications de contrôle.

Le second objectif de la plateforme CityHub est de faciliter le développement d'applications au travers d'une API REST. Cette API se base sur la spécification HyperCat [41] : un catalogue de ressources REST annoté par des méta-données sémantiques est exposé aux applications. Ces ressources REST sont liées à des capteurs où les annotations sémantiques, basées sur des instructions RDF, décrivent les informations de celles-ci (unité de mesure, précision, etc.). Outre l'interopérabilité de la plateforme, les applications peuvent chercher les ressources REST qu'elles souhaitent utiliser et obtenir des informations à propos de celles-ci.

La sécurité est mentionné comme étant prévue d'être traitée pour la prochaine version de la plateforme. Entre autre, cela comprend l'ajout d'un système d'autorisation, sur la base d'API Keys, permettant de contrôler l'accès aux ressources REST des applications et de faire en sorte que les applications ne puissent voir, dans le catalogue de ressources, que les ressources REST auxquelles elles ont accès.

Il est à noter que la plateforme CityHub a été expérimentée à deux endroits : au Royaume-Uni et au Canada. Au Royaume-Uni, cette plateforme est utilisée pour la supervision et la maintenance des autoroutes, les données collectées étant relatives aux conditions de trafic (accidents, travaux, bouchons, etc.) et aux conditions météorologiques (inondation, pluie, etc.). Au Canada, la plateforme est utilisée dans la ville de Vancouver et utilise les données publiques de la municipalité ainsi que les données de l'opérateur fournissant des vélos en libre service.

Critère / Plateforme	CityHub
Niveau d'abstraction / Présence de métadonnées	Relais de données structurées / Métadonnées formelles
Périmètre de couverture	Tout domaine d'application et tout type de réseaux de capteurs et actionneurs
Sécurité / confidentialité	Oui (prévu) / Non
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Non / Non
Infrastructure de déploiement	Cloud et passerelles
Interface exposée aux applications	REST
Device management / Auto-configuration	Non / Non
Simulations, expérimentations, prototypage	Expérimentations dans deux villes (échelle non-communiquée)

Projet européen "VITAL" La plateforme IoT du projet européen VITAL [167] vise à fédérer les silos hétérogènes de la ville intelligente en intégrant les données de leurs capteurs respectifs issues de leurs propres infrastructures logicielles. Comme décrit en [134, 133], cette plateforme propose une approche "Cloud of Things" relative à l'utilisation d'une infrastructure de cloud computing, pour assurer le passage à l'échelle de la plateforme, qui intègre et abstrait les données des capteurs hétérogènes qu'elle récolte.

L'acquisition de ces données de capteurs hétérogènes est rendue possible par le modèle "Sensing as a Service" [131]. Celui-ci définit comment les données des capteurs peuvent être publiées et consommées entre les différents acteurs afin que celles-ci soient partagées et réutilisées, moyennant des contreparties financières. Les données des différents capteurs étant acheminées à la plateforme par le biais d'une infrastructure dédiée à la collecte comprenant des passerelles disséminées dans la ville.

Une fois les données de capteurs collectées, il est proposé de les représenter de manière générique en utilisant l'ontologie Semantic Sensor Network [40] (SSN) qui décrit en détail les capacités des capteurs, leurs propriétés et les mesures qu'ils relèvent. Une multitude de méta-données sont associées aux différentes données des capteurs sur la base de la réutilisation d'un grand nombre, autres que SSN, permettant de décrire en détails le contexte lié à chaque capteur ou à un ensemble de capteurs (objet physique lié, domaine d'application, zone géographique, etc.). Par ailleurs, il est possible d'agréger les données des capteurs via l'utilisation d'un Complex Event Processing, par exemple en calculant la médiane ou la moyenne des valeurs d'un capteur.

Les applications utilisent une interface REST nommée "Virtualized Unified Ac-

cess Interfaces" pour interagir avec la plateforme VITAL. Les applications peuvent utiliser les méta-données présentes dans les modèles afin de sélectionner et filtrer les données des capteurs qu'elles souhaitent obtenir (exemple : filtrer des données de capteurs par le type de ceux-ci ou leur géolocalisation). Ce filtrage s'effectue par l'interrogation d'une ressource REST particulière pouvant être vue comme une ressource catalogue.

En terme de sécurité, la plateforme VITAL supporte les communications chiffrées et dispose d'un mécanisme d'authentification des applications. Cependant peu d'informations sont données à ce propos puisque l'accent n'est pas mis sur ces problématiques. Il est prévu que la plateforme VITAL soit expérimentée dans la ville de Londres et dans la ville d'Istanbul, au travers de cas d'usages liés au domaine du trafic et du transport routier, mais peu d'informations sont données à ce propos.

Critère / Plateforme	VITAL
Niveau d'abstraction / Présence de métadonnées	Relais de données structurées / Métadonnées formelles
Périmètre de couverture	Tout domaine d'application et tout type de réseaux de capteurs et actionneurs
Sécurité / Confidentialité	Oui / Non
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Non / Non
Infrastructure de déploiement	Cloud et passerelles
Interface exposée aux applications	REST
Device management / Auto-configuration	Non / Non
Simulations, expérimentations, prototypage	Expérimentations prévues dans deux villes

IoT4s IoT4s est une proposition d'architecture de plateforme pour la ville intelligente [49, 48] qui a notamment pour but de fournir un modèle homogène des différents capteurs hétérogènes présents dans la ville et des métriques qui sont collectées et stockées par la plateforme. Cette plateforme vise à être hébergée dans une infrastructure comprenant des passerelles appelées "Sensors Manager" et des serveurs hébergés dans des centre de données.

Les auteurs font l'hypothèse que les différents capteurs présents dans la ville intelligente seront pourvus du système d'exploitation ContikiOS [45] ou TinyOS [97], permettant aux capteurs de disposer d'une connectivité IPv6. Les capteurs envoient ainsi leurs données au travers du protocole XMPP, lequel permet l'envoi de données chiffrées.

L'architecture de cette plateforme est composée de quatre couches : (i) une API REST utilisée par les applications ; (ii) un "Sensor Observation Service Agent" (SOS

Agent) qui est une abstraction des capteurs et de leurs mesures ; (iii) un "Sensor Manager" (SM) qui permet de collecter les différents données des capteurs ; (iv) les différents réseaux de capteurs qui interagissent avec le SM. Les communications entre le SOS Agent et le SM ont lieu au travers du protocole XMPP qui permet des communications chiffrées. En revanche, les communications avec les applications ne sont pas mentionnées comme étant chiffrées. De manière générale, la sécurité et la confidentialité ne sont pas considérées dans IoT4s.

L'abstraction proposée repose sur le standard Sensor Web Enablement (SWE) qui a été proposée par l'Open Geospatial Consortium [42]. Ce standard est un modèle qui décrit les capteurs eux-mêmes (altitude, longitude, numéro de série, etc.) et leurs mesures (timestamp, valeur, unité, etc.). Il est à noter que le standard SWE est utilisé par l'ontologie SSN qui propose une ontologie comprenant plus de détails (exemple : précision et fréquence des mesures, observations liées aux mesures, etc.) et qui est plus facilement extensible puisqu'elle intègre plusieurs autres ontologies.

Enfin, un prototype d'une plateforme a été réalisé où quatre capteurs physiques sont utilisés et une trentaine d'autres sont simulés via l'utilisation de Cooja [128] qui est un outil permettant d'intégrer des simulations de capteurs équipés du système d'exploitation ContikiOS.

Critère / Plateforme	IoT4s
Niveau d'abstraction / Présence de métadonnées	Relais de données structurées / Métadonnées formelles
Périmètre de couverture	Tout domaine d'application et uniquement pour les réseaux de capteurs IPv6
Sécurité / confidentialité	Oui, partiellement / Non
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Non / Non
Infrastructure de déploiement	Cloud et passerelles
Interface exposée aux applications	REST
Device management / Auto-configuration	Non / Non
Simulations, expérimentations, prototypage	Prototype avec quelques capteurs physiques et plusieurs simulés

Plateforme "SPITFIRE" La plateforme IoT "SPITFIRE" [135, 26] appartient au domaine du "Semantic Web of Things" (SWoT) et se focalise sur le partage de données provenant de capteurs hétérogènes, peu importe le domaine d'applications auquel ces capteurs sont liés. Le SWoT est défini [150] comme regroupant le domaine de l'IoT et du Semantic Web afin de décrire, en réutilisant des ontologies, les capteurs, leurs données et les éléments physiques (routes, places de parking, etc.) liés aux capteurs.

De la même manière que la plateforme VITAL, qui peut être considérée comme une plateforme SWoT, la plateforme SPITFIRE réutilise une multitude d'ontologies (Dolce Ultralite [58], SSN [40], etc.) afin de décrire les capteurs et leurs mesures en leur associant des méta-données. L'ajout de méta-données aux capteurs et à leurs données collectées se fait de manière semi-automatique par la plateforme. En effet, une reconnaissance de motif est effectuée par la plateforme, en fonction de la corrélation de données de capteurs, pour générer ces annotations. Par exemple, si deux capteurs émettent les mêmes mesures dans un court intervalle de temps, alors il pourra être déduit que ces capteurs sont potentiellement au même endroit et des méta-données relatives à leur géolocalisation seront ajoutées à ces capteurs.

Le niveau d'abstraction proposé par la plateforme ne se focalise pas seulement sur les données des capteurs. Une abstraction de plus haut niveau sous la forme d'entités appelées "Semantic Entities" est proposée. Des états sont obtenus (places de parking vides ou occupées) par la consolidation et/ou l'agrégation de données de capteurs. Les cibles de cette abstraction de plus haut niveau (places de parking, route, etc.) sont également décrites par des ontologies.

La plateforme SPITFIRE expose une API REST afin que les applications puissent obtenir des informations qui sont contenues dans le modèle à deux niveaux (capteurs et leurs données, éléments physiques et leurs états). Un point d'entrée SPARQL permet aux applications la découverte et le filtrage des données de capteurs et des états des éléments physiques en fonction des critères relatifs aux méta-données décrites par des ontologies (unité de mesure, localisation, etc.)

Enfin, la faisabilité de la plateforme a été démontrée par la conception d'un prototype basé sur un cas d'usage relevant du domaine du "Smart Parking". Le prototype réalisé de la plateforme utilise les données émises par une quarantaine de capteurs qui sont pourvues d'une connectivité IPv6.

Critère / Plateforme	SPITFIRE
Niveau d'abstraction / Présence de métadonnées	Relais de données structurées et consolidation sous la forme d'entités / Métadonnées formelles
Périmètre de couverture	Tout domaine d'application et tout type de réseaux de capteurs
Sécurité / confidentialité	Non / Non
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Non / Non
Infrastructure de déploiement	Pas d'information
Interface exposée aux applications	REST
Device management / Auto-configuration	Non / Semi-automatique
Simulations, expérimentations, prototypage	Prototype

3.2.3 Propositions de plateformes fédératrices pour la supervision et le contrôle

Plateforme FIWARE La plateforme pour l'internet des objets FIWARE du programme européen éponyme [1] est composé d'un ensemble d'éléments appelés "Generic Enablers" [5] qui ont pour objectifs d'être réutilisables afin de permettre une médiation des données des capteurs et des accès aux actionneurs. A titre d'exemple, il existe des Generic Enablers dédiés au device management, à la confidentialité et à la sécurité des données présents dans la plateforme, à la persistance des données, etc.

En terme de niveau d'abstraction, la plateforme FIWARE consolide les données des capteurs sous la forme des entités. Le modèle choisi s'articule autour du standard Next Generation Services Interface [123] qui a été défini par l'Open Mobile Alliance.

En terme de sécurité, FIWARE repose sur l'utilisation de tokens d'accès pour que les applications puissent interagir avec la plateforme. De plus la plateforme supporte le protocole HTTPS pour le chiffrement des communications.

Critère / Plateforme	FIWARE
Niveau d'abstraction / Présence de métadonnées	Consolidation sous la forme d'entités / Métadonnées formelles
Périmètre de couverture	Tout domaine d'application et tout type de réseaux de capteurs
Sécurité / confidentialité	Oui / Oui
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Oui / Non
Infrastructure de déploiement	Cloud et passerelles (possibilité d'edge computing)
Interface exposée aux applications	REST
Device management / Auto-configuration	Oui / Non
Simulations, expérimentations, prototypage	Expérimentations dans plusieurs villes

Plateforme du projet européen"ALMANAC" La plateforme pour les villes intelligentes [27] issue du projet européen ALMANAC [13], vise à intégrer les silos de la ville intelligente c'est-à-dire les différents réseaux de capteurs et d'actionneurs présents dans la ville ainsi que les données ouvertes provenant de la municipalité.

Une architecture composée de trois couches est proposée, qui permet : (i) de communiquer avec les capteurs et actionneurs (via des passerelles) ; (ii) d'enrichir des données de capteurs et d'actionneurs par des méta-données sémantiques provenant de la réutilisation d'ontologies (SSN pour les capteurs, GeoNames [166] pour la

5. https://catalogue.fiware.org/enablers

géographie, Places [161] pour les lieux publics, etc.) ; (iii) d'exposer une API REST aux applications pour interagir avec ce modèle riche qui se focalise sur les capteurs, leurs données et les différentes informations contextuelles utilisées pour enrichir celles-ci.

L'API REST permet également de définir des requêtes complexes afin d'agréger des données de capteurs ou envoyer des commandes à plusieurs actionneurs. Un catalogue de ressources REST permet aux applications de découvrir les différentes ressources qu'elles souhaitent utiliser et de les sélectionner en fonction des métadonnées présentes dans les modèles (exemple : une application souhaite obtenir les données des capteurs d'un lieu géographique précis).

L'authentification et l'autorisation des applications sont gérées dans la plateforme par un module nommé "Federated Identity and Access Manager" (FIAM). La confidentialité des données est gérée au travers de politiques de confidentialité préalablement configurées pour chaque device permettant de contrôler l'accès des applications aux données des capteurs (exemple : masquer les données de géolocalisation lorsque des applications accèdent à des devices en particulier). Le standard XACML [114] est utilisé par la plateforme pour définir ces politiques de confidentialité.

Le FIAM gère également l'authentification des applications au travers de communications réseaux avec le protocole TLS. Le standard SAML [34] est utilisé pour définir les différentes autorisations auxquelles les applications peuvent prétendre.

Une expérimentation a été effectuée dans la ville de Turin où la plateforme ALMANAC a été déployée afin de superviser 28000 poubelles publiques. Le but de cette expérimentation était de montrer que la plateforme peut gérer un grand nombre de capteurs et de données.

Critère / Plateforme	ALMANAC
Niveau d'abstraction / Présence de métadonnées	Relais de données structurées / Métadonnées formelles
Périmètre de couverture	Tout domaine d'application et tout type de réseaux de capteurs
Sécurité / confidentialité	Oui / Oui
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Oui / Non
Infrastructure de déploiement	Cloud et passerelles
Interface exposée aux applications	REST
Device management / Auto-configuration	Oui / Non
Simulations, expérimentations, prototypage	Expérimentation à Turin pour superviser 28 000 poubelles

Plateforme du projet européen "iCore" Dans le cadre du projet européen iCore [83], une plateforme qui adresse deux problématiques du domaine de l'internet des objets a été développée [168, 149]. La première problématique est liée à l'hétérogénéité des capteurs et actionneurs présents dans l'environnement (ville intelligente, bâtiment intelligent, etc.) et dont une vue abstraite doit être offerte afin que les applications de supervision et de contrôle puissent émerger. La seconde est relative à la configuration automatique des applications (en fonction des exigences de celles-ci) qu'assure la plateforme.

La plateforme iCore abstrait l'hétérogénéité des capteurs et actionneurs par des proxies qui leur sont associés, appelés "Virtual Objects" (VOs), et qui représentent leurs fonctionnalités (capteur de présence, actionneur d'éclairage). Par exemple, un VO représentant un capteur de présence peut être attaché à tout capteur qui détecte une présence (caméra, capteur ultrason, magnétique, etc.).

Un second niveau d'abstraction centré sur les éléments physiques et l'agrégation de VOs est proposé via des "Composite Virtual Objects" (CVOs). Par exemple un CVO qui représente les places d'un parking est issu de la composition des VOs représentant les capteurs de présence de la zone géographique . Des méta-données sémantiques (localisation, unité, temps, etc.) sont associées aux VOs et CVOs pour qu'elles soient génériques. Les VOs et les CVOs sont auto-configurables, leurs associations avec les capteurs et actionneurs sont effectuées de manière automatique par la plateforme via des techniques relevant de cognition à propos lesquelles peu de détails sont donnés.

L'utilisation des VOs et CVOs est transparente pour les applications. Elles envoient des requêtes comportant des exigences à satisfaire par la plateforme ; par exemple, trouver le chemin le plus rapide pour atteindre une zone géographique précise. La plateforme se charge alors de convertir cette requête, de découvrir et de sélectionner automatiquement les CVOs et VOs qui sont les plus pertinents pour satisfaire les exigences de l'application. Par exemple, en sélectionnant les routes (CVOs associés aux routes) où le trafic est le moins dense (VOs associés aux capteurs de présence). Il est à noter que pour interagir avec les VOs et CVOs, les applications utilisent une interface REST

Des modules sont présents dans la plateforme afin de gérer l'authentification des applications et leurs autorisations. Les différents acteurs de la ville peuvent définir des règles de contrôle d'accès aux VOs et CVOs qui sont liées aux capteurs et actionneurs qu'ils possèdent. Lorsqu'il s'agit de données potentiellement confidentielles (exemple : géolocalisation d'un véhicule), la plateforme permet aux détenteurs de ces données de définir un degré d'anonymisation de celles-ci (exemple : géolocalisation approximative). Par ailleurs, en terme de sécurité, la plateforme supporte les communications chiffrées vers les devices et les applications. L'intégrité des données collectées est également mentionnée comme une problématique adressée.

Enfin, la plateforme a été prototypée autour de huit cas d'usages différents, liés entre autres, à la ville intelligente (sécurité urbaine, transport, etc.), au bâtiment intelligent (réservation de salles de réunions) et à la maison intelligente (assistant de vie pour les personnes âgées). Le but était de valider la pertinence des différentes

idées et concepts proposés dans cette plateforme, tels que ceux relatifs à l'auto-configuration des VOs et CVOs et à la sélection de ceux-ci.

Critère / Plateforme	iCore
Niveau d'abstraction / Présence de métadonnées	Consolidation sous la forme d'entités / Métadonnées formelles
Périmètre de couverture	Tout domaine d'application et tout type de réseaux de capteurs
Sécurité / confidentialité	Oui / Oui
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Oui / Non
Infrastructure de déploiement	Non définie
Interface exposée aux applications	REST
Device management / Auto-configuration	Oui / Oui
Simulations, expérimentations, prototypage	Prototype (8 cas d'usages)

Plateforme "OM2M" La plateforme OM2M [8] implémente l'architecture de référence décrite par l'organisme de standardisation OneM2M [11]. Elle fait partie du projet "Iot Eclipse" [140] qui appartient à la fondation Eclipse [56] et qui vise à promouvoir des solutions open-source pour les développeurs souhaitant concevoir des applications de l'internet des objets.

Le but de cette plateforme est d'être interopérable avec n'importe quel matériel (devices, passerelles, serveurs) qui implémente également cette architecture de référence, tout en permettant une intégration aisée des devices hétérogènes qui n'implémentent pas l'architecture en question. Les applications interagissent avec la plateforme par le biais d'un arbre de ressources REST, nommé Service Capabilities Layer, via lequel elles obtiennent les données des capteurs et peuvent envoyer des commandes aux actionneurs. L'interaction entre la plateforme et les devices pourvus de capteurs et/ou d'actionneurs s'effectue au travers de cet arbre de ressources où les devices s'enregistrent auprès de la plateforme et envoient leurs données dans une ressource REST qui leur est dédiée.

La plateforme OM2M se focalise principalement sur les données des capteurs et les actionneurs (et leurs données et commandes respectives). Une agrégation de ceux-ci peut être effectuée par la plateforme sous forme de services. Ces services se retrouvent dans l'arbre de ressources REST exposé aux applications. Par exemple une application pourra interagir avec l'une de ces ressources afin d'éteindre différents lampadaires ou pour obtenir une moyenne de température. Ainsi, le niveau d'abstraction proposé repose principalement sur la notion de capteurs et d'actionneurs. A défaut d'autres plateformes telles que iCore qui s'en affranchissent en proposant une abstraction qui se focalise sur les éléments physiques.

Une ontologie nommée IoT-O a été proposée par les concepteurs de la plateforme [7]. Elle se base sur la réutilisation de nombreuses ontologies (DUL, SSN, MSM, etc.) pour décrire les concepts liés aux capteurs, actionneurs et services. La plateforme utilise cette ontologie, à l'aide de règles SWRL [81] (Semantic Web Rules Languages), afin que les applications puissent être configurer de manière autonomique. Par exemple, une application surveillant la luminosité d'un endroit peut être re-configurée autonomiquement lorsque de nouveaux capteurs qui influent sur la luminosité sont présents dans l'endroit en question. Cette plateforme permet également d'effectuer du device management. Elle implémente le standard OMA-DM [122] qui est un protocole conçu pour la gestion des devices et qui permet l'envoi d'opérations à des devices pour, par exemple, re-configurer leur firmware, suivre leur état de fonctionnement, etc.

Les communications HTTPS entre les applications et la plateforme sont supportées au travers d'une authentification basée sur le protocole TLS-PSK. Une extension à la plateforme a été proposée pour traiter des aspects liés à la confidentialité [155], où des règles d'accès peuvent être spécifiées pour chaque ressource REST exposée permettant à des applications d'interagir avec celle-ci ou non. Le principe présenté se repose sur la notion de rôle où une politique est associée à une ressource définissant les rôles pouvant accéder à la ressource en question. Les applications peuvent avoir un ou plusieurs rôles permettant d'identifier si elles peuvent accéder aux ressources en fonction des politiques liées à chacune d'elle.

Plusieurs expérimentations ont été menées dans un bâtiment intelligent afin d'expérimenter les fonctionnalités de la plateforme. Bien qu'à ce jour il n'y ait eu de déploiement significatif de la plateforme, le rayonnement du projet "Iot Eclipse", dont elle fait partie, et le support de l'organisme de standardisation OneM2M font que nous considérons cette plateforme comme semi-opérationnelle.

Critère / Plateforme	OM2M
Niveau d'abstraction / Présence de métadonnées	Relais de données structurées / Métadonnées formelles
Périmètre de couverture	Tout domaine d'application et tout type de réseaux de capteurs
Sécurité / confidentialité	Oui / Oui, prévu
Partage des actionneurs / Prise en compte du temps-réel et des accès concurrents	Oui / Non
Infrastructure de déploiement	Cloud et passerelles (possibilité d'edge computing)
Interface exposée aux applications	REST et MQTT
Device management / Auto-configuration	Oui / Oui
Simulations, expérimentations, prototypage	Plateforme semi-opérationnelle

3.2.4 Simulateurs et expérimentations existantes

Expérimentations Plusieurs expérimentations ont été menées autour de la ville intelligente. Nous en choisissons un échantillon afin de montrer que ces expérimentations ne sont pour le moment que peu, voire pas, destinées à permettre le partage des données entre les acteurs de la ville.

SmartSantander [147] est une expérimentation menée dans la ville de Santander en Espagne. Plus de 2000 capteurs ont été déployés dans la ville afin de superviser le taux d'occupation des places de parking et le taux d'irrigation des parcs et jardins et d'obtenir des mesures environnementales (qualité de l'air, bruit, luminosité). Il s'agit de capteurs déployés pour les besoins de l'expérimentation. L'un des buts de la plateforme SmartSantander est de promouvoir une plateforme fédératrice ouverte où les différentes données des capteurs sont partagées entre les différentes applications.

La plateforme permet d'authentifier les applications qui ont des autorisations d'accès à certaines données de capteurs. Les applications doivent au préalable s'enregistrer dans la plateforme pour interagir avec celle-ci. Il s'agit uniquement d'applications de supervision de la ville intelligente. Par ailleurs, l'infrastructure de déploiement de la plateforme SmartSantander se compose de capteurs, de passerelles relayant les données des capteurs ainsi que de serveurs hébergés dans des centres de données.

Une expérimentation a été menée dans la ville de Padoue en Italie où plus de 300 capteurs ont été déployés [173]. Les capteurs déployés mesurent l'intensité de la lumière émise par les lampadaires, l'humidité et la température de l'air. Une seule application a été développée pour superviser les différents lampadaires, les mesures des autres capteurs n'étant pour le moment pas utilisées. Au contraire de Santander, il s'agit d'une expérimentation fermée puisque le but de cette expérimentation n'est pas de partager les données des différents capteurs. L'objectif de l'expérimentation menée à Padoue est de montrer techniquement la faisabilité d'une plateforme pour la ville intelligente. Ainsi, des choix techniques sont discutés : base de données à utiliser dans une plateforme, protocole réseau pour la communication avec les capteurs et les applications, etc. Cependant, il est évoqué dans les perspectives de cette expérimentation d'intégrer des données de différents systèmes dédiés (parking, transport, trafic routier, etc.). Nous pouvons noter que l'infrastructure de déploiement utilisée dans cette expérimentation se compose de capteurs et de passerelles transférant les données des capteurs vers des serveurs dont la nature de l'hébergement n'est pas précisée.

Enfin, une expérimentation est également en cours dans la ville de Barcelone où plus de 1800 capteurs ont été déployés [158]. Le taux de remplissage des poubelles, l'occupation des places de parking, le bruit, la température et l'humidité de l'air sont les objets des mesures des capteurs. Tout comme l'expérimentation dans la ville de Padoue, il s'agit également d'une expérimentation fermée puisque l'objectif n'est pas que les données des capteurs soient partagées. Le but est de montrer la faisabilité technique d'une plateforme pour la ville en décrivant l'architecture de la plateforme et le volume de données que celle-ci peut traiter. Compte-tenu du

caractère récent de l'expérimentation, peu d'informations sont données à ce propos. Nous notons que l'infrastructure de déploiement utilisée est la même que dans les deux expérimentations décrites précédemment.

Nous pouvons remarquer que ces expérimentations sont uniquement destinées à la supervision de la ville intelligente. Par ailleurs, hormis le projet SmartSantander, les différentes expérimentations ne font pas état de l'utilisation d'une plateforme fédératrice. Cependant, nous pouvons noter que le point commun de ces expérimentations est qu'elles possèdent toutes une infrastructure de déploiement composée de passerelles et de serveurs. Nous reviendrons sur ce point dans la section 4.1.4.1 afin de montrer que les architectures de déploiement actuelles sont conformes avec nos objectifs de contrôle.

Simulateurs Afin de pouvoir de pouvoir évaluer notre travail, nous avons étudié plusieurs simulateurs en lien avec la ville intelligente et l'internet des objets. Notre but étant de pouvoir simuler des valeurs de capteurs, du temps, des commandes envoyées aux actionneurs ainsi que des défaillances matérielles des actionneurs.

Nous avons identifié qu'une grande partie des simulateurs pour l'internet des objets étaient proposés pour simuler des données de capteurs envoyées par des capteurs dont les caractéristiques réseaux ou énergétiques sont configurables. Ces simulateurs relèvent du domaine des réseaux de capteurs puisque ces caractéristiques sont relatives à la portée du signal [128] des capteurs ainsi qu'à leur consommation énergétique et au ratio de paquets réseaux pouvant être perdus [29]. De ce fait, ces simulateurs sont trop spécifiques à un domaine particulier et ne répondent pas à nos attentes.

Nous nous sommes également aperçus qu'il existait beaucoup de simulateurs dédiés à certains domaines de la ville intelligente. Il existe, par exemple, des simulateurs dans le domaine du transport [82] pour simuler le trafic routier. De même, il existe des simulateurs dans le domaine du Smart Grid [88] permettant de simuler la production ainsi que la consommation d'énergie d'électrique d'appareils.

Nous avons évalué le simulateur Picse [144] qui n'est pas lié à un domaine particulier. Il permet la création programmatique de capteurs et d'actionneurs ainsi que la simulation d'envoi de données et de commandes. Cependant, ce simulateur ne comporte pas d'interface graphique et n'inclut pas de gestion du temps. De plus, très peu de documentation est disponible à propos de celui-ci.

Nous avons également identifié l'existence du simulateur DiaSim [32] qui permet de programmatiquement créer des capteurs et actionneurs et de simuler l'envoi de données et de commandes. Ce simulateur possède une interface graphique et permet de simuler la vitesse du temps qui s'écoule. Malheureusement, il est conçu pour le domaine de la maison connectée et un important travail aurait dû être effectué pour qu'il soit adapté à la ville intelligente.

Enfin, nous avons également pris connaissance du simulateur open source SCSimulator [148]. Il permet de créer des capteurs de différents types (température, par exemple) et de les déployer virtuellement dans une carte Google Maps. Il permet

également de simuler du temps virtuel. Cependant, il ne permet pas de créer des actionneurs et de simuler leurs défaillances.

Ainsi, puisque qu'aucun ne correspond à nos attentes , nous avons décidé de concevoir notre propre simulateur (cf. section 7.2.3).

3.3 Comparaison entre les diverses plateformes et résumé

3.3.1 Comparaison de nos travaux avec les plateformes opérationnelles

Niveau d'abstraction et méta-données Nous pouvons constater qu'en terme de niveau d'abstraction, les plateformes opérationnelles servent principalement à relayer les données brutes des capteurs ou à faiblement les agréger et consolider. Ainsi, le niveau d'abstraction de ces plateformes se focalise sur les données des capteurs ("stream", "flux", "channel", etc.) qui sont exposés aux applications.

Les plateformes ajoutent quelques méta-données à leurs modèles, cependant celles-ci sont définies de manière ad-hoc. Par exemple, un capteur d'humidité sera mentionné comme tel par une méta-donnée "type_sensor=humidity" dans une plateforme et par une méta-donnée "type=humi" dans une autre plateforme. Outre des éléments informatifs supplémentaires pour les développeurs d'applications, cela ne permet pas à des machines de pouvoir utiliser ces méta-données qui ne reposent sur aucun standard.

Certaines catégories d'applications (exemple : analyse prédictive du trafic routier) peuvent se contenter de ce type de données brutes et/ou faiblement consolidées de la part des plateformes. Cependant, certaines applications n'ont pas nécessairement besoin de connaître les données mêmes des capteurs mais ce qu'elles représentent (exemple : savoir qu'une place est vide plutôt que la donnée du capteur même l'indiquant). Dans cette thèse, nous adressons ce type d'application qui requiert une consolidation des données effectuée par la plateforme.

Nous verrons en section 4.1.1.1 que catégorisons ces plateformes sous le nom de "plateforme de relais de données", en tenant compte du niveau d'abstractions qu'elles proposent, afin de positionner nos travaux.

Périmètre de couverture, infrastructure, partage des actionneurs Les plateformes opérationnelles ont des périmètres de couverture restreints car elles adressent des capteurs : soit qui sont la propriété d'un fabricant d'équipements spécifique (exemple : Netatmo), soit qui possèdent une connectivité propriétaire (exemple : Sigfox), soit car la connectivité requise pour que les capteurs communiquent avec la plateforme se base sur le protocole IP (exemple : Xively, ThingSpeak, etc.) qui est consommatrice en ressources pour ces matériels contraints en énergie.

Ces plateformes sont hébergées au sein d'une infrastructure de cloud computing, exception faite pour la plateforme Sigfox qui comprend des passerelles mais dont

leur utilité n'est que de faire transiter les données vers l'infrastructure cloud. Notre travail traite de la problématique des applications de contrôle temps-réel. Nous avons identifié que ces infrastructures ne permettent pas de satisfaire les exigences de ces applications (cf. section 4.1.2).

Nous constatons que les plateformes opérationnelles mettent l'accent sur le partage des données de capteurs et non sur le partage des accès aux actionneurs. Ce constat est cohérent avec l'infrastructure de déploiement qu'elles utilisent.

Au contraire, nous visons dans notre travail de permettre aux applications contrôle temps-réel d'utiliser une plateforme partagée pour contrôler la ville.

Interface exposée Pour fournir une interface d'accès aux données des capteurs collectées, l'utilisation du style architectural REST est privilégié par les plateformes opérationnelles. Ce choix d'interface est également le même pour les autres plateformes que nous avons comparé De ce fait, nous reviendrons sur ce point dans la section 3.3.2 suivante qui est consacrée également à ce critère.

Il est à noter que MQTT est également présent dans les interfaces exposées des plateformes opérationnelles en complément des APIs REST. Ce choix s'explique par le fait que HTTP est utilisé comme protocole applicatif lorsqu'une API REST est exposée, lequel ne supporte pas l'envoi d'une notification à un client, au contraire du protocole MQTT.

Sécurité et confidentialité La sécurité est un critère que toutes les plateformes opérationnelles comparées adressent compte-tenu du fait qu'elles soient actuellement déployées. Le chiffrement des communications est l'un des éléments principaux permettant à ces plateformes de pouvoir garantir la sécurité des données. Ce chiffrement a lieu au travers de l'utilisation du protocole sécurisé HTTPS ainsi que via l'utilisation de clé privée intégrée aux devices.

Ces plateformes intègrent également des mécanismes relatifs à l'authentification des applications, via par exemple le protocole OAuth2, et ont la possibilité de définir des autorisations au travers de l'utilisation d'API Key. Toutefois, la définition de ces autorisations restent sommaire puisqu'il n'est possible que de définir des droits de lecture et d'écriture.

En revanche, la confidentialité n'est pas adressée dans les plateformes opérationnelles que nous avons pris pour échantillon. Il est de la responsabilité des utilisateurs de la plateforme de s'assurer que les données qu'ils partagent ne sont pas dangereuses pour la vie privée d'autrui.

Nous verrons dans la section suivante qu'à l'inverse des plateformes opérationnelles, les plateformes qui ont été proposées mettent plus l'accent sur la gestion de la confidentialité que sur la sécurité. Il s'agit de problématiques que nous n'adressons pas notre travail mais qui doivent être considérées dans la conception d'une plateforme horizontale.

Device Management et auto-configuration Le device management est une fonctionnalité présente dans l'ensemble des plateformes opérationnelles puisqu'elle permet aux clients de ces plateformes et à leurs applications d'effectuer la maintenance de leurs devices à distance et de manière centralisée.

En revanche, l'auto-configuration n'est pas adressée puisqu'elle relève, à l'heure actuelle, du domaine de la recherche.

3.3.2 Comparaison de nos travaux avec les plateformes fédératrices proposées

Niveau d'abstraction et méta-données Les plateformes fédératrices qui ont été proposées pour l'internet des objets et la ville intelligente proposent de fournir une abstraction homogène aux applications de la grande diversité des capteurs et actionneurs.

Le niveau d'abstraction fournit par ces plateformes, exceptions faites pour les plateformes iCore, SPITIFIRE et FIWARE, se concentre sur la description des capteurs et des actionneurs. Au contraire des plateformes opérationnelles, l'abstraction proposée relève de l'agrégation de données ainsi que, parfois, de la consolidation de celles-ci. Des méta-données viennent enrichir cette abstraction par le biais de la ré-utilisation d'ontologies qui sont issues d'organismes de standardisation (SSN standardisé par le W3C, par exemple) et qui sont décrites par un langage formel de description d'ontologies (RDF, OWL, etc.). Ces méta-données permettent aux applications d'utiliser des opérations de filtrage et d'agrégation lorsqu'elles accèdent aux données des capteurs et aux actionneurs.

Comme évoquée précédemment, ce niveau d'abstraction n'est pas nécessaire à toutes les applications. Par exemple, une application souhaitant connaître la densité du trafic routier d'une route n'a pas nécessairement besoin d'obtenir les données de chaque capteur mesurant cette densité. En effet, une information consolidée de ces capteurs sous la forme de taux d'occupation peut être suffisante pour une telle application.

Nous proposons un niveau d'abstraction des capteurs et actionneurs permettant aux applications de facilement contrôler et superviser la ville intelligente. Le niveau d'abstraction que nous proposons se base sur la consolidation des données des capteurs et actionneurs au travers d'entités via lesquelles nous modélisons les éléments physiques de la ville (cf. section 5.1). Contrairement aux plateformes iCore et SPITFIRE, nous utilisons des modèles génériques décrits sous la forme d'automates à états finis puisque la ville intelligente est similaire à un système réactif (cf. section 5.2).

Périmètre de couverture et infrastructure Les plateformes fédératrices qui sont à un stade moins avancé que les plateformes opérationnelles ne se restreignent pas à quelques domaines d'applications de la ville intelligente ni à l'utilisation de capteurs propriétaires. De même, la multitude de capteurs et actionneurs présents dans la ville intelligente est adressée par ces plateformes fédératrices peu importe

leur connectivité. Par conséquent, cela permet à ces plateformes de couvrir les divers domaines d'applications existants dans la ville intelligente. Le périmètre de couverture des plateformes fédératrices est liée au fait que ces plateformes se basent sur d'autres plateformes pour obtenir des données de capteurs ou envoyer des commandes aux actionneurs, comme nous le décrivons en section 4.1.2.

Nous remarquons que les plateformes qui ont un large périmètre de couverture possèdent une infrastructure de cloud computing et des passerelles. Ceci s'explique par le fait que les passerelles permettent à la plateforme d'intégrer les données des capteurs et/ou actionneurs ne disposant pas d'une connectivité IP puisque ces données peuvent être relayées vers l'infrastructure cloud.

Cependant, les passerelles de ces plateformes sont utilisées principalement pour faire transiter les données dans un sens comme dans l'autre des capteurs et des actionneurs. Ces passerelles pourraient être également utilisées en tant qu'unité de calcul à part entière afin de former l'infrastructure edge. La plateforme FIWARE utilisant une infrastructure edge, nous présentons en section 4.1.3 comment s'insère notre travail par rapport à celle-ci.

Partage des actionneurs et prise en compte du temps-réel De manière générale, ces plateformes font peu de partage des actionneurs. De plus, aucunes d'elles ne prend en compte les contraintes temps-réel permettant à des applications de contrôle temps-réel d'exister. Seules des applications ne nécessitant pas ces contraintes (exemple : certaines applications d'éclairage public) sont adressées par ces plateformes.

Notre travail adresse les problématiques liées au partage des actionneurs et à la prise en compte du temps-réel, qui nous conduit à proposer des mécanismes liés à celles-ci (cf. chapitre 6). Il s'agit d'un critère différenciateur par rapport aux plateformes horizontales qui, compte-tenu de l'infrastructure qu'elles utilisent, ne peuvent adresser les applications de contrôle temps-réel puisque les passerelles sont uniquement utilisées comme relais de données.

Notre travail se base sur l'infrastructure utilisée par la plateforme FIWARE qui inclut du edge-computing. Cette infrastructure permet l'existence d'applications de contrôle temps-réel dès lors que des garanties leurs sont données comme nous le décrivons en section 4.3.

Interface exposée De la même manière que pour les plateformes opérationnelles, toutes les plateformes horizontales comparées exposent au minimum une API REST aux applications. Puisqu'une architecture REST repose, entre autre, sur le concept d'interface uniforme (cf. section 2.1.1.1), les applications peuvent ainsi interagir uniformément avec l'API exposée par la plateforme. De plus, ce style architecture est très utilisé dans le domaine de l'internet des objets et de la ville et demeure un standard de fait (cf. section 2.1.1.2)

Nous avons également choisi d'utiliser ce style architectural, non seulement pour les mêmes raisons citées que précédemment mais également car la notion de res-

sources est en adéquation avec notre choix de modélisation. De ce fait, nous avons choisi d'utiliser une architecture orientée ressource dans notre travail (cf. section 4.4).

Sécurité et confidentialité Contrairement aux plateformes opérationnelles, la sécurité n'est pas prise en compte par l'intégralité des plateformes de cette catégorie qui sont à un stade moins avancé. Ceci s'explique par le fait que certaines des ces plateformes relèvent du prototype et que l'accent est mis sur certaines fonctionnalités plutôt que d'autres, comme c'est le cas dans la plateforme CityHub qui se focalise sur l'infrastructure et le modèle de données. Il est à noter qu'aucune plateforme hormis la plateforme iCore ne traite de la sécurité dans la plateforme même, c'est-à-dire le fait que les données soient intègre au sein même de la plateforme via, par exemple, l'utilisation de signature ou de chiffrement homomorphe [63] des données dans la plateforme comme proposés dans d'autres domaines [99].

La gestion de la confidentialité des données est, contrairement aux plateformes opérationnelles, une problématique adressée par quelques unes des plateformes que nous avons comparées bien que ce sujet soit traité de manière marginale. Les aspects légaux relatifs à la gestion de la confidentialité [107] des données ajoutent un degré de complexité à la conception de plateforme partagée. Nous pouvons constater que la confidentialité est en général gérée dans les plateformes comparées par des règles de contrôle d'accès définies par les fournisseurs des données. Cependant, le contrôle d'accès, comme décrit en [156], n'est pas le seul moyen d'assurer la confidentialité des données.

Bien que la sécurité et la confidentialité soient des thématiques importantes où demeurent diverses problématiques à résoudre [4], nous n'en traiterons pas dans notre travail. Cependant, elles sont prépondérantes dans la conception d'une plateforme horizontale pour la ville intelligente et l'internet des objets.

Device Management et auto-configuration Le device management n'est pas une problématique nouvelle puisque des standards sont déjà présents l'industrie (TR-069 [165], OMA-DM [122]) pour permettre de gérer des devices. Par conséquent, les plateformes proposées pour l'internet des objets et la ville intelligente n'adressent pas ou peu ce sujet qui est déjà couvert dans des plateformes opérationnelles. Pour ces mêmes raisons, la thématique du device management n'est pas développée dans ce manuscrit.

L'auto-configuration est une thématique ambitieuse mais peu traitée dans les plateformes choisies bien qu'elle permettent aux applications de se configurer dynamiquement peu importe les changements qui surviennent au sein de la ville intelligente. A notre sens, l'auto-configuration de la même façon que les autre propriétés d'auto-adaptation [146] constituent une évolution future des plateformes horizontales que nous ne traitons pas dans cette thèse.

Chapitre 4

Contributions à une plateforme de l'Internet des objets pour la ville intelligente

Ce chapitre vise à décrire le contexte des contributions réalisées ainsi que le travail d'identification que nous avons réalisé, permettant à la fois de cerner les exigences relatives à notre travail et d'apporter des premiers éléments de réponse.

Après avoir classifié les plateformes qui existent (cf. section 4.1.1), dont certaines sont décrites dans le chapitre précédent, nous positionnons notre travail par rapport aux infrastructures utilisées par celles-ci (cf. section 4.1.2). Notre travail s'insère dans la plateforme FIWARE qui utilise une infrastructure edge. Nous présentons nos objectifs (cf. section 4.1.3) : favoriser à la "plateformisation" des applications de contrôle temps-réel dans la ville intelligente ainsi que l'ouverture de la plateforme aux différents acteurs. Ces objectifs sont en phase avec notre identification des architectures de déploiement actuelles dans les diverses villes expérimentales (cf. section 4.1.4).

Sur la base d'un exemple typique de ville que nous avons élaboré (cf. section 4.2.1), nous décrivons des cas d'usages d'applications de supervision et de contrôle de la ville intelligente (cf. section 4.2.2). A partir de ces scénarios, nous formulons les besoins que peuvent avoir les applications utilisant une plateforme fédératrice. Nous présentons également des scénarios relatifs à l'arrivée de nouveaux opérateurs dans la ville (cf. section 4.2.3) pour décrire les exigences relatives à l'ouverture de la plateforme.

Puisque notre travail vise à favoriser la "plateformisation" d'applications de contrôle temps-réel, nous décrivons les exigences que cela requièrent vis-à-vis des latences des diverses communications réseaux (cf. section 4.3.1) mais également vis-à-vis des solutions que l'on propose dans notre travail (cf. section 4.3.2).

Enfin, nous l'intégration de notre travail dans une architecture orientée ressources (cf. section 4.4). D'une part car REST est un choix idéal pour standardiser les interactions avec la plateforme, mais également car nous avons identifié qu'il existait une adéquation entre le système que nous souhaitons modéliser et la notion de ressources.

Sommaire

4.1 Insertion et positionnement du travail

4.1.1 Positionnement par rapport aux plateformes existantes

On propose de distinguer deux catégories de plateforme pour la ville intelligente et l'internet des objets : les plateformes de relais de données ainsi que les plateformes fédératrices. Toutes deux ont été décrites en section 3.2. Comme illustré par la figure 4.1, ces deux types de plateformes n'ont pas les mêmes objectifs.

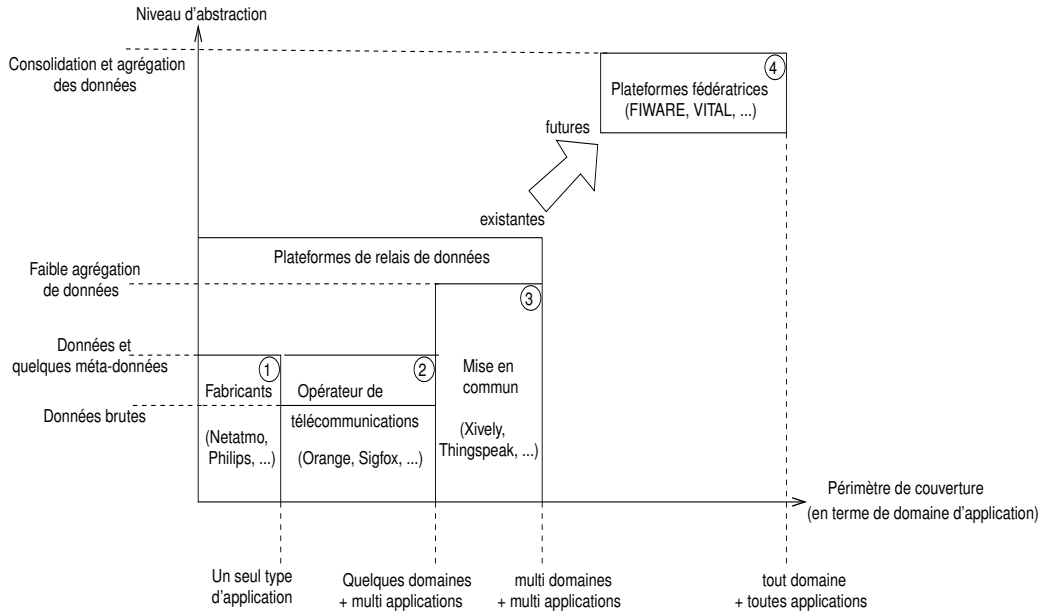


FIGURE 4.1 – Distinction entre les plateformes de relais de données et les plateformes fédératrices. L'axe des abscisses indique la couverture des plateformes en terme de domaine d'application. L'axe des ordonnées représente le degré d'abstraction des capteurs et actionneurs que propose les plateformes aux applications.

Les plateformes de relais de données offrent aux applications un faible niveau d'abstraction puisqu'elles ne consolident que peu ou pas les données des capteurs et ne cherchent donc pas ou peu à les abstraire. Il en est de même pour les commandes envoyées aux actionneurs. A l'inverse, les plateformes fédératrices cherchent à consolider et abstraire les données de capteurs ainsi que les commandes envoyées aux actionneurs en se focalisant sur ce qu'elles représentent. Quant aux domaines d'applications couverts, les plateformes fédératrices visent à être génériques et non dédiées à un domaine d'application particulier à l'instar de certaines plateformes de relais que nous décrivons par la suite.

Nous décrivons plus en détails ces deux types de plateformes dans les sous-sections suivantes.

4.1.1.1 Plateformes de relais de données

Plateformes des fabricants d'équipements La première catégorie de plateformes de relais données est celle appartenant à des fabricants d'équipements (Netatmo [118], Philips Hue [136], etc.), et portent le numéro 1 dans la figure 4.1. Nous en avons décrit un échantillon en section 3.2.1.2.

Ces plateformes sont destinées à l'usage exclusif des capteurs et actionneurs intégrés aux équipements de ces fabricants. En d'autres termes, les données des capteurs transitent nécessairement par la plateforme du fabricant d'équipements, de même pour l'envoi d'ordres aux actionneurs qui se fait via l'utilisation de cette plateforme. Des méta-données sont définies par le fabricant qui sont attachées aux données de capteurs et aux commandes envoyées aux actionneurs.

Les devices des fabricants sont destinés à une application précise d'un domaine d'application particulier. Par exemple la lampe Philips Hue est conçue pour une application d'éclairage dans la maison intelligente et la station météo Netatmo pour une application de météorologie individuelle. Ainsi, ces plateformes de relais ont une couverture limitée des domaines d'applications comme illustré par la figure 4.1.

Plateformes des opérateurs de télécommunications Des opérateurs de télécommunications ont également leurs propres plateformes de relais de données, référencées par le numéro 2 dans la figure 4.1. Un échantillon de ce type de plateforme a été présenté en section 3.2.1.3.s Ces plateformes sont destinées à collecter les données et/ou à transmettre des commandes uniquement aux capteurs et actionneurs qui utilisent les antennes de transmission des opérateurs. Ces plateformes de relais offrent une faible d'abstraction des données de capteurs et les commandes envoyées actionneurs. Dans le cas de Sigfox, seules des trames réseau transitent par la plateforme et qui doivent être par conséquent analysées.

Ces capteurs et actionneurs peuvent être pour différents usages, tels que des actionneurs intégrés à des lampes pour l'éclairage dans des batiments ou dans la ville, ou encore des capteurs intégrés aux places de parking. Ainsi, les plateformes des opérateurs de télécommunications couvrent plusieurs domaines d'applications différents.

D'autres plateformes similaires existent et sont destinées à certains types de réseaux, par exemple au réseau LoRa [10] ou Sigfox [157]. Cependant, elles sont opérées par des industriels qui ne sont pas des opérateurs de télécommunications (Actility, Cisco, etc.).

Plateformes de mise en commun Enfin, il existe des plateformes de relais qui sont des plateformes de mise en commun de capteurs et d'actionneurs (Xively [172], Thingspeak [164], Carriots [37], etc.), elles portent le numéro 3 dans la figure 4.1. Deux d'entre elles, Xively et Thingspeak, ont été décrites en section 3.2.1.1.

Ces plateformes visent à servir de concentrateur pour interagir avec le plus grand nombre de capteurs et d'actionneurs. Contrairement aux précédentes plateformes, celles-ci intègrent des opérations de consolidation de données tel que du filtrage ou de l'agrégation, permettant ainsi d'offrir un niveau d'abstraction supérieur aux simples données de capteurs.

Ces plateformes ne sont pas restreintes à un domaine d'application particulier et par conséquent en couvre un large spectre.

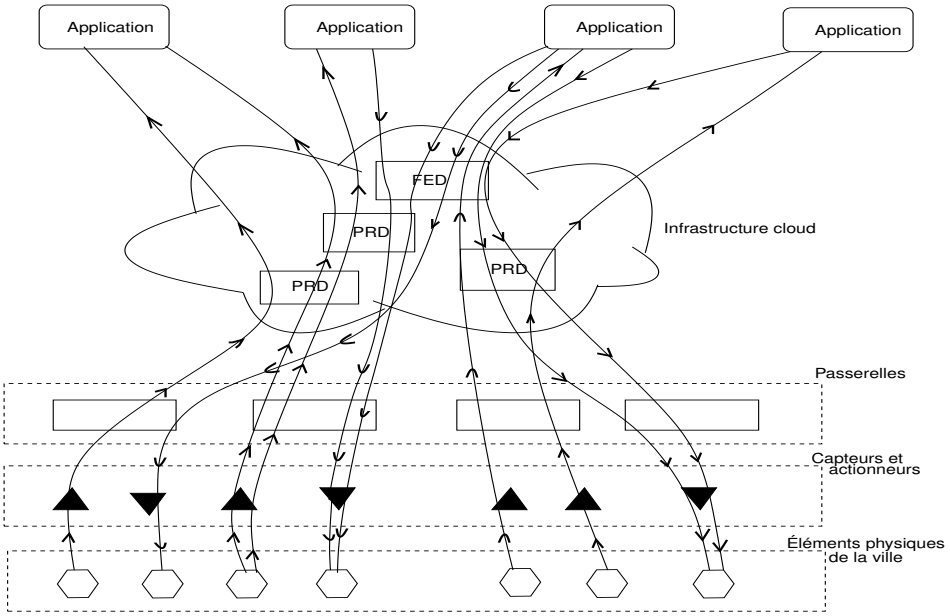
4.1.1.2 Vers des plateformes fédératrices

Les plateformes fédératrices (FIWARE [1], iCore [83], Vital [167], etc.) mutualisent les données des capteurs et les accès aux actionneurs quelque soit le domaine d'application. Ces plateformes sont référencées par le numéro 4 dans la figure 4.1.

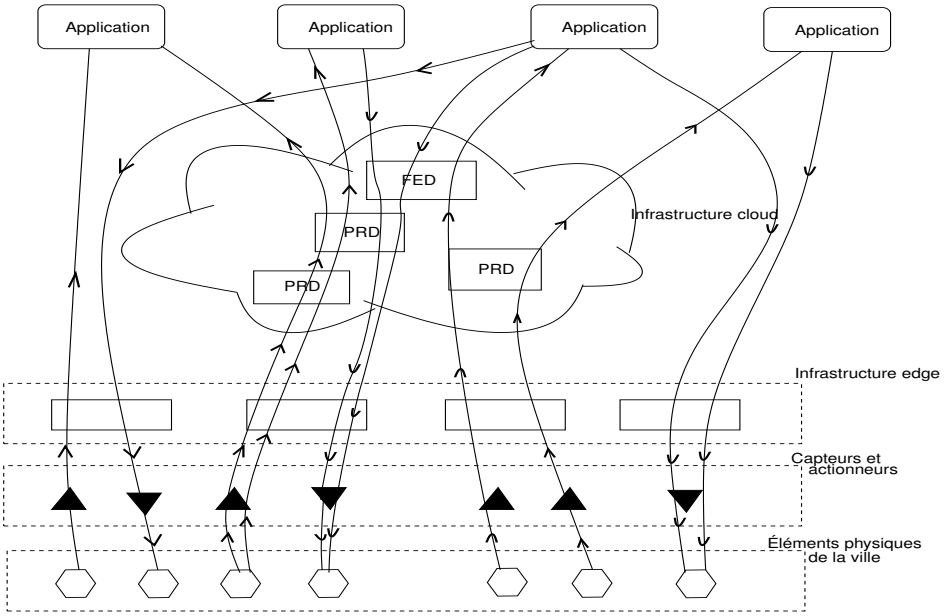
Contrairement aux plateformes de relais qui se focalisent sur les données des capteurs et les commandes des actionneurs, les plateformes fédératrices visent à agréger et consolider celles-ci sous forme de services. Par exemple, le projet européen IoT-A [85] visent à promouvoir un modèle de référence dans l'IoT où est fait la distinction entre les élément physiques ("Physical Entity") d'un environnement (la ville, la maison, etc.) et ceux qui les captent et les actionnent, c'est-à-dire les capteurs et actionneurs.

Les plateformes fédératrices se reposent en partie sur des plateformes de relais pour obtenir des données de capteurs ainsi qu'envoyer des commandes aux actionneurs. Par exemple, la plateforme fédératrice Vital utilise les plateformes de relais de données Xively et ThingWorx. Afin de collecter ou transmettre des données, les plateformes fédératrices peuvent également utiliser des passerelles qui peuvent leur appartenir(cf. section 4.1.2). Ainsi, elles ne dépendent pas uniquement des plateformes de relais.

Notre travail se positionne dans l'utilisation d'une plateforme fédératrice à savoir celle du projet FIWARE où l'objectif est de parvenir à une situation horizontale dans la ville intelligente.



(a) Infrastructure orientée Cloud computing



(b) Infrastructure incluant du edge computing

FIGURE 4.2 – Infrastructures orientées cloud et edge computing. Les abréviations PRD et et FED désignent respectivement les plateformes de relais de données et les plateformes fédératrices. Un rectangle représente le matériel physique sur lequel sont déployés les composants logicielles relatifs à une plateforme fédératrice ou à une plateforme de relais de données. Un triangle plein pointant vers le bas représente un capteur, un triangle pointant vers le haut représente un actionneur. Les rectangles représentent les divers éléments physiques de la ville. Un nuage représente l'infrastructure cloud au sein de laquelle sont hébergées les plateformes fédératrices et les plateformes de relais. Les flèches représentent les communications qui ont lieu soit au travers de réseaux ou soit de manière physique (capteurs/actionneurs et éléments physiques). Le sens des flèches indique s'il s'agit d'ordres envoyés à des actionneurs (flèche vers le bas) ou de données de capteurs remontées (flèche vers le haut).

4.1.2 Identification des infrastructures des plateformes fédératrices existantes

Afin de superviser et contrôler la ville, les applications utiliseront une plateforme fédératrice. Nous avons identifié que ces plateformes étaient majoritairement hébergées au sein d'une infrastructure cloud computing (cf. section 3.2.1 et 3.2.2), comme nous le montrons sur la figure 4.2a. Dans quelques cas ces plateformes sont hébergées dans une infrastructure incluant du edge computing, comme représenté par la figure 4.2b.

4.1.2.1 Infrastructure classique de "cloud computing"

Une majorité de plateformes fédératrices pour la ville intelligente (Vital, CityHub [96], etc.) visent à être hébergées au sein d'une infrastructure cloud. De plus, toutes les expérimentations conduites comportaient une plateforme hébergée sur ce type d'infrastructure.

Comme illustrées sur la figure 4.2a, les plateformes de relais de données sont elles-mêmes hébergées au sein d'une infrastructure cloud computing, dans des centres de données faisant partie d'un cloud public ou privé. Elles peuvent être utilisées par des applications, en fonction des besoins de celles-ci, par exemple pour des applications d'analyse de données qui ne requièrent pas de données consolidées. Les données des capteurs et des actionneurs peuvent transiter par plusieurs plateformes de relais. C'est le cas par exemple des plateformes de mise en commun utilisant des plateformes de propriétaires d'équipements afin d'interagir avec les capteurs et actionneurs de ceux-ci.

Puisque ces plateformes fédératrices sont hébergées dans une infrastructure cloud et peuvent se reposer sur plusieurs plateformes de relais (cf. sous-section 4.1.1.2), le délai d'envoi des commandes aux actionneurs est non-déterministe et varie de plusieurs centaines de millisecondes à quelques secondes.

Ainsi, ces plateformes fédératrices sont destinées principalement à des applications de supervision et à des applications effectuant ce que nous appelons du contrôle simple. Le contrôle simple tolère une variation du délai des communications entre les applications et les actionneurs (centaines de millisecondes à quelques secondes) qui est acceptable pour ce type d'application de contrôle. Par exemple, l'application d'éclairage de parcs publics décrite en [109], ne requiert pas des délais de communication bornés lors d'envoi d'ordre afin d'allumer ou éteindre les lampes du parc. De ce fait, celle-ci utilise une plateforme hébergée au sein d'une infrastructure cloud.

4.1.2.2 Infrastructure incluant du edge computing

Telles qu'illustrées par la figure 4.2b, les infrastructures des plateformes fédératrices peuvent également inclure du edge computing.

Ce matériel proche des capteurs et des actionneurs correspond dans la figure précédente aux matériels edge, qui contrairement aux passerelles de la figure 4.2a

n'ont pas seulement un rôle d'intermédiaire mais peuvent effectuer du calcul et également stocker des données (cf. section 2.1.2).

Les plateformes fédératrices déployées au sein de ce type d'infrastructure ont des briques logicielles déployées à la fois dans des serveurs hébergés dans une infrastructure cloud, et dans du matériel edge. Les plateformes de relais sont également utilisées mais, comme le montre la figure 4.2b, sont moins sollicitées puisque des calculs (réponses aux requêtes des applications, par exemple) sont déjà effectués par le matériel formant l'infrastructure edge computing.

Ce type d'infrastructure permet à des applications de pouvoir accéder aux données des capteurs et d'envoyer des ordres aux actionneurs en s'interfaçant directement avec le matériel situé. De ce fait, puisque les ordres envoyés aux actionneurs par les applications ne transitent plus par des serveurs hébergés dans une infrastructure cloud (cf. section 4.1.2.1), les délais de communication sont alors déterministes et les latences faibles (quelques dizaines de millisecondes à quelques centaines de millisecondes). Ainsi, une infrastructure edge supporte la mise en oeuvre d'applications de contrôle temps-réel.

Il existe peu de plateformes fédératrices, tel que FIWARE, visant à être hébergées au sein de ce type d'infrastructure (cf. section 3.2).

4.1.3 Insertion du travail au sein d'une plateforme fédératrice

4.1.3.1 Objectifs du travail

Notre travail s'insère dans le contexte de la plateforme fédératrice européenne FIWARE, dont nous avons donné une description en section 2.1 et qui est destinée à être hébergée au sein d'une infrastructure incluant du edge computing.

Compte-tenu de l'infrastructure de cette plateforme, notre premier objectif est de permettre que les applications de contrôle temps réel, se basant sur une plateforme horizontale, se développent. Plus de détails sont donnés en section 4.3 où nous décrivons les exigences que nous avons identifiées pour que ces applications de contrôle puissent exister, ainsi que les implications que cela engendre sur les solutions que nous proposons.

Notre second objectif est en lien avec l'aspect horizontal et partagé de cette plateforme fédératrice. Nous cherchons à favoriser l'ouverture de cette plateforme aux multiples acteurs de ville en facilitant l'intégration de nouveaux équipementiers et en facilitant le développement d'applications de supervision et de contrôle pour la ville intelligente.

Afin d'illustrer ces objectifs, nous présentons un exemple typique présent de la ville intelligente. Cet exemple est composé de multiples éléments physiques pourvus de capteurs et actionneurs (cf. section 4.2.1) utilisés par des applications de contrôle et de supervision (cf. section 4.2.2). Nous montrons également que de nouveaux opérateurs doivent pouvoir s'intégrer aisément dans la ville intelligente (cf. section 4.2.3).

4.1.3.2 Éléments non couverts par notre travail

Bien qu'étant connexes aux objectifs de notre travail, nous ne traitons pas des aspects relatifs à l'authentification et aux autorisations des applications qui interagissent avec notre plateforme. Cependant, ceux-ci devraient être considérés pour que la plateforme puisse être partagée entre les différents acteurs. Des moyens relatifs à la sécurisation des échanges avec les applications devraient être fournies afin que nos contributions puissent être prêtes à s'appliquer. Cependant, nous n'aborderons pas ces problématiques relatives à la sécurité puisqu'elles sont éloignées du sujet de thèse.

Pour permettre l'interopérabilité d'applications dans l'internet des objets, l'utilisation de méta-données sémantiques décrites dans des ontologies doit être considérée (cf. section 2.3) . Ces méta-données sémantiques peuvent être gérées dans une plateforme [100] située au-dessus de la plateforme FIWARE. Nous n'en traiterons pas mais mentionnons l'usage de ces méta-données dans les modèles que nous proposons (cf. section 5.5.3) et dans notre mécanisme de coordination (cf. section 6.3.1).

De même, nous avons identifié que les applications pouvaient avoir besoin d'informations géographiques pour qu'elles supervisent et contrôlent la ville intelligente. Nous ne développerons pas de cet aspect, mais nous avons identifié qu'il existait un standard CityGML [91], comportant des modèles spatio-sémantiques qui décrivent la signification des différents éléments physiques de la ville ainsi que leurs coordonnées à différents niveaux de granularité.

D'autre part, une plateforme fédératrice doit posséder des briques logicielles relatives à, par exemple, la gestion de la persistance, la confidentialité des données ou encore la gestion des capteurs et actionneurs (mise à jour du firmware, configuration, etc.). Compte tenu du fait qu'il s'agisse de problématiques éloignées de notre travail, nous ne les aborderons pas.

4.1.4 Architecture de déploiement typique pour la ville intelligente

4.1.4.1 Déploiement dans des villes existantes

Nous avons identifié que les plateformes qui sont déployées dans les villes existantes, à savoir Santander [147] et Barcelone [158] en Espagne, Barcelone et à proximité de l'université de la ville de Padoue [173] en Italie, reposaient sur une même architecture de déploiement composée de trois étages.

Le premier étage est constitué de capteurs et d'actionneurs déployés dans la ville. Comme nous le décrivions en section 3.2.4, ces expérimentations visent principalement à superviser la ville. Au sein de ce premier étage sont également associés les répéteurs qui permettent d'assurer la liaison avec le matériel présent dans le deuxième étage.

Le deuxième étage comprend des passerelles collectant les données des différents capteurs dans les divers lieux d'expérimentations. Ces passerelles couvrent des zones géographiques limitées de l'ordre de grandeur de quelques rues. Elles stockent les données de capteurs et les communiquent au troisième étage par le réseau internet

ou par le réseau GPRS en cas de défaillance.

Ce troisième étage est constitué d'un ensemble de serveurs hébergés dans une infrastructure de cloud computing. Les différents composants logiciels y sont déployés tels que ceux relatifs à la sécurité (authentification et autorisation des applications), à la gestion des expérimentations (configuration, réservation, analyse, etc.) ou encore à la persistance des données (des capteurs, des expérimentations, etc.). Une interface est fournie aux applications par laquelle elles peuvent superviser la ville.

4.1.4.2 Correspondance avec nos objectifs de contrôle

L'architecture de déploiement mise en place dans les diverses expérimentations, qui est composée de trois étages, est en phase avec les objectifs de contrôle que nous visons. En effet, le deuxième étage, qui comprend des passerelles servant uniquement de rôle d'intermédiaire aux serveurs, constitue un choix de placement idéal pour la couche d'abstraction que nous proposons, comme nous le verrons en section 5.2.2. Comme nous le décrivions précédemment, ces passerelles peuvent être utilisées pour former l'infrastructure edge et permettre aux applications de contrôle de s'interfacer avec elles.

Les délais de communications entre les applications et les actionneurs sont alors plus faibles (quelques dizaines de millisecondes à quelques centaines de millisecondes), si le réseau utilisé le permet, puisque les communications ne transitent plus par des serveurs hébergés dans une infrastructure cloud.

Similairement, ces passerelles formant l'infrastructure edge pourraient être utilisées pour héberger des applications de contrôle temps-réel. Nous donnons plus de détails à ce propos en section 7.4. Par ailleurs, les applications qui n'ont pas de telles contraintes temps-réel peuvent s'interfacer avec les serveurs déployés au sein du troisième étage. Ainsi, l'architecture de déploiement actuelle des villes intelligentes que nous avons identifiées est un choix idéal.

Enfin, il est à noter qu'au sein des expérimentations qui ont été conduites, les matériels sont contraints en ressources. Par exemple, dans la ville de Santander, les passerelles fournies par le fabricant Libellium ont des ressources très limitées [101] (mono-coeur, 500Mhz de fréquence processeur, 256MB de RAM).

Cependant, nous pouvons supposer que, compte-tenu de l'évolution des performances du matériel ces dernières années, ces caractéristiques pourraient être grandement améliorées. Ainsi, les passerelles pourraient jouer un rôle qui n'est plus uniquement limité à servir d'intermédiaire pour les serveurs mais qui pourrait être de constituer l'infrastructure edge.

4.2 Élaboration d'un exemple typique et de scénarios d'utilisation

4.2.1 Éléments présents dans l'exemple

La figure 4.3 illustre un exemple typique du quartier d'une ville. Comme nous le décrivons dans la sous-section 4.2.1.1, les éléments présents dans cette figure appartiennent à différents acteurs de la ville, les capteurs et actionneurs sont quant à eux installés par différents équipementiers. Nous décrivons en section 4.2.1.2 les différents réseaux qu'utilisent les capteurs et actionneurs pour communiquer avec notre plateforme fédératrice.

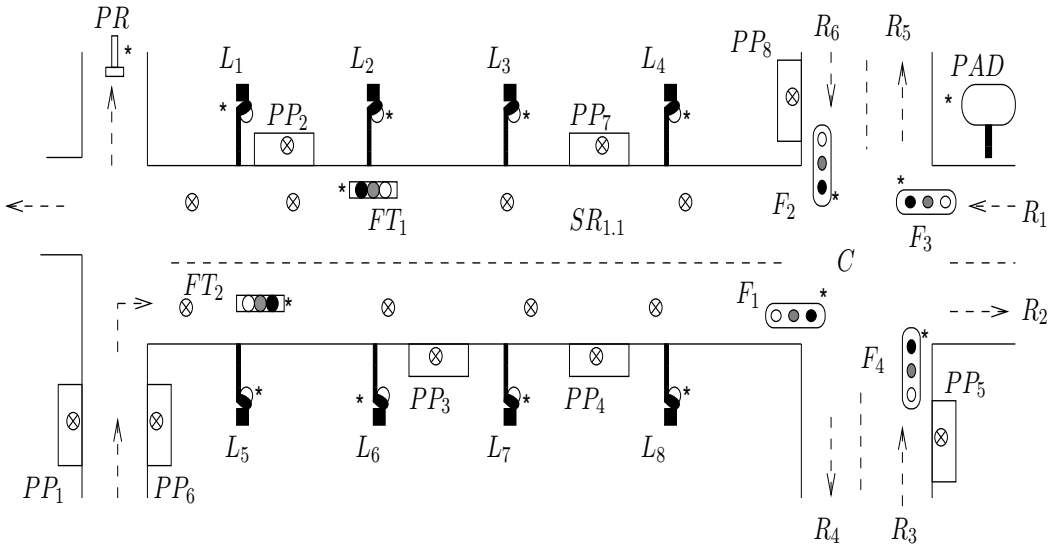


FIGURE 4.3 – Exemple d'un quartier d'une ville. Un rond barré désigne un capteur de présence, un carré plein représente un capteur de luminosité et une étoile indique la présence d'un actionneur. L_1 à L_8 désignent huit lampadaires, PP_1 à PP_8 huit places de parkings. R_1 et R_2 représentent deux routes sur lesquels nous nous focaliserons et $SR_{1.1}$ désigne un segment de la route R_1. C désigne un carrefour composé de quatre feux tricolores notés F_1 à F_4. PR désigne à un plot rétractable, PAD un panneau d'affichage digital, et enfin FT_1 et FT_2 désignent deux feux de travaux.

4.2.1.1 Acteurs présents dans la ville

Les différents éléments physiques présents sur cette figure, c'est-à-dire ceux qui sont étiquetés par un label, appartiennent à différents acteurs de la ville

Les lampadaires (L_1 à L_8) appartiennent à l'opérateur de l'éclairage public. Les quatre feux du carrefour C (ici, F_1 à F_4) sont détenus par la municipalité, de même pour les différentes routes (ici, R_1 à R_9). La municipalité possède également des plots rétractables tel que le plot PR, utilisé ici pour restreindre l'accès à une zone, par

exemple dans le cas de livraison de marchandises. Une borne RFID est également présente à proximité du plot rétractable PR afin de contrôler l'accès à cette zone.

Le panneau d'affichage digital appartient à l'entreprise en charge des transports publics de la ville. Il est utilisé pour informer les citoyens des différentes offres promotionnelles ou des perturbations (manifestations, pannes, etc.) impactant leur réseau. Un opérateur est en charge de la gestion des places de parking, dont les places PP_1 à PP_8. Enfin, l'acteur de la ville chargé d'effectuer des travaux sur la voirie possède les feux de travaux FT_1 et FT_2, qui sont utilisés afin d'alterner la circulation en cas de travaux en cours.

Des équipementiers ont également été en charge d'installer les différents capteurs et actionneurs présents sur cette figure. Pour superviser le taux d'occupation des routes R_1 et R_2, des capteurs de présence ont été installés sous la chaussée sur différents segments. Il s'agit de capteurs magnétiques qui sont placés aux extrémités de chaque segment de route, tels que ceux placés sur les extrémités du segment $SR_{1.1}$. Ces capteurs magnétiques émettent un signal lorsqu'un véhicule passe dessus, cette détection étant réalisée grâce au changement de champ magnétique.

Des capteurs de présence sont également installés par un second équipementier, spécialisé dans ce type de matériel, au niveau des places de parking. Ici, il s'agit de capteurs magnétiques qui permettent de détecter la présence d'un véhicule, ceux-ci étant placés sous la chaussée. Ces capteurs permettent de savoir si les différentes places de parking sont occupées ou libres. Les lampadaires L_1 à L_8 intègrent des capteurs de luminosité. Il s'agit ici d'un équipementier ayant installé ces différents lampadaires intelligents. Chaque lampadaire dispose de trois modes de fonctionnement : allumé, éteint ou mis en veille. Enfin, le plot rétractable PR ainsi que la borne RFID ont également été installés par un équipementier spécialiste de ce domaine.

4.2.1.2 Communication avec la plateforme fédératrice

Réseaux de capteurs et d'actionneurs Différents types de réseaux sont utilisés pour assurer la communication entre les capteurs/actionneurs et la plateforme fédératrice. Le contrôle du panneau digital PAD s'effectue via l'utilisation du réseau Wifi. Ce dernier possède un point d'accès Wifi par lequel la transmission des données s'effectuent afin d'afficher différents messages. Les feux de travaux FT_1 et FT_2 peuvent être contrôlés de manière sans-fil via la technologie réseau Bluetooth Low-Energy (BLE) puisque ceux-ci intègrent des récepteurs radio équipés de cette technologie. Les lampadaires L_1 à L_8 ont chacun un émetteur et un récepteur radio Zigbee, à la fois pour émettre des données relatives à la luminosité mesurée et pour recevoir des ordres afin d'être allumés, éteints ou mis en veille.

D'autre actionneurs utilisent le réseau électrique par lequel ils sont alimentés pour recevoir des commandes, c'est le cas du plot rétractable PR. La borne RFID à proximité utilise, comme son nom l'indique, la technologie radio RFID pour vérifier l'identité d'un véhicule. Les feux F_1 à F_4 faisant partie du carrefour C, sont également actionnables via le réseau électrique. Les commandes sont envoyées aux contrôleurs intégrés sur chaque feu du carrefour.

Enfin, les capteurs ultrasons présents sous les places de parking communiquent via la technologie Long Power Wide Area (LPWA) qui est LoRa, de manière événementielle lorsqu'une voiture est détectée. Similairement, les capteurs magnétiques communiquent également de manière événementielle mais utilisent la technologie de communication enOcean qui est de moindre portée que LoRa.

Matériels edge Les différents réseaux, par lesquels transitent les données des capteurs et commandes envoyées aux actionneurs, sont utilisés par une passerelle qui fait partie de l'infrastructure edge et qui gère cette zone géographique. Comme nous l'avons identifié en section 4.1.4, ce découpage correspond aux déploiements existants. Ainsi, le matériel, faisant partie de l'infrastructure du edge, s'interconnecte avec les différents réseaux des capteurs et actionneurs. Le matériel est responsable des données collectées et des commandes envoyées aux différents actionneurs.

En revanche, les capteurs ultrasons qui utilisent une technologie longue portée n'envoient pas leurs données à cette passerelle, mais à une passerelle dédiée à la collecte et à l'envoi d'ordres aux capteurs et actionneurs utilisant cette technologie de communication. Puisque les données des capteurs placés sous les places de parking PP_1 à PP_8 sont collectées par une plateforme qui ne nous appartient pas (ici une plateforme de relais sur laquelle nous nous reposons), seuls les serveurs hébergés au sein d'une infrastructure cloud obtiennent ces données.

4.2.2 Scénarios d'utilisation par des applications

4.2.2.1 Partage des données de capteurs

Plusieurs applications peuvent réutiliser des données de capteurs pour leurs propres besoins. Dans la figure 4.3, les données des capteurs sous les places de parking PP_1 à PP_8 sont utilisées par l'acteur qui gère les places de parking dans la ville. Ce dernier a conçu une application permettant d'aiguiller les automobilistes quant à la disponibilité des places de parking à proximité de leur position géographique.

Les données de ces capteurs peuvent être réutilisées par une application qui éclaire de manière efficiente les routes dont R_1 et R_2. Celle-ci tient compte de la luminosité ambiante, captée par les capteurs des lampadaires L_1 à L_8, du taux d'occupation des routes, mesuré par les capteurs enfouis dans les routes R_1 et R_2, et de la présence de véhicules stationnés sur les places de parking. En fonction de ces métriques, l'application allume ou éteint certains lampadaires, par exemple en éteignant le lampadaire qui éclaire inutilement une place occupée.

Il est à noter que cette application n'a pas de contraintes fortes par rapport aux délais de communications des ordres envoyés. Il s'agit ici d'une optimisation de l'éclairage public où il est jugé acceptable que les lampadaires reçoivent leurs commandes après plusieurs centaines de millisecondes voire quelques secondes.

Ce scénario d'utilisation montre qu'une plateforme fédératrice permet la réutilisation de données de capteurs par des applications qui ont des objectifs différents. Au lieu d'interroger la plateforme pour acquérir des données de capteurs, certaines

applications peuvent souhaiter recevoir des notifications, telle que l'application qui gère la disponibilité des places de parking. Ainsi, une plateforme fédératrice doit inclure un système de notification pour répondre aux exigences de ce type d'applications.

4.2.2.2 Accès partagé aux actionneurs

Des actionneurs, tel que celui intégré au panneau d'affichage digital PAD peuvent être partagés par plusieurs applications. Par exemple, une application de gestion du trafic peut s'en servir pour informer les automobilistes de l'état du trafic alors qu'une application gérant les places de parking peut contrôler le panneau afin d'informer les conducteurs du nombre de places disponibles à proximité. Ces deux applications de contrôle n'ont ici pas de contrainte de contrôle temps-réel.

De même, le plot rétractable PR est utilisé par une application pour gérer l'accès à une zone via la borne RFID, lors de livraison de marchandises par exemple. Mais une application de gestion du trafic peut en faire un usage opportuniste : lorsqu'il y a des bouchons, les conducteurs peuvent être amenés à emprunter cette zone d'accès en guise de déviation, ce qui implique de contrôler le plot sans qu'il y ait de contrôle d'accès effectué par la borne. Contrairement aux deux applications précédentes, celles-ci ont des contraintes de contrôle temps-réel et doivent voir leurs ordres être effectués au plus sous quelques centaines de millisecondes.

Ainsi, nous avons identifié qu'une plateforme fédératrice doit tenir compte des accès concurrents effectués par des applications de contrôle. De ce fait, une telle plateforme doit posséder un mécanisme de concurrence permettant de gérer ces cas là. Nous décrivons en section 6.2 notre proposition à ce sujet.

4.2.2.3 Accès à plusieurs actionneurs espacés géographiquement

Nous prenons le cas d'utilisation d'une application de régulation du trafic routier qui contrôle les quatre feux tricolores (F_1, F_2, F_3 et F_4) du carrefour C ainsi que les deux feux de travaux FT_1 et FT_2 en fonction du taux d'occupation des différentes routes. Dans ce cas d'usage, seule la partie dédiée au contrôle nous intéresse.

Cette application contrôle les feux du carrefour C de manière simultanée afin d'autoriser ou non le passage des voitures d'Est en Ouest et du Nord au Sud (vice-versa). L'application réutilise les feux de travaux, appartenant à l'opérateur effectuant des travaux sur la voirie, et les coordonne avec les feux du carrefour C. Contrairement aux feux du carrefour C qui sont reliés physiquement ensemble, les feux de travaux ne sont pas reliés physiquement aux feux du carrefour compte-tenu de l'espace géographique les séparant. Cette application a des contraintes temps-réel quant aux ordres qui sont envoyés aux différents feux présents, pour que la circulation s'effectue de la manière la plus fluide qui soit.

Nous avons identifié qu'une plateforme fédératrice doit intégrer un mécanisme de coordination permettant aux applications d'utiliser conjointement des actionneurs espacés géographiquement. Comme évoqué dans notre scénario d'utilisation, cela

favorise la réutilisation de ceux-ci : coordonner des actionneurs permet à l'application de régulation de trafic de réutiliser les feux de travaux dans un autre but que celui prévu initialement, c'est-à-dire l'alternance de la circulation. Nous présentons notre proposition à ce sujet en section 6.4.

4.2.3 Scénarios d'arrivée de nouveaux opérateurs

4.2.3.1 Intégration de nouveaux capteurs et actionneurs

Il peut apparaître dans la ville intelligente que de nouveaux opérateurs déploient leurs propres capteurs et actionneurs après signature d'un accord avec la municipalité. Plusieurs cas d'usages peuvent apparaître : le déploiement de caméras pour mesurer le taux d'occupation sur certains axes qui ne sont pas encore pourvus de capteurs, le déploiement de capteurs pour mesurer le taux de remplissage dans des conteneurs ou encore le déploiement de feux travaux par l'opérateur qui effectue sporadiquement des travaux sur la voie publique.

Pour favoriser l'ouverture de la plateforme fédératrice à de nouveaux opérateurs, celle-ci doit faciliter l'intégration des capteurs et actionneurs fournies par les nouveaux opérateurs. Afin de parvenir à ce but, la couche d'abstraction que nous proposons dans le chapitre 5 n'est pas dédiée à des capteurs ou actionneurs spécifiques. De ce fait, puisque notre plateforme n'est pas restreinte à certains capteurs ou actionneurs, l'intégration de nouveaux équipementiers en est facilité.

4.2.3.2 Développement de nouvelles applications

L'arrivée de nouveaux opérateurs peut amener au développement de nouvelles applications en tirant partie du nombre de capteurs et actionneurs déjà existants dans la ville et en réutilisant ceux-ci. Par exemple, l'opérateur déployant ses capteurs au sein de conteneurs, afin de vérifier leurs taux de remplissage, peut développer une application gèrant la collecte des déchets de ses conteneurs en tenant compte du taux d'occupation présent sur les routes proches des conteneurs. Comme évoqué en sous-section 4.1.3.2, les autorisations que requièrent ces nouvelles applications ainsi que leur authentification et leurs autorisations pour accéder aux données des capteurs et des actionneurs sont hors du périmètre de cette thèse.

Le but ici est de faciliter le développement de nouvelles applications, permettant de favoriser l'ouverture de la plateforme au plus grand nombre d'acteurs via la réutilisation des capteurs et actionneurs existants dans la ville.

Pour parvenir à ce but nous proposons d'adhérer aux principes architecturaux REST, puisqu'il s'agit d'un standard de fait dans le domaine de l'internet des objets pour la conception d'APIs qui facilite le développement d'applications. En respectant les principes REST, nous avons choisi d'intégrer notre travail dans une architecture orientée ressources qui est en adéquation avec notre objectif de modélisation (cf. section 4.4). Toujours dans le même but, nous proposons une couche d'abstraction composée d'un ensemble de modèles des éléments physiques (cf. chapitre 5.1) qui permettent aux applications de plus aisément contrôler et superviser la ville.

4.3 Qualité de service et exigences à satisfaire pour le contrôle

La plateforme dans laquelle notre travail s'insère vise à être hébergée dans une infrastructure incluant du edge computing, permettant ainsi à des applications de contrôle ayant des contraintes temps-réel de s'exécuter, qui est l'un de nos objectifs. Pour parvenir à celui-ci, nous avons identifié que la plateforme doit satisfaire des exigences non-fonctionnelles qui concernent à la fois les latences des diverses communications réseaux (cf. section 4.3.1) et nos propositions (cf. section 4.3.2). De même, notre travail doit également satisfaire des exigences fonctionnelles (cf. section 4.3.2).

4.3.1 Latence des communications réseaux

4.3.1.1 Latence maîtrisée des réseaux de capteurs et d'actionneurs

Pour permettre aux applications d'effectuer des opérations de contrôle via l'usage d'actionneurs, une exigence non-fonctionnelle propre à la performance des communications réseaux entre la plateforme et les actionneurs doit être respectée. Cette exigence est relative au fait que les latences de communications avec les actionneurs doivent être compatibles avec les contraintes temps-réel des applications de contrôle. Plus généralement, les délais de communication doivent être déterministes, sans quoi les applications de contrôle temps-réel ne pourraient avoir de garantie concernant leurs opérations.

Pour satisfaire cette exigence, un déploiement adéquat des capteurs et des actionneurs mais également du matériel formant l'infrastructure edge doit être effectué. Ce déploiement, basé sur l'identification que nous en avons fait (cf. sous-section 4.1.4.1), doit limiter le matériel edge à gérer des zones géographiques où sont présents des capteurs et actionneurs. Ces zones géographiques ne doivent pas être trop conséquentes pour limiter le nombre de répétiteurs et de fait diminuer les délais de communication. De même, le choix des réseaux d'accès aux capteurs et aux actionneurs joue un rôle prépondérant. Par exemple, il ne doit pas se produire d'interférences lors des communications avec les actionneurs puisque cela impacterait les latences de communication.

Le respect de cette exigence non-fonctionnelle est importante pour notre travail mais est hors du périmètre de celui-ci. Nous partons ainsi du principe que cette exigence est satisfaite et que les délais de communication entre le matériel edge et les capteurs et actionneurs sont déterministes et faibles, c'est-à-dire étant au plus de quelques dizaines de millisecondes. Ainsi, nous considérons que les latences réseaux sont compatibles avec les exigences des applications de contrôle temps-réel.

4.3.1.2 Latence modulable des communications avec les applications

Les applications qui envoient des requêtes à la plateforme fédératrice pour contrôler la ville peuvent avoir des latences de communication variables suivant l'infrastructure au sein de laquelle elles sont hébergées. Une application peut être hébergée dans

une infrastructure distante, qui n'est pas du ressort de notre plateforme, entraînant ainsi des délais de communication très variables allant de quelques millisecondes à quelques secondes. Ces latences peuvent être acceptables par des applications de contrôle simple, comme certains cas d'éclairage public (cf. sous-section 4.2.2.1), ou encore comme certains types d'applications de supervision. En revanche, elles ne le sont pas pour des applications ayant des contraintes de contrôle temps-réel qui ne peuvent être soumises à ces ordres de grandeur, comme c'est le cas pour l'application de gestion du trafic routier (cf. sous-section 4.2.2.3)

Concernant les applications de contrôle temps réel, notre plateforme doit ainsi tenir compte d'une exigence non-fonctionnelle relative à la performance des communications des applications de contrôle temps réel vers notre plateforme. Cette exigence est relative au déploiement judicieux de ce type d'applications.

Ici, nous proposons pour les applications ayant des contraintes temps-réel de limiter les communications réseaux vers notre plateforme en les hébergeant localement au sein de la plateforme edge qu'elles utilisent. De ce fait, cela permet une performance accrue des communications vers la plateforme. Nous verrons plus en détails dans la section 7.4, cet aspect relatif au déploiement des applications.

4.3.2 Exigences relatives aux solutions proposées

4.3.2.1 Prise en compte dans la conception des mécanismes liés au contrôle

Une plateforme fédératrice doit prendre en compte les besoins des applications (cf. section 4.2.2 et 4.2.3) et fournir des mécanismes de contrôle de concurrence et de coordination. Compte-tenu du fait que les applications de contrôle peuvent avoir des contraintes temps-réel, nous avons identifié que cela impactait la conception de ces deux mécanismes.

En effet, lorsqu'une application de contrôle temps-réel envoie une requête de contrôle, un mécanisme de contrôle doit satisfaire ou non immédiatement cette requête afin de ne pas engendrer un délai supplémentaire où la requête n'aurait plus de sens d'être effectuée. Par exemple, un mécanisme de contrôle ne doit pas mettre en attente une requête de contrôle puisque le moment où celle-ci s'exécutera n'aura potentiellement plus de sens, dans le cas où il s'agit d'une application de contrôle temps-réel.

Nous avons pris en considération cette exigence dans la conception du mécanisme de contrôle de concurrence (cf. section 6.2) puisque les applications sont informées dès lors que leurs opérations de contrôle sont refusées. Cette même exigence est également prise en compte, pour les mêmes raisons, dans la conception du mécanisme de coordination (cf. section 6.4) puisqu'il n'y a pas de retransmissions durant les divers phases de ce mécanisme (cf. section 6.4.2).

4.3.2.2 Prise en compte dans les modèles

Les applications de contrôle temps-réel peuvent être amenées à contrôler des éléments de la ville qui ont des caractéristiques physiques et/ou fonctionnelles. Ces caractéristiques impliquent des délais qui sont propres aux éléments contrôlés.

Par exemple, le plot rétractable de la figure 4.3 (cf. section 4.2.1) peut être contrôlé par des applications afin d'être baissé ou levé. Cependant il ne s'abaisse pas ni ne se lève instantanément compte-tenu de ses caractéristiques physiques liées à la vitesse de ses moteurs. De même, un feu ne peut passer du vert au rouge instantanément puisqu'il possède une caractéristique fonctionnelle impliquant d'être transitoirement à l'orange.

Ainsi, les applications de contrôle temps réel doivent être informées que leurs opérations de contrôle ne peuvent prendre effet immédiatement dès lors que les éléments contrôlés ont des caractéristiques physiques et/ou fonctionnelles. Ces délais doivent être exposés aux applications afin de leur permettre de les laisser juge des actions à entreprendre.

Nous verrons en section 5.3 la manière dont nous reflétons ces caractéristiques dans les modèles que nous proposons, et en section 7.3 l'incidence que cela a sur le modèle de programmation des applications.

4.3.2.3 Prise en compte dans le déploiement du code

La couche d'abstraction que nous proposons doit permettre de faciliter le contrôle et la supervision de la ville intelligente.

Compte-tenu de l'infrastructure dans laquelle notre travail s'insère, il ne ferait pas sens que cette couche d'abstraction soit hébergée au sein d'une infrastructure de cloud computing. Par ailleurs, ce choix serait en contradiction avec les objectifs que nous visons, c'est-à-dire favoriser l'émergence des applications de contrôle temps-réel dans la ville intelligente. Nous donnons plus de détails à ce propos dans la section 5.2.2.2.

4.4 Intégration dans une architecture orientée ressource

Comme nous l'avons évoqué en section 2.1, REST est largement adopté dans le domaine de l'internet des objets. De plus, la propriété d'interface qu'il comporte facilite le développement d'applications.

Dans le contexte d'une plateforme partagée par différentes applications, standardiser les interactions avec la plateforme, grâce à une interface uniforme, amène à favoriser l'ouverture de la plateforme au plus grand nombre d'acteurs dans la ville. En effet, il est plus facile pour les acteurs de la ville d'utiliser une interface qui est standardisé pour superviser et contrôler la ville intelligente

Il s'agit la première raison du choix de REST dans notre travail. Pour respecter les principes présents dans ce style d'architecture, nous devons ainsi exposer des ressources aux applications.

La seconde raison est liée à l'adéquation de la notion de ressources (au sens défini dans REST) avec les éléments physiques de la ville à propos desquels nous focalisons notre modélisation (cf. chapitre 5). Puisqu'une architecture orientée ressource [145] est basée sur l'utilisation de ressources comme modèles, nous intégrons notre travail dans ce type d'architecture. Sur la base de la figure 4.4, nous en détaillons les raisons en prenant les critères sur lesquels sont fondés la notion de ressources :

— Tout comme les ressources, les éléments physiques de la ville sont uniques. Par exemple, le carrefour C est identifié de manière unique à la fois du fait de sa configuration et de sa position géographique dans la ville.

— Tout comme les ressources, les éléments physiques ont des relations entre eux. Le lampadaire L_1 est lié à l'élément physique qu'est la route R_1 ainsi que la place de parking PP_1 compte-tenu du fait qu'il illumine celle-ci. Quelques exemples de relations sont illustrés dans la figure 4.4.

— La représentation des éléments physiques correspond à ce qu'ils sont dans la réalité.

— Les éléments physiques, tout comme les ressources, peuvent être utilisés uniformément par les applications. Les applications peuvent contrôler ceux-ci, en allumant ou éteignant le lampadaire L_1 par exemple, et/ou superviser les éléments physiques, tel que surveiller le taux d'occupation de la route R_1 par exemple.

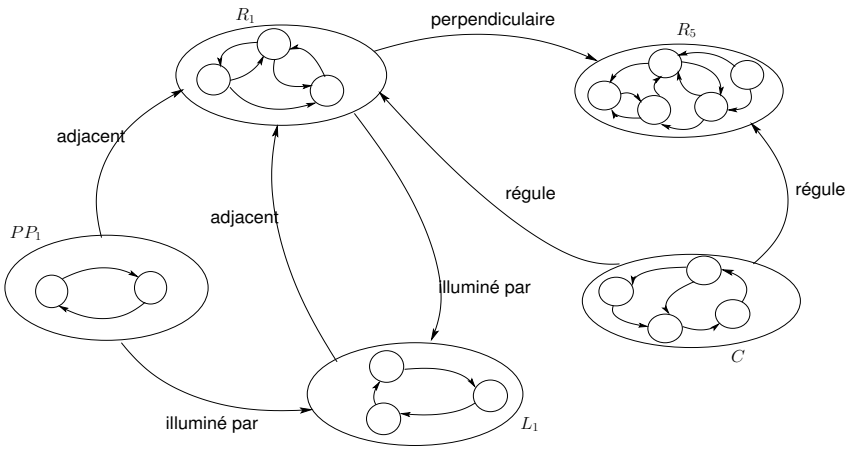


FIGURE 4.4 – Les ovales représentent des éléments physiques : le lampadaire L_1, les routes R_1, R_5, la place de parking PP_1, le carrefour C. Les ronds représentent les états possibles des éléments physiques. Les flèches indiquent des relations entre éléments physiques ou états.

La notion de ressources est également en adéquation, à un second niveau, avec les états des éléments physiques. En effet, tout élément physique possède un ou plusieurs états, par exemple la place de parking PP_1 possède deux états relatifs au fait qu'elle soit libre ou occupée. De même le lampadaire L_1 possède trois états liés

au fait qu'il soit allumé, éteint ou mis en veille. Nous verrons en section 5.2 qu'afin de modéliser la ville intelligente, nous avons choisi des modèles à états finis.

Ces états sont uniques dans le contexte d'un élément physique, un feu tricolore n'a, par exemple, qu'un seul état représentant le fait qu'il soit vert. Les états sont également liés entre eux, un feu tricolore est dans l'état rouge avant d'être dans l'état vert. Ils ont également une représentation. L'état occupé de la place PP_1 représente le fait qu'un véhicule occupe cette place. Enfin, les états peuvent être utilisés de manière uniforme par les applications : en supervisant les états ou en les contrôlant. Par l'état d'une route, qui représente un certain taux d'occupation, peut être observé par les applications. Similairement, l'état d'un lampadaire peut être contrôlé afin que celui-ci soit allumé, éteint ou mis en veille dans notre cas.

Par conséquent, une architecture orientée ressources est un choix idéal dans notre cas. A la fois car les éléments physiques de la ville, que nous visons à modéliser, sont en adéquation avec la notion de ressources présente dans une architecture orientée ressources, mais également car les états des éléments physiques, sur lesquels nous basons notre choix de modèles, sont également en adéquation avec cette architecture. Nous reviendrons sur cette concordance (cf. section 5.4) après avoir décrit les modèles que nous avons choisi dans le chapitre suivant.

Chapitre 5

Modèles et interfaces partagés pour la supervision et le contrôle

Dans ce chapitre, nous proposons des modèles et présentons leurs interfaces utilisées par les applications pour superviser et contrôler la ville. Ces modèles sont partagés compte-tenu du contexte de notre travail.

L'abstraction que nous proposons se focalise sur les éléments physiques de la ville (cf. section 5.1.1) puisqu'ils sont le point commun de toute instance de ville. Notre abstraction se base sur la notion d'entité et d'entité contrôlable dont nous présentons la définition en section 5.1.2 ainsi que les motivations (cf. section 5.1.3).

Afin de décrire les entités, nous utilisons des modèles discrets à états finis sous forme d'automates de Mealy décrits par des éléments de syntaxe du langage synchrone Argos (cf. section 2.2.1.3). Nous expliquons les raisons de ce choix en section 5.2.1 et présentons les interactions qui existent entre les entités, les applications et les éléments physiques représentés (cf. section 5.2.2). Ces interactions nous conduisent à évoquer le déploiement du code nécessaire à l'implémentation des modèles des entités.

Nous expliquons ce que comporte nos automates, en présentant les différents éléments utilisés pour satisfaire les requêtes des applications de contrôle (cf. section 5.3.1). Nous montrons que les automates proposés sont agnostiques du type de capteurs présents dans la ville (cf. section 5.3.2) pour de ne pas restreindre les entités à une instance de ville en particulier. Les automates représentent également à différents niveaux de granularité les éléments physiques (cf. section 5.3.3).

Nous présentons en section 5.4 les différents principes de définition des automates. En effet, les automates ont une structure particulière (cf. section 5.4.1) en termes d'états, d'entrées et de sorties que nous exploitons pour générer des ressources REST. De plus, lorsque l'utilisation d'états dans l'automate s'avère fastidieuse nous proposons l'usage de variables (cf. section 5.4.2).

Enfin, nous montrons qu'il existe une correspondance entre les entités et les ressources. D'une part, nous avons identifié qu'il existait des similitudes entre le critère HATEOAS et les modèles des entités (cf. section 5.5.1). D'autre part, nous générons automatiquement des ressources REST à partir de nos modèles qui sont présentés au travers de divers exemples (cf. section 5.5.2).

Nous montrons également que des annotations sémantiques peuvent être ajoutées à ces ressources (cf. section 5.5.3).

Sommaire

5.1 Abstraction des éléments physiques par des entités

5.1.1 Les éléments physiques comme point commun de la ville

Une ville se compose d'un ensemble d'*éléments physiques* que nous définissons comme étant des équipements : lampadaires, plots rétractables, feux tricolores, etc. ; ou des zones géographiques : segments de routes, places de parking, carrefours, etc.

Chaque instance de la ville possède ses propres spécificités (géographiques, institutionnelles, etc.), mais chacune d'entre elles est composée des mêmes éléments physiques. Par exemple, l'exemple conducteur (cf. section 4.2) est composé d'éléments physiques tels que les lampadaires L_1 à L_7, les segments de route $SR_{1.1}$, $SR_{1.2}$, les routes R_1 et R_2, etc. que toute instance de ville possède. Les éléments physiques sont ainsi le point commun à toutes les instances de ville.

Afin de rendre la ville intelligente, ces éléments physiques intègrent des capteurs et actionneurs. Un équipement tel que le plot rétractable PR intègre des actionneurs permettant de lever ou monter celui-ci. Les équipements peuvent également être pourvus de capteurs, tel que le capteur présent dans le plot rétractable PR pour connaître sa hauteur.

Les zones géographiques sont également pourvues de capteurs : le segment de route $SR_{1.1}$ possède des capteurs magnétiques à chaque extrémité, les places de parking PP_1, PP_2 possèdent des capteurs de présence sous le bitume. Les zones géographiques ne sont pas actionnables directement, mais les effets qu'engendrent

des actions sur les équipements à proximité de ces zones géographiques influencent les valeurs des capteurs présents sur ces zones : mettre un feu tricolore au rouge aura une influence sur le taux d'occupation de la route se situant à proximité.

5.1.2 Définition des entités

Nous proposons de représenter les éléments physiques de la ville intelligente par ce qu'on appelle des *entités*. Une *entité* est une abstraction d'un ou plusieurs éléments physiques de la ville comportant des capteurs et actionneurs. On parle d'*entité contrôlable* lorsqu'une entité est associée à un ou plusieurs éléments physiques qui sont pourvus d'au moins un actionneur.

Par exemple, une entité associée à un plot rétractable est considérée comme contrôlable puisque des actionneurs sont présents au sein du plot pour le faire monter ou descendre. A l'inverse, l'entité qui est associée à un segment de route n'est pas contrôlable puisque seuls des capteurs de passage sont présents sur celui-ci.

La figure 5.1 illustre le diagramme de cas d'utilisation des entités où les applications utilisent les entités à la fois pour pouvoir superviser mais également contrôler les éléments physiques de la ville intelligente. Le contrôle des éléments physiques s'effectue par l'intermédiaire de commandes envoyées aux actionneurs présents sur les équipements. Tels que montrés sur cette figure, les modèles des entités peuvent être modifiés en fonction des données des capteurs et des ordres envoyés aux actionneurs.

Nous verrons plus en détails la nature de ces interactions entre les entités et les différents acteurs en sous-section 5.2.2.

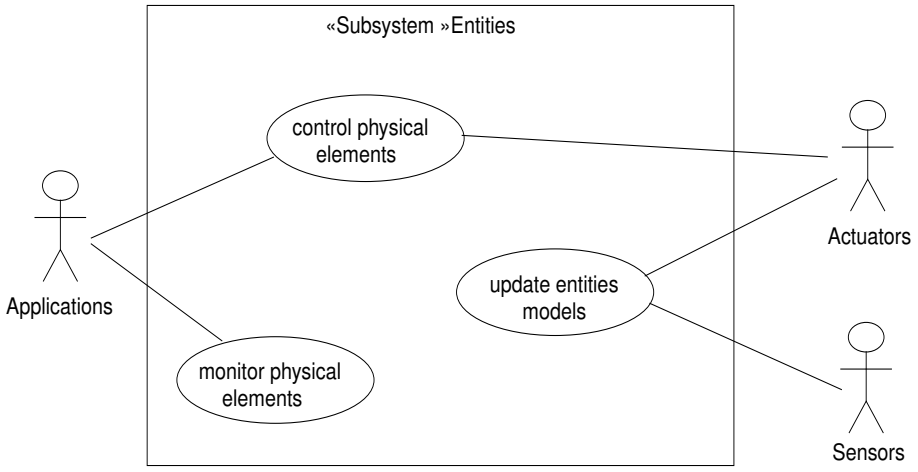


FIGURE 5.1 – Cas d'utilisation montrant les fonctions remplies par les entités

5.1.3 Motivations de l'abstraction proposée

5.1.3.1 Les entités comme généralisation de la ville

L'abstraction que nous proposons par le biais des entités a pour objectif d'être généralisable à toute instance de ville. Ainsi, les applications utilisent les entités

pour superviser ou contrôler la ville ce qui amène à ce que les applications ne soient plus conçues au cas par cas suivant les spécificités de chaque ville.

Une entité associée au segment de route $SR_{1.1}$ doit être généralisable à tout segment de route de la ville intelligente, et non d'une ville en particulier.

De plus, différents types de capteurs peuvent servir à mesurer la même information. Par exemple, le taux d'occupation des véhicules présents sur une route peut être mesuré par différents types de capteurs : capteurs magnétiques (comme c'est le cas pour le segment $SR_{1.1}$), boucles à induction, capteurs ultrasons ou encore caméras (Pan Tilt Zoom, par exemple). Le type de capteur utilisé est donc spécifique à une instance de ville et afin d'être généralisable, les entités doivent pouvoir s'abstraire de la nature des capteurs.

5.1.3.2 Les entités comme granularité des éléments physiques

Les entités doivent abstraire les éléments physiques qu'elles représentent à différents niveaux de granularité pour permettre aux applications qui les utilisent de superviser et contrôler plus facilement la ville.

Par exemple, une entité peut être associée à l'ensemble des lampadaires se trouvant le long d'une rue, et permettre aux applications de contrôler les lampadaires de cette rue de manière groupée. Le contrôle des lampadaires d'une rue peut consister à tous les allumer, en allumer un sur deux ou encore tous les mettre en veille. Comme évoqué en sous-section 5.1.3.1, cette entité peut se généraliser à n'importe quelle instance de ville puisqu'elle est indépendante de la configuration d'une rue et de son nombre de lampadaires.

Sur la base de notre exemple conducteur (cf. section 4.2.1), une application d'éclairage public doit pouvoir allumer ou éteindre l'ensemble des lampadaires d'une rue, les lampadaires L_1 à L_7. Ici, une entité e_i associée à chaque lampadaire L_i serait fastidieuse à utiliser par cette application d'où l'avantage d'une entité regroupant les lampadaires.

Similairement, une abstraction de plus forte granularité doit permettre aux applications de superviser plus facilement un ensemble d'éléments physiques. Par exemple, une entité pourrait être associée à un ensemble de places de parking en fournissant une abstraction de la disponibilité de celles-ci, à savoir si elles sont toutes occupées, toutes libres ou à moitié libres par exemple.

5.1.3.3 Les entités comme moyen de protéger les équipements

Les équipements ne doivent pas pouvoir être contrôlés de manière incohérente par les applications. Une utilisation inconsistante résulte d'un état inconsistant d'un ou plusieurs équipements, c'est-à-dire provoquant un fonctionnement anormal de ceux-ci. Par exemple, une utilisation inconsistante d'un feu rouge est de le faire passer au feu vert alors qu'il est à l'orange. Similairement, lorsqu'il s'agit de plusieurs équipements, ici l'ensemble des feux d'un carrefour, une utilisation inconsistante produisant un état inconsistant est de les mettre tous au feu vert en même temps.

Cette notion de consistance est largement présente dans les bases de données, où l'état d'un élément du système est dit consistant lorsque les contraintes de consistance qui lui sont associées sont satisfaites [47]. Par exemple, une transaction impliquant deux comptes bancaires est dite consistante si, à la fin de la transaction, la contrainte de consistance suivante est respectée : la somme débitée est égale à la somme créditée.

Ainsi, en nous inspirant de cette notion de consistance, les entités contrôlables doivent intégrer des contraintes de consistance. Ces contraintes de consistance doivent permettre d'assurer que l'état des équipements ne soient pas inconsistants résultant d'un fonctionnement anormal de ceux-ci.

5.2 Représentation et utilisation des entités

5.2.1 Choix de représentation des entités

Nous avons choisi de représenter les entités par des automates discrets à états finis, puisqu'il s'agit de modèles particulièrement utilisés dans les systèmes réactifs (cf. section 2.2.1).

Notre système est similaire à un système réactif : l'ensemble des entités constitue notre système interagissant avec l'environnement à savoir l'ensemble des éléments physiques formant l'environnement de la ville intelligente (cf. section 5.1.1). Notre système interagit avec l'environnement via des sorties qui sont des commandes envoyées aux actionneurs pour contrôler les éléments physiques. A l'inverse, notre système reçoit de l'environnement des entrées qui sont des données provenant des capteurs.

Par ailleurs, nous avons choisi d'utiliser des automates puisque ceux-ci permettent de rendre notre modélisation générique : peu importe les éléments physiques associés aux entités, le modèle d'une entité est toujours un automate composé d'entrées, de sorties, de variables, de transitions et d'états. L'utilisation du langage Argos nous permet l'utilisation de profiter d'un vocabulaire étendue pour la description de ces automates. Puisque le modèle est le même quelle que soit l'entité, cela rend plus aisé le développement d'applications. Comme nous le verrons en section 5.5, l'interface uniforme issue des principes REST permet également de satisfaire ce critère.

Enfin, le choix de représenter les entités par des automates est motivé par le fait que les automates comportent des transitions qui peuvent être restreintes afin de protéger les équipements contre des utilisations incorrectes. Les automates comportent également des états qui peuvent être de granularité plus ou moins fine. Nous reviendrons plus en détails sur ces deux points dans la section 5.4.

5.2.2 Interactions entre applications, entités et éléments physiques

5.2.2.1 Vue d'ensemble

Nous définissons un exemple qui est généralisable, afin d'illustrer nos propos concernant les interactions existantes entre applications, entités et éléments. Celui-

ci est composé des deux applications : App_1 supervise le segment de route $SR_{1.1}$ via l'entité E_1 qui lui est associée, App_2 contrôle le feu de travaux FT_2 au travers de l'entité qui le représente. La figure 5.2 et la figure 5.3 sont des diagrammes de séquence qui décrivent exhaustivement ces interactions.

Comme montré dans les deux figures, les applications interagissent avec les entités au moyen d'états qui leur sont exposés pour superviser et contrôler la ville. Les applications peuvent observer mais également modifier l'état courant des entités.

Nous avons fait le choix d'exposer les états des modèles d'entités aux applications que nous appelons plus simplement les états des entités. La notion d'état est facile à appréhender pour des néophytes qui ne possèdent pas de connaissances techniques liées aux automates discrets à états finis que nous utilisons. Ainsi, il est rendu plus facile aux développeurs d'applications d'utiliser les entités pour contrôler et superviser la ville. Comme nous le verrons en section 5.4.1, le choix d'exposer des états aux applications a une influence sur la structure des automates.

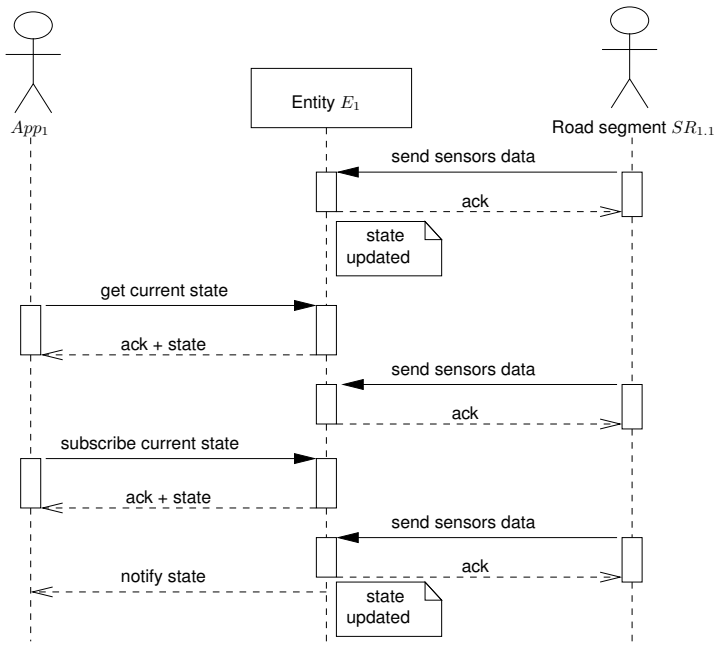


FIGURE 5.2 – Interaction entre l'application App_1 et le segment de route $SR_{1.1}$ via son entité E_1. App_1 lit l'état courant de E_1 qui change d'état en fonction des valeurs de capteurs reçues. App_1 souscrit également à une notification de changement d'état de E_1.

Supervision de la ville au travers des entités Les applications supervisent les éléments physiques de la ville via des requêtes envoyées aux entités pour connaître leur état courant. En effet, une entité qui reçoit une requête de supervision renvoie en retour son état courant, qui correspond à l'état courant de l'automate. Le choix de renvoyer l'état courant est pertinent puisqu'il correspond à l'état de l'entité à l'instant où la requête est reçue. La figure 5.2 montre que l'application App_1

supervise le segment de route $SR_{1.1}$ via une requête qui est envoyée à son entité, l'application recevra en retour la valeur de l'état courant.

Les états des entités sont issus de la consolidation des données de capteurs. L'entité du segment de route $SR_{1.1}$ reçoit les données des capteurs magnétiques (cf. section 4.2.1) consolidées sous forme de taux d'occupation. Par exemple, l'entité de $SR_{1.1}$ aurait des états (LOW TRAFFIC, MEDIUM TRAFFIC, HIGH TRAFFIC) correspondant aux différents taux d'occupation du segment de route. Nous verrons en sous-section 5.3.2 plus en détail l'entité du segment $SR_{1.1}$ et son modèle associé, qui nous permet de mettre en avant l'indépendance des entités vis-à-vis des types de capteurs présents dans la ville.

Enfin, afin de pouvoir superviser les éléments physiques dans un mode événementiel, les applications peuvent demander à être notifiées lors des changements d'états des entités.

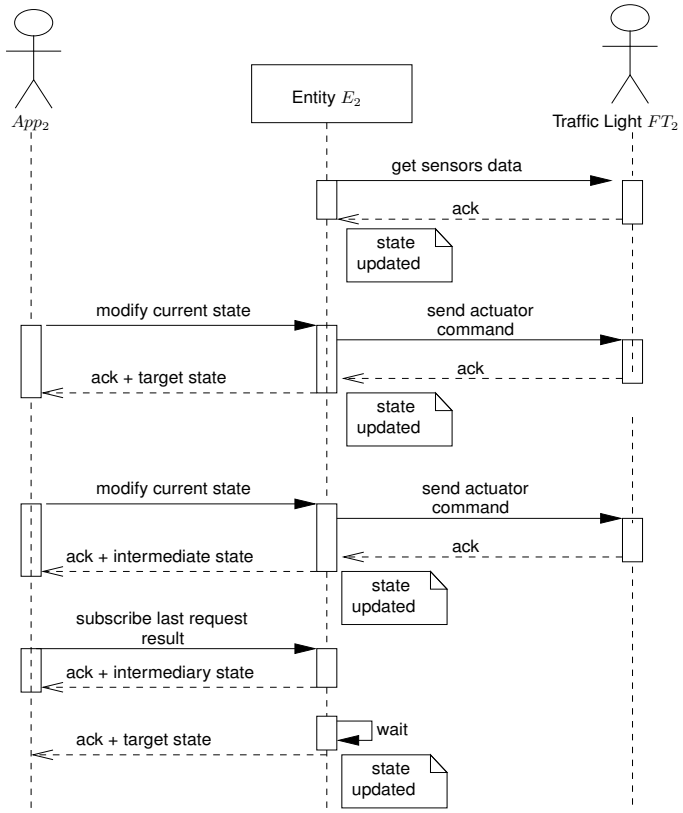


FIGURE 5.3 – Interaction entre l'application App_2 et le feu de travaux FT_2 via son entité E_2. App_2 modifie l'état courant de E_2. Lorsque l'état voulu par App_2 ne peut pas être atteint tout de suite, E_2 notifie App_2 qu'un état intermédiaire existe.

Contrôle de la ville au travers des entités Les applications peuvent également contrôler les entités en envoyant des requêtes pour changer l'état courant de celles-ci, comme illustré par le diagramme de séquence de la figure 5.3 où l'application App_2

change la couleur du feu de travaux FT_2 par l'intermédiaire de l'entité E_2 qui lui est associée.

L'application App_2 modifie l'état courant de l'entité, puisque les automates ont des entrées spécifiques qui le permettent (cf. section 5.3.1.1). Suite à la réception d'une requête de changement d'état, une entité contrôlable envoie des requêtes aux actionneurs pour atteindre l'état demandé puis reçoit en retour des acquittements, comme illustrés par les communications entre l'entité E_2 et le feu tricolore FT_2. Enfin, l'entité contrôlable retourne un acquittement à l'application pour lui notifier le résultat. Pour cela nos automates comportent des sorties qui sont des acquittements positifs ou négatifs (cf. section 5.3.1.3).

Comme le montre ce diagramme de séquence, avant d'atteindre l'état souhaité par l'application, l'entité peut atteindre un état dit intermédiaire dont nous donnons plus de détails en sous-section 5.3.1.2. Par exemple, si l'application App_2 demande à passer le feu FT_2 rouge, via l'entité E_2, celui-ci sera orange avant d'être rouge. Pour être informée que l'état voulu est atteint, l'application doit indiquer à l'entité qu'elle souhaite être notifiée. Une fois l'état cible atteint, après avoir été provisoirement dans l'état orange pour l'entité E_2, une notification est envoyée à l'application, ici informant App_2 que l'état du feu est rouge.

Enfin, les entités peuvent interroger les capteurs des équipements. Ici un capteur est présent dans le feu FT_2 permettant de savoir si une panne de courant est survenue. En l'interrogeant l'entité E_2 peut mettre à jour son état. Nous illustrons l'automate de cette entité dans la figure 5.4 qui est décrite en section 5.3.1.

5.2.2.2 Déploiement de code

Code relatif à l'implémentation des entités Le code qui décrit l'implémentation des entités sous forme d'automates doit être deployé au plus proche des éléments physiques qu'elles représentent. Cela permet de limiter la latence de communication, en communiquant directement avec les capteurs et actionneurs des éléments physiques, et ainsi de garantir que les états des entités changent en un temps compatible avec le contrôle temps réel.

Nous verrons en section 7.4 que le code décrivant l'implémentation des entités vise à être déployé sur les plateformes edges de la plateforme FIWARE qui correspondent au matériel le plus proche des éléments physiques. Cela constitue un choix idéal pour satisfaire nos exigences de temps-réel (cf. section 4.3) et permet d'assurer une meilleure tolérance aux pannes.

Code relatif aux applications utilisant des entités Les applications peuvent, suivant leurs besoins, avoir des contraintes temps-réel alors que d'autres non. Pour satisfaire les critères temps-réel de certaines applications, les communications réseaux doivent être réduites. Par conséquent les applications doivent s'exécuter au plus proche des entités qu'elles utilisent. Cela permet d'assurer également un fonctionnement autonome lorsque d'éventuelles pannes de matériels ou du réseau surviennent.

Nous verrons en détail en section 7.4 où doit s'exécuter le code des applications suivant leurs besoins respectifs.

5.3 Démarche d'invention des entités

5.3.1 Éléments des modèles d'entités contrôlables

Comme évoqué en sous-section 5.2.1, les automates représentant les entités contrôlables possèdent des entrées qui sont utilisées pour effectuer les changements d'états demandés par les applications, mais également des sorties représentant les acquittements envoyés aux applications. La figure 5.4 représente l'automate de l'entité associée au feu de travaux FT_1 et nous sert d'exemple pour illustrer ces notions d'entrées et de sorties des automates représentant les entités contrôlables. Par ailleurs, nous présentons également la notion d'état intermédiaire propre aux automates des entités contrôlables.

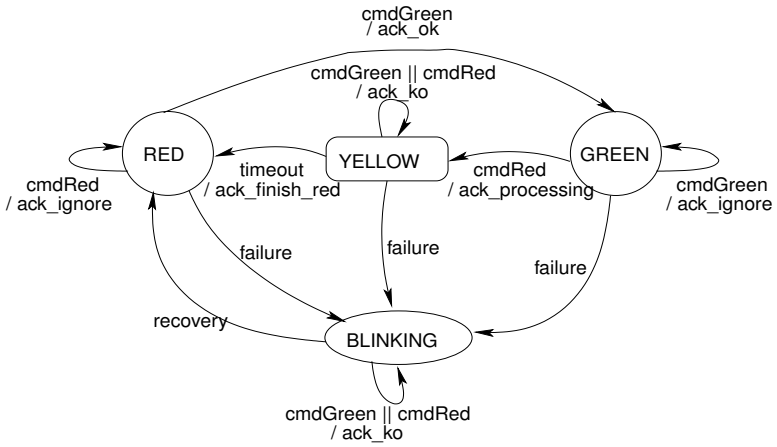


FIGURE 5.4 – Modèle d'entité du feu de travaux FT_1.

5.3.1.1 Entrées utilisées pour les requêtes de changement d'état

Les automates comportent des entrées permettant d'atteindre les états demandés par les applications de contrôle. Le modèle de l'entité du feu FT_1 illustré par la figure 5.4 comporte les entrées cmdGreen et cmdRed. Ici, l'entrée cmdGreen est utilisée pour atteindre l'état GREEN via une transition depuis l'état RED. De même, l'entrée cmdRed est utilisée pour atteindre l'état RED via une transition depuis l'état GREEN en passant par l'état temporisé YELLOW qui est un état intermédiaire sur lequel nous reviendrons.

Changement d'état effectué suite à une requête Lorsqu'un état demandé par une application peut être atteint, cela signifie qu'une transition existe dans

l'automate qui relie l'état courant à l'état souhaité. Cette transition comporte nécessairement une entrée destinée à effectuer les changements d'états souhaités par les applications (transitions comportant cmdGreen ou cmdRed dans notre exemple).

Comme nous l'avons illustré dans la figure 5.3, suite à une requête de changement d'état une requête est envoyée au(x) actionneur(s) de ou des équipement(s) représenté(s) par l'entité. Les ordres envoyés aux actionneurs par une entité contrôlable pour aller atteindre l'état souhaité par l'application sont transparents pour cette dernière.

Lorsque l'entité associée au feu FT_1 reçoit une requête d'une application pour aller dans l'état GREEN, alors que son état courant est RED, elle active la transition qui comporte l'entrée cmdGreen pour atteindre l'état souhaité. De manière sous-jacente, une commande est envoyée à l'actionneur du feu tricolore pour le faire passer au vert. Après réception de l'acquittement de la commande, l'entité notifie l'application via un acquittement ack_ok représenté en tant que sortie de la transition. Nous verrons en détail la nature des acquittements des automates en sous-section 5.3.1.3.

Un état cible n'est pas nécessairement atteint directement compte tenu des délais (cf. section 4.3.2.2) qui peuvent exister au sein de l'équipement, ce qui se traduit par la présence d'un état intermédiaire, comme nous le verrons en sous-section 5.3.1.2. Ici, l'automate de l'entité du feu FT_1 possède un état intermédiaire YELLOW qui précède l'état RED. Lorsque l'entité reçoit une requête pour atteindre l'état RED alors que l'état courant est GREEN, la transition comportant l'entrée cmdRed est activée et une communication a lieu entre l'entité et l'actionneur de l'équipement. Suite à quoi, l'état YELLOW est atteint provisoirement avant que l'état RED soit finalement atteint.

États impossibles à atteindre par les applications

Restriction sur les transitions Il se peut qu'un état cible d'une requête de changement d'état ne puisse pas être atteint depuis n'importe quel état.

Lorsque nous voulons empêcher qu'un équipement soit dans un état incohérent (cf. sous-section 5.1.3.3), en empêchant par exemple qu'un feu soit passé au vert alors qu'il est orange, nous appliquons des restrictions sur les transitions de l'automate de l'entité qui le représente. Dans notre exemple, l'état GREEN ne doit pas pouvoir être atteint depuis l'état YELLOW. Par conséquent, il n'y a pas de transition de l'état YELLOW vers l'état GREEN.

Comme nous le verrons dans la sous-section 5.3.1.3, un acquittement négatif est renvoyé aux applications qui ont demandé à atteindre un état qui n'est pas atteignable depuis l'état courant.

Entrées non utilisables par les applications Les automates des entités contrôlables peuvent comporter des entrées qui ne sont pas liées aux états demandés par les applications. De la même manière que pour les automates représentant des entités non-contrôlables, ceux-ci peuvent comporter des entrées correspondant à des

données de capteurs. Ces capteurs sont présents sur les équipements et leurs entrées dans les modèles et amènent l'état des entités à changer sans que l'application l'ait demandé.

La figure 5.4 illustre les entrées failure et recovery permettant respectivement d'atteindre l'état BLINKING depuis n'importe quel état et d'atteindre l'état RED depuis l'état BLINKING. Ces entrées sont relatives aux capteurs présents dans le feu tricolore : l'entrée failure correspond à un capteur qui détecte une panne de courant provoquant le passage du feu en clignotement via l'utilisation de la batterie intégrée au feu, l'entrée recovery correspond à la remise en fonctionnement normal du feu, par exemple suite à une intervention manuelle. Il est également à noter que l'état BLINKING est dit "fail-safe" puisqu'il s'agit d'un état représentant la gestion des panne dans l'équipement.

5.3.1.2 États intermédiaires

Les automates des entités contrôlables peuvent comporter des états dits intermédiaires. Les états intermédiaires représentent les délais des équipements induits par leurs contraintes physiques ou fonctionnelles. Ainsi, avant d'atteindre un état demandé par une application, un état intermédiaire peut exister.

Par exemple, avant d'être rouge, un feu tricolore est pendant un court instant orange, il s'agit ici d'une contrainte fonctionnelle du feu tricolore. Par conséquent, le modèle de l'entité du feu tricolore (cf. figure 5.4) comporte un état intermédiaire qui est l'état YELLOW et qui est atteint avant l'état RED. Ici, l'état YELLOW est temporisé, cependant un état intermédiaire n'est pas forcément un état temporisé.

Nous prenons le cas d'un autre équipement pour montrer qu'un état intermédiaire peut refléter également les contraintes physiques d'un équipement. La figure 5.5 illustre le modèle de l'entité associé au plot rétractable PR. Les états MOVING UP et MOVING DOWN sont des états intermédiaires et représentent le fait qu'un plot rétractable possède des contraintes physiques, liées à la vitesse de ses moteurs, qui font qu'il ne s'abaisse pas ou ne se lève pas instantanément. Ici, le capteur présent dans le plot rétractable permet de savoir si le plot est baissé ou levé, comme représenté par l'entrée sensorHeight.

5.3.1.3 Acquittements

Les automates comportent des sorties qui correspondent à des acquittements à destination des applications lorsque ces dernières envoient des requêtes de changements d'états. Il existe cinq types d'acquittements différents.

Acquittement de succès ack_ok L'acquittement ack_ok indique à l'application le succès de sa requête de changement d'état, ce qui signifie que les ordres envoyés aux actionneurs de l'équipement ont été acquittés avec succès et que l'état cible est atteint.

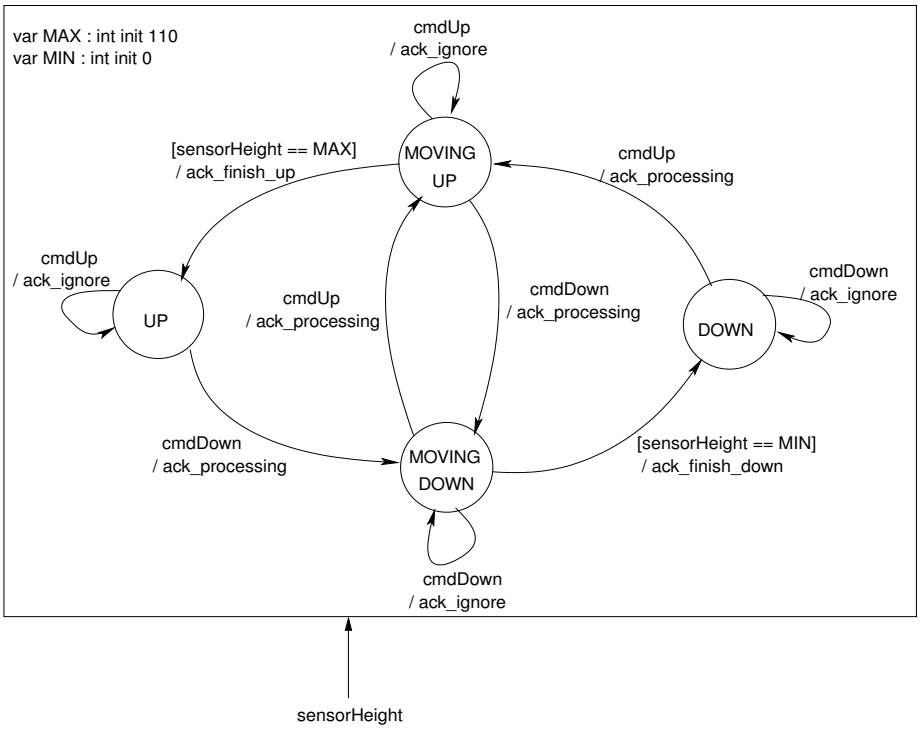


FIGURE 5.5 – Modèle de l'entité associé au plot rétractable PR

Dans la figure 5.4, qui illustre le modèle de l'entité représentant le feu de travaux FT_1, lorsque l'entité atteint l'état GREEN depuis l'état RED, elle envoie l'acquittement ack_ok indiquant à l'application qu'elle a atteint avec succès l'état demandé.

Acquittement ack_processing de traitement d'une requête de changement d'état L'acquittement ack_processing est retourné à une application ayant émis une requête de changement pour lui signifier que le changement d'état est en cours. Comme expliqué dans la sous-section précédente, une entité contrôlable peut comporter une contrainte physique que nous modélisons par un état intermédiaire. Ainsi, cet acquittement permet d'informer l'application que les requêtes aux actionneurs de l'équipement ont été acquittées avec succès mais que l'état cible n'est pas encore atteint.

Dans l'exemple du modèle de l'entité du feu FT_1, lorsqu'une application a envoyé une requête pour atteindre l'état RED, le feu passe d'abord à l'orange et un acquittement ack_processing est retourné à l'application pour lui signifier que le changement d'état est en cours mais pas encore fini.

Acquittement de fin de traitement ack_finish Afin d'indiquer à une application que l'état cible est bel et bien atteint, après s'être vu signifier qu'il était en cours, un acquittement ack_finish est retourné. Cet acquittement comporte en plus le nom de l'état cible qui est atteint afin que l'application n'ait pas à se rappeler de

l'état cible qu'elle avait initialement demandé.

Sur la base de la figure 5.4, après avoir atteint temporairement l'état YELLOW, l'état RED est atteint et un acquittement ack_finish_red est envoyé à l'application émettrice de la requête pour atteindre RED.

Acquittement ack_ignore pour ignorer une requête de changement d'état ack_ignore est l'acquittement qui indique à une application que sa requête est ignorée puisque l'état cible de cette requête est déjà l'état courant de l'entité.

En reprenant notre exemple, une application se verra renvoyer l'acquittement ack_ignore si elle envoie une demande à l'entité d'aller dans l'état GREEN alors que l'entité est déjà dans l'état GREEN.

Acquittement d'échec ack_ko L'acquittement ack_ko indique à une application l'échec de sa requête de changement d'état pour empêcher une utilisation incorrecte des équipements.

Dans notre exemple, lorsque l'entité associée à FT_1 est dans l'état YELLOW et reçoit une requête de la part d'une application pour atteindre l'état GREEN, un acquittement ack_ko lui indique que ce n'est pas possible et que sa requête est un échec.

Nous verrons en sous-section 5.4.1.3 que les acquittements que nous avons décrits sont générés automatiquement à partir de la structure des automates.

5.3.2 Indépendance vis-à-vis des capteurs

Afin d'être généralisables, les entités sont indépendantes des capteurs utilisés dans une instance de ville. Nous prenons l'exemple de l'entité représentant le segment de route $SR_{1.1}$ qui possède des états représentant les divers niveaux d'occupation du segment : LOW TRAFFIC, MEDIUM TRAFFIC et HIGH TRAFFIC. Ces états sont issus de la consolidation des capteurs magnétiques présents à chaque extrémité du segment, comme le montre la figure 5.6. En effet, un compteur qui représente le nombre de véhicules au sein du segment $SR_{1.1}$ est incrémenté à chaque passage d'un véhicule dans le segment et décrémenté lorsqu'un véhicule en sort. En fonction du nombre de véhicules, l'état actuel de l'automate est l'un des trois états : LOW TRAFFIC, MEDIUM TRAFFIC ou HIGH TRAFFIC.

Nous pouvons constater que l'état du modèle d'entité $SR_{1.1}$ pourrait osciller lorsqu'un véhicule sort du segment $SR_{1.1}$ et qu'un autre véhicule y entre.

Nous prenons le cas où la variable LOW a pour la valeur 15 (MEDIUM a peu d'importance dans cet exemple) et illustrons de phénomène d'oscillation dans la figure 5.7. Nous pouvons remarquer que lorsque le nombre de voitures fluctue entre 15 et 16, l'état oscille entre LOW TRAFFIC et MEDIUM TRAFFIC. Afin de palier à cette oscillation, nous appliquons une valeur Δ aux transitions de l'entité pour créer une hystérésis entre les seuils de changement d'état. L'intervalle de valeur étant plus grand pour passer d'un état à un autre, il n'y a plus de fluctuation de l'état. Dans le

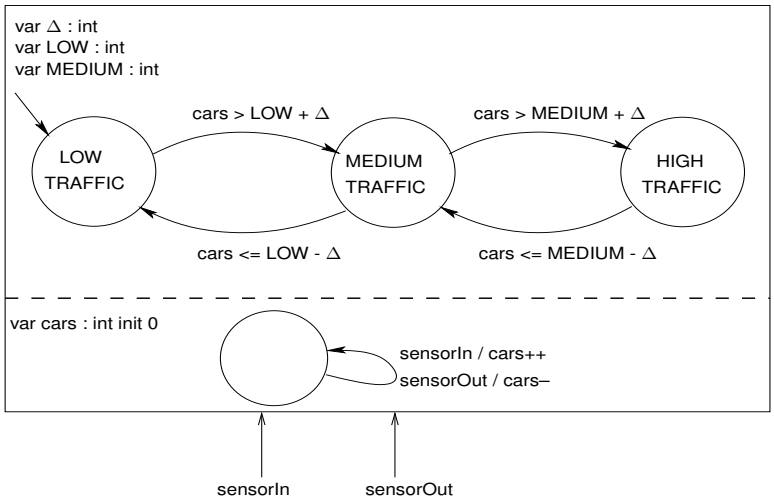


FIGURE 5.6 – Modèle d'entité du segment $SR_{1.1}$ comprenant des hystérésis (Δ présent sur les seuils).

cas de notre exemple, l'état MEDIUM TRAFFIC sera atteint lorsqu'il y a 17 voitures dans le segment de route, l'état LOW TRAFFIC lorsque qu'il y a 13 voitures.

Par ailleurs, l'application n'a pas besoin de connaître les détails relatifs aux capteurs utilisés pour la fabrication des états, ni comment est effectué le passage d'un état à un autre.

La figure 5.8 illustre l'entité générique du segment de route qui comprend seulement les différents états d'occupation. Comme illustré, ce qui est relatif à la manière dont sont comptés les véhicules (état compteur) et à comment passer d'un état à un autre (label des transitions) ne figure plus, car ceci est relatif à une instance de ville spécifique dont les applications n'ont pas à se préoccuper. En revanche, les transitions présentes permettent de garantir la cohérence de l'évolution du taux d'occupation : celui-ci devient forcément moyen avant de devenir élevé ou faible.

Ainsi, cela rend les applications indépendantes des types des capteurs présents dans la ville. Dans notre exemple, si d'autres capteurs que ceux électromagnétiques sont utilisés pour compter le nombre de véhicules (caméra, par exemple), cela sera transparent pour les applications qui utilisent des entités et non les capteurs mêmes pour obtenir le taux d'occupation

5.3.3 Abstraction de plusieurs éléments physiques

5.3.3.1 Granularité des états

Les états exposés aux applications peuvent être de différentes granularités en fonction de l'abstraction réalisée via les entités.

Nous prenons ici l'exemple des lampadaires L_1 à L_7 qui possèdent chacun trois états (ON, OFF et STAND-BY). Nous pourrions créer un modèles d'entités de n lampadaire à 3^n états qui est le résultat de la composition parallèle (cf. section 2.2.1.3)

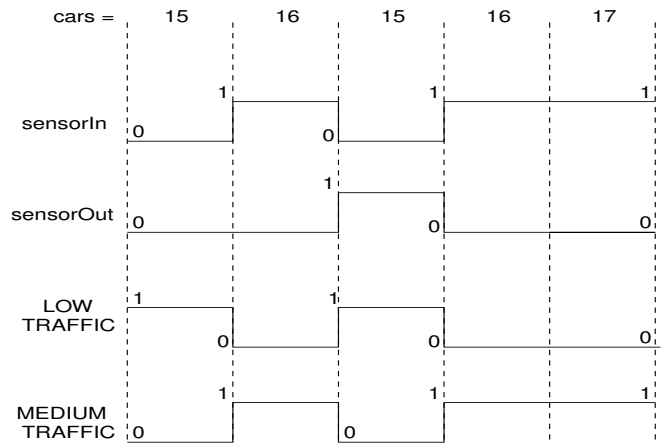


FIGURE 5.7 – Oscillation de l'état courant du modèle d'entité $SR_{1.1}$ lorsque le nombre de voitures présentes sur la route fluctue entre 15 et 16.

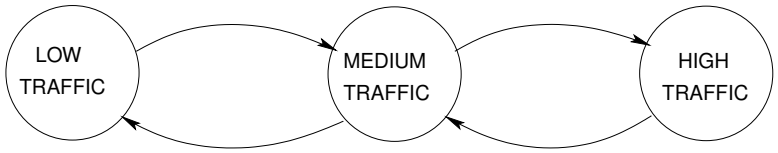


FIGURE 5.8 – Représentation de l'entité générique du segment de route

du modèle de chaque lampadaire. Cependant, si l'on veut observer et contrôler des lampadaires de manière globale, nous devons inventer une abstraction de ceux-ci. Ici, nous avons fait le choix de garder seulement les états ALL ON, ALL OFF et ALL STAND-BY. Il est également à noter qu'un modèle de groupe d'entités peut comporter des délais (états temporisés, par exemple) représentant l'envoi de commandes à de multiples actionneurs, tels qu'ici à plusieurs lampadaires.

Nous illustrons dans la figure 5.9 le modèle d'entité du groupe des lampadaires L_1 à L_7 que nous notons GL. Comme évoqué, les lampadaires peuvent être tous allumés (ALL ON), tous éteints (ALL OFF) ou tous en veille (ALL STAND-BY). Les états temporaires PROCESSING ALL STAND-BY et PROCESSING ALL ON représentent le délai physique pour mettre en veille ou allumer tous les lampadaires en même temps. L'état de cette entité peut être changé, ce qui aura pour effet d'envoyer des commandes aux actionneurs de chaque lampadaire afin d'atteindre l'état cible.

Cet exemple permet de mettre en évidence que les états d'une entité peuvent être une abstraction plus ou moins fine des éléments physiques. Cela permet aux applications de contrôler et superviser ensemble des groupes d'éléments physiques à différents niveaux de granularité.

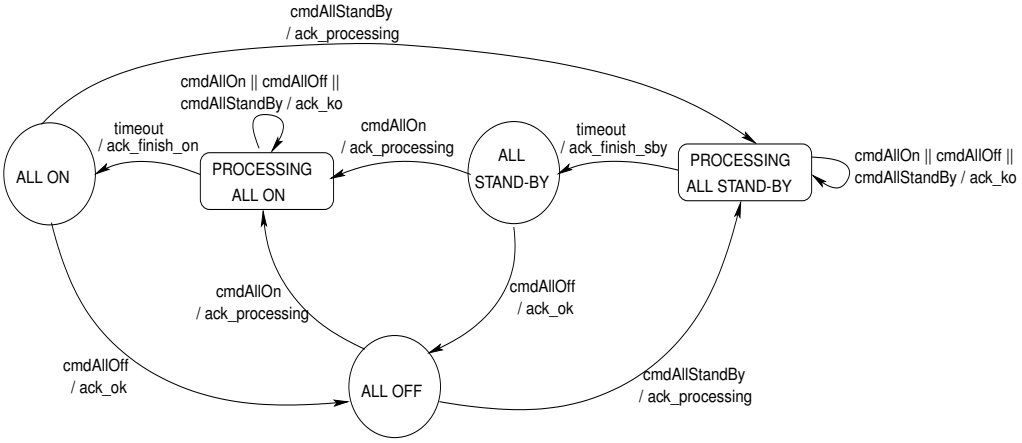


FIGURE 5.9 – Modèle de l'entité associée au groupe de lampadaires GL. Les états sous forme de rectangles sont des états temporisés (cf. section 2.2.1.3). Les entrées correspondent à des entrées contrôlables par les applications et les sorties à des acquittements envoyés aux applications (cf. section 5.3.1.3).

5.3.3.2 Protection de plusieurs équipements

Lors de l'abstraction de plusieurs équipements par une entité contrôlable, il peut être nécessaire de protéger les équipements contre des utilisations incohérentes. Des états sont de fait interdits lorsqu'ils ne sont atteignables d'aucune façon et, par conséquent, ne peuvent être la cible de requête de changement d'état envoyée par une application.

Nous prenons l'exemple des quatre feux tricolores (F_1, F_2, F_3 et F_4) qui composent le carrefour C. Si ces feux tricolores n'ont pas vocation à être controlés séparément, une entité contrôlable est associée à C permettant d'autoriser certaines voies à circuler et d'autres non via le contrôle des feux tricolores.

La figure 5.10 illustre le modèle de cette entité. Cette entité permet d'autoriser le sens de passage du Nord au Sud ou d'Est en Ouest en actionnant les feux correspondants à ces sens de passage. Afin d'éviter que les quatre feux du carrefour puissent être verts ou oranges au même moment, l'entité ne possède pas d'états représentant ces comportement. De ce fait, les applications ne peuvent effectuer des actions dangereuses pour les usagers de la route.

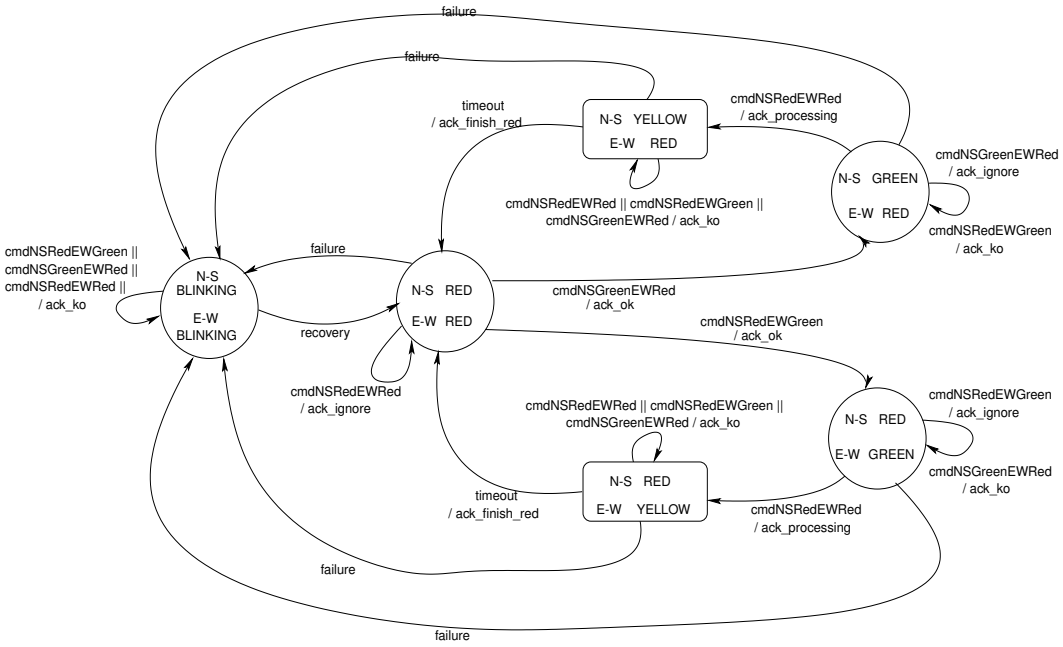


FIGURE 5.10 – Modèle d'entité du carrefour C

5.4 Principes de représentation des entités par des automates

5.4.1 Structure particulière des automates

5.4.1.1 Éléments des automates conduisant à cette structure particulière

Les automates qui décrivent les entités ont une structure particulière qui est exploitée pour générer des ressources REST (cf. sous-section 5.5.2). La figure 5.11 est l'automate d'une entité servant de cas général pour illustrer la structure particulière des automates. Cette entité ne correspond à aucun élément physique, ainsi les états n'ont pas de signification particulière.

Comme décrit dans les paragraphes suivants, les états intermédiaires et les entrées utilisées pour changer d'état, suite aux requêtes reçues provenant des applications, conduisent à cette structure. On note $TS = \{TS_1, TS_2, ..., TS_n\}$ les états pouvant être la cible de requêtes de changement d'état émises par les applications. Ici, seuls les états A, B et C peuvent demander à être atteints par les applications.

Entrées utilisées lors de requêtes de changement d'état Un état TS_i est atteignable par les applications uniquement au moyen d'une seule et unique entrée, que nous notons TI_i, depuis un autre état. Dans la figure précédente, seule l'entrée cmdC permet d'atteindre l'état C que ce soit depuis les états interm2 A, A ou B. De même pour l'état B qui est atteint uniquement au moyen de l'entrée cmdB depuis les état D et C ainsi que pour l'état A atteint au moyen de l'entrée cmdA depuis les

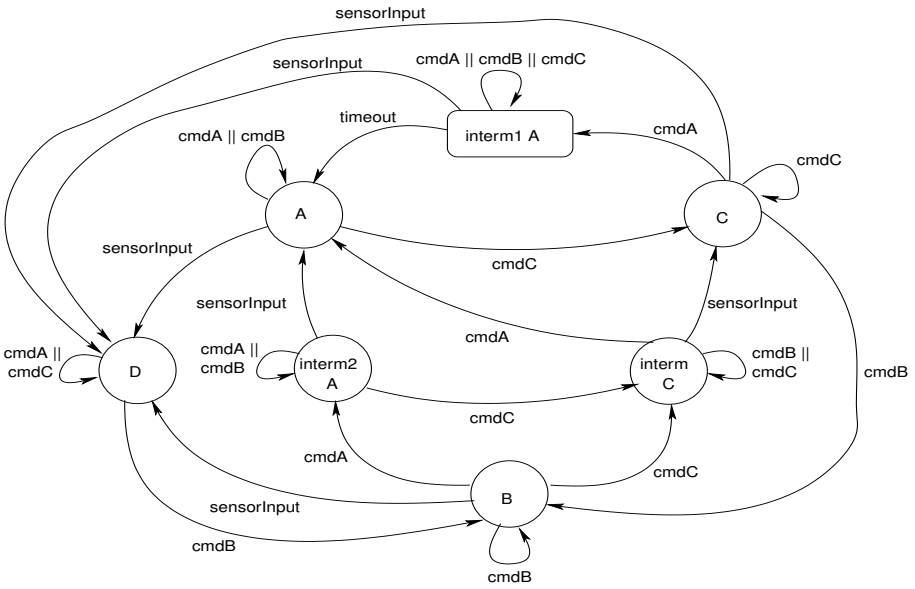


FIGURE 5.11 – Exemple type illustrant la structure particulière de nos automates

états C, interm C et B. En revanche, puisque l'état D n'est pas atteignable, il n'y a pas d'entrée cmdD qui existe.

Réciproquement, une entrée TI_i est associée à un seul et unique état TS_i. Dans notre exemple la commande cmdB ne sert qu'à aller dans l'état B et non dans un autre état, de même pour les autres entrées cmdC pour l'état C et cmdA pour l'état A.

A noter qu'un état TS_i qui est atteint au moyen de l'entrée TI_i correspondante, n'est pas forcément atteint via une seule transition. C'est le cas lorsqu'il existe un état intermédiaire dont nous décrivons les particularités dans le paragraphe suivant. Par exemple, l'état A est atteint au moyen de l'entrée cmdA depuis l'état C, mais deux transitions sont nécessaires : celle depuis l'état C vers l'état intermédiaire interm1 A (labelisée cmdA) et celle depuis l'état interm1 A vers l'état A (labellisée timeout).

États intermédiaires Il ne peut y avoir au plus qu'un seul état intermédiaire qui précède un état atteignable TS_i. Cet état intermédiaire est nécessairement atteint par une transition comportant l'entrée TI_i. Comme illustré dans la figure 5.11 où l'état intermédiaire interm1 A est atteint depuis l'état C par la transition comportant l'entrée cmdA, cet état intermédiaire précède par conséquent l'état A.

Un état intermédiaire peut être utilisé par plusieurs autres états pour atteindre un état TS_i. Cet état intermédiaire est atteint depuis les autres états par une transition comportant l'entrée TI_i. Dans notre exemple, l'état intermédiaire interm2 A est atteint depuis les états B et C via pour chacune une transition comportant l'entrée cmdA. Cet état intermédiaire précède l'état A.

Les automates ne comportent que deux configurations d'états intermédiaires :

ceux donnant des garanties pour atteindre un TS_i et ceux ne pouvant en donner. Dans le premier cas, cela se traduit par la présence d'une seule et unique transition sortante de l'état intermédiaire vers un état TS_i. Tel qu'illustré dans la figure 5.11, l'état temporisé interm1 A permettra d'atteindre toujours l'état A depuis l'état C.

Dans le second cas, où un état intermédiaire ne garantit pas d'atteindre un état TS_i, cela se traduit par la présence d'au moins deux transitions sortantes depuis cet état intermédiaire. Par exemple, l'état intermédiaire interm C peut atteindre l'état cible C, via la transition comportant l'entrée sensorInput. Cependant, aucune garantie n'est donnée pour atteindre C puisque interm C a une seconde transition sortante qui comporte l'entrée cmdA pour atteindre l'état A.

Enfin, il n'y a pas de restriction particulière concernant les entrées utilisées par un état intermédiaire pour atteindre l'état souhaité par une application : il peut s'agir de temps ou d'un seuil atteint par une mesure de capteur par exemple. Dans la figure précédente, l'état intermédiaire interm1 A atteint l'état A souhaité suite à un certain délai, alors que l'état intermédiaire interm C atteint l'état C souhaité après qu'une valeur de capteur ait été atteinte.

5.4.1.2 Interrogation et commande de l'état courant

Tout état qui est l'état courant d'un modèle d'entité peut être interrogé. En revanche, en fonction de l'état courant il peut exister des états commandables et non-commandables par les applications. Cette différence s'effectue en fonction des transitions qui ont une entrée permettant de satisfaire les requêtes de changement d'état des applications.

Un état non-commandable est un état atteint depuis l'état courant par une transition qui a une entrée ne faisant pas partie de l'ensemble TI. Par conséquent, un état est non-commandable lorsque qu'il est atteignable par une transition qui a une entrée associée à une unité de temps ou à une ou plusieurs mesure(s) de capteurs.

Par exemple, sensorInput ne fait pas partie de l'ensemble TI. Par conséquent, dans l'exemple de la figure 5.11, l'état D est non-commandable lorsque l'état courant est l'état A, B, ou interm1 A.

Similairement, l'état A est non-commandable par les applications lorsque l'état courant est interm1 A. Cependant, nous pouvons exploiter le fait que interm1 A soit un état intermédiaire de A pour différencier la sémantique de non-commandable entre l'état A et l'état D. En effet, l'état A est non-commandable puisqu'une requête a été envoyée pour l'atteindre, amenant à aller dans l'état interm1 A ; ce qui n'est pas le cas pour l'état D. Nous verrons en section 5.5.2 comment nous reflétons cette information aux applications. Enfin, un état commandable est un état qui peut être atteint depuis l'état courant par une transition qui a une entrée faisant partie de l'ensemble TI via, ou non, un état intermédiaire. Par exemple, l'état A est commandable depuis l'état C, interm C et B.

5.4.1.3 Génération des acquittements

Les acquittements sont générés et renvoyés automatiquement, sur la base de la structure des automates, aux applications qui ont souhaité effectuer un changement d'état.

L'acquittement ack_ok est renvoyé automatiquement à une application lorsqu'un état est atteint directement via une transition depuis un autre état. Un acquittement ack_processing est renvoyé automatiquement à une application lorsqu'un état intermédiaire est atteint. Concernant l'acquittement ack_finish, celui-ci est retourné à l'application après que l'état cible voulu par l'application ait été atteint au moyen d'une transition sortante d'un état intermédiaire. Un suffixe correspondant au nom de l'état atteint est ajouté à cet acquittement.

Lorsque l'état cible demandé est le même que l'état courant de l'automate, un acquittement ack_ignore est renvoyé à l'application. Formellement il s'agit d'une transition sur soi-même d'un état comportant les différentes entrées permettant de l'atteindre. Enfin, un acquittement ack_ko est retourné à une application lorsque l'état actuel reste l'état courant de l'automate et que celui-ci n'est pas l'état cible de l'application. Formellement dans l'automate cela se traduit par une transition sur soi-même d'un état, laquelle comporte des entrées utilisées pour atteindre d'autres états.

5.4.2 Extension de la notion d'états par des variables

Il peut être fastidieux de modéliser les automates des entités par des états, surtout lorsque cela conduit à obtenir des automates avec un grand nombre des états. Dans ce cas, au lieu de décrire les automates par des états, nous utilisons des variables qui sont une extension de notation.

En effet, les variables ont la même signification que les états car elles sont une abstraction des éléments physiques associés aux entités via la consolidation de données des capteurs ou l'utilisation d'actionneurs. Cependant, les variables ont une notation simplifiée permettant aux applications de ne pas devoir manipuler un trop grand nombre d'états. Puisque les variables sont l'équivalent des états, elles peuvent être également commandables ou non.

Le modèle d'entité du plot rétractable PR illustré en figure 5.5 est composé de deux variables correspondant à la hauteur minimale et maximale du plot en centimètres. Ces variables sont non commandables puisqu'elles ne peuvent pas être modifiées. Cependant, afin que les applications puissent connaître les caractéristiques d'un plot, elles peuvent être interrogées.

La présence de variables commandables dans les automates des entités contrôlables présente une facilité d'utilisation pour les applications. Pour illustrer nos propos, nous prenons le cas d'une entité contrôlable qui représente le panneau d'affichage digital PAD dont le modèle est illustré dans la figure 5.12. Ici, la variable msg correspond au message affiché sur le panneau digital. Elle est commandable et permet de mettre à jour le message affiché comme nous l'illustrons avec la méthode

void setMessage() du modèle. Bien qu'il serait possible d'utiliser des états pour représenter toutes combinaisons possibles de message à afficher, une variable est plus facile à utiliser dans ce cas.

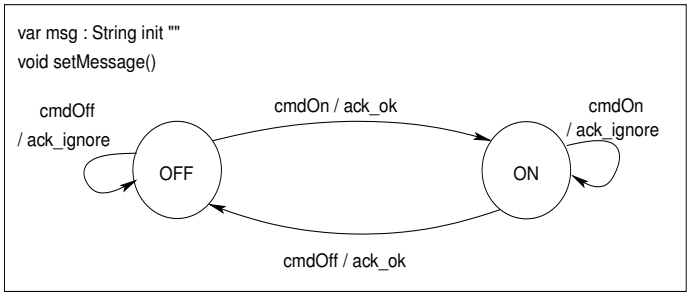


FIGURE 5.12 – Modèle de l'entité représentant le panneau d'affichage digital PAD

Suite à la modification d'une variable par une application, les mêmes acquittements que décrits en section 5.3.1.3 peuvent être renvoyés aux applications, exceptés les acquittements ack_processing et ack_finish, qui sont liés à la structure des automates et à la présence d'état intermédiaire.

Dans l'exemple du modèle de l'entité PAD, lors du changement de valeur de la variable msg par une application, un acquittement ack_ok est retourné lorsque l'ordre a été envoyé à l'actionneur de l'équipement et a été acquitté. Si la valeur souhaitée est la même que celle actuelle, un ack_ignore est retourné. Enfin, un acquittement ack_ko est renvoyé si la variable msg a des contraintes sur ses valeurs, telle que sur la longueur de la chaine de caractère par exemple.

Pour illustrer un cas où une variable non-commandable seule fait sens d'être utilisée, nous prenons comme exemple l'entité représentant un ensemble d'éléments physiques à savoir les places de parking PP_1 à PP_8. Ici, une variable, dont l'échelle de valeur est comprise entre 0 et 8 inclus, pourrait représenter le nombre de places de parking disponibles. Bien qu'il serait possible de discrétiser en 9 états, il est ici plus simple de représenter le nombre de places de parking disponibles par une variable. A noter que, de la même manière que les états, cette variable est indépendante du type de capteur utilisé pour connaître la présence d'un véhicule stationné.

5.5 Correspondances entre entités et ressources REST

5.5.1 Liens entre les modèles d'entités et le critère HATEOAS

Comme décrit en section 4.4, nous avons choisi d'adhérer aux principes du style architectural REST. Pour rappel, HATEOAS désigne le fait que les représentations des ressources contiennent des liens hypermédias vers d'autres ressources avec lesquels le client va pouvoir ainsi interagir. Le critère HATEOAS est également présent dans une architecture orientée ressource. Il est indiqué que les ressources doivent être connectées les unes ou autres lorsqu'il existe des relations sémantiques entre elles [145].

Dans notre travail, les modèles comportent des états liés par des transitions. Depuis l'état courant il est possible de connaître quels sont les états qui peuvent être atteints en fonction des transitions existantes. Dans notre contexte, les états pouvant être atteints sont commandables ou non par les applications (cf. section 5.4.1.2). Ainsi, les notions de transitions et d'états présents dans les automates recoupent les mêmes notions évoquées dans le critère HATEOAS où des liens doivent être présents entre les ressources.

La similitude entre le critère HATEOAS et les automates est un argument supplémentaire pour l'utilisation de ressources REST afin de représenter les automates des entités. Cette similitude se traduit par le fait qu'en fonction de l'état courant, il peut être indiqué au client les états commandables ou non. Cela permet au client de découvrir au fur et à mesure les états des automates et d'être faiblement couplé au serveur.

5.5.2 Transformation des modèles d'entités en ressources

5.5.2.1 Génération des ressources

A partir d'un modèle d'entité, nous générons automatiquement plusieurs ressources REST en exploitant le fait que les automates aient une structure particulière (cf. section 5.4.1).

Pour illustrer la génération de ressources, nous prenons pour exemple l'automate de l'entité du plot rétractable PR illustrée dans la figure 5.5. Pour rappel, cet automate possède quatre états UP, DOWN, MOVING UP et MOVING UP . Seuls les deux premiers états cités sont commandables par les applications. Cet automate possède également deux variables MAX et MIN qui ne sont pas commandables par les applications.

Le principe de génération des ressources est le suivant :

— Une ressource R_A est associée à un automate A contenant toutes les informations à propos de l'automate (valeur des variables, de l'état courant, etc.).

— Une sous-ressource notée R_A/cs est générée et représente l'état courant d'un automate A pouvant être modifié.

— A chaque état de l'automate A, noté $s_i = \{s_1, s_2, ...\}$, est associé une sous-ressource que nous notons $R_A/s_1, R_A/s_2, ...$ Ces sous-ressources permettent d'identifier les états pour qu'elles contiennent des annotations sémantiques (cf. section 5.5.3).

— A chaque variable de l'automate A, notée $v_i = \{v_1, v_2, ...\}$, est associée une sous-ressource notée $R_A/v_1, R_A/v_2, ...$ Ces sous-ressources permettent à des variables commandables d'être modifiées.

D'après le modèle de l'entité du plot PR, les ressources générées seraient les suivantes :

— Une ressource R_{PR} associée à l'automate de l'entité de PR.

— Une sous-ressource R_{PR}/cs associée à l'état courant de l'automate de l'entité de PR. Cette ressource sera utilisée afin de faire monter ou descendre le plot.
— Quatre sous-ressources associées aux différents états de l'automate : R_{PR}/s_{up} pour l'état UP, $R_{PR}/s_{movingup}$ pour l'état MOVING UP, $R_{PR}/s_{movingdown}$ pour l'état MOVING DOWN et R_{PR}/s_{down} pour l'état DOWN.
— Deux sous-ressources associées aux deux variables : R_{PR}/v_{min} pour la variable MIN et R_{PR}/v_{max} pour la variable MAX.

Il est à noter que l'URI de chacune des ressources et sous-ressources n'a pas de sémantique particulière (cf. section 7.2.2.3). La sémantique des automates, états et variables devrait être incluse dans les représentations des ressources dès lors que des ontologies de la ville intelligente seraient intégrées à notre travail (cf. section 8.2.1.1).

5.5.2.2 Représentations de la ressource d'un automate

La représentation de la ressource R_A contient des informations liées à l'état courant ainsi que les états et variables pouvant être commandables ou non de l'automate A. Ce choix a été fait afin que les applications ne multiplient pas les requêtes pour obtenir ces informations.

Nous illustrons un exemple typique de la représentation d'une ressource R_A en prenant pour exemple la représentation de la ressource R_{PR} illustrée en figure 5.13.

La ligne 2 correspond à la requête envoyée par l'application à la ressource R_{PR}. Il s'agit d'une requête GET puisque ce verbe permet d'obtenir la représentation d'une ressource (lignes 5 à 20). La représentation de R_{PR} contient :

— L'état courant de l'entité (ligne 12) : son nom (ligne 14) ainsi qu'un lien vers la ressource R_A/cs liée à l'état courant. La représentation de cette sous-ressource peut être changée par les applications afin de mettre à jour l'état d'une entité, comme décrit en section 5.5.2.3.
— Les états commandables (ligne 17) : le nom de ceux-ci (ligne 17) et un URI auquel sont associées leurs sous-ressources respectives. à savoir uniquement R_{PR}/s_{up} ici (ligne 18). Cette sous-ressource permet à un état d'être identifié avant de pouvoir être annotée par des méta-données sémantiques.
— Les variables non commandables (lignes 5 à 14) : leur nom (lignes 6 et 12), valeur (lignes 7 et 11) ainsi que l'URI associé à leur sous-ressource qui est R_{PR}/v_{min}) (ligne 8) et R_{PR}/v_{max}) (ligne 13). Un URI permet à la fois d'identifier une variable et de changer la valeur de celle-ci lorsqu'elle est commandable (cf. section 5.5.2.3).

La représentation d'une ressource R_A peut également contenir une liste d'états non-commandables et de variables commandables. Cette liste contient les mêmes informations que celles associées aux états commandables (nom, URI) et variables non-commandables (nom, valeur, URI). Les variables et états commandables/non-commandables sont déterminés à partir de l'automate A et de sa structure particulière comme nous le décrivions précédemment (cf. section 5.4.1).

```
1 --->
2 GET /PR/
3 --->
4 <----
5     "uncontrolable_var": [{
6         "var_name": "MIN",
7         "var_value": "0",
8         "var_uri": "/PR/v_min"
9     },
10    "uncontrolable_var": {
12        "var_name": "MAX",
11        "var_value": "110",
13        "var_uri": "/PR/v_max"
14    }],
12    "current_state": {
13        "current_state_uri": "/cs",
14        "state_name": "DOWN",
15        "state_uri": "/PR/s_down"
16    },
17    "controlable_state": [{
18        "state_name": "UP",
19        "stateUri": "/PR/s_up"
20    }]
21 <----
```

FIGURE 5.13 – Représentation de la ressource R_{PR} associée au modèle de l'entité PR

Ainsi, la représentation d'une ressource R_A permet aux applications de connaître les états et variables commandables. La présence de nombreux liens vers les sous-ressources nous permet de respecter le principe HATEOAS. Les applications doivent utiliser ces liens pour modifier ou lire l'état courant de l'entité (sous-ressource R_A/cs) et/ou pour modifier la valeur des variables (sous-ressource R_A/v_i).

La représentation d'une sous-ressource R_A comporte également une déclinaison d'un état considéré comme étant non-commandable, lorsque l'état courant est un état commandable (cf. section 5.4.1.2). Nous illustrons un exemple dans la figure 5.14 où une application obtient la représentation de la ressource R_{PR} qui a pour état courant l'état intermédiaire MOVING UP.

Ici, l'état UP n'est pas mentionné comme étant non-commandable mais comme étant l'état futur de l'entité PR (ligne 15 à 18). Cette distinction peut être effectuée grâce à la structure particulière d'un automate qui ne comporte au plus qu'un état intermédiaire avant un état commandable. De fait, il ne peut avoir qu'un seul état futur dans la représentation de la ressource R_A. Un état futur indique aux applica-

```
 1 --->
 2 GET /PR/
 3 --->
 4 <----
 5     [...]
 6     "current_state": {
 7         "current_state_uri": "/PR/cs",
 8         "state_name": "MOVING UP",
 9         "state_uri": "/PR/s_moving_up"
10     },
11     "controlable_state": [{
12         "state_name": "DOWN",
13         "stateUri": "/PR/s_down"
14     }],
15     "future_state": {
16         "state_name": "UP",
17         "stateUri": "/PR/s_up"
18     }
19 <----
```

FIGURE 5.14 – Représentation de la ressource R_{PR} associée au modèle de l'entité PR lorsque l'état courant est un état intermédiaire.

tions qu'elles n'ont pas besoin d'interroger continuellement l'état courant de l'entité pour connaître sa prochaine valeur, contrairement à un état non-commandable.

5.5.2.3 Représentations des sous-ressources

Sous-ressource associée à l'état courant La représentation de la sous-ressource R_A/cs contient le nom de l'état courant, un lien vers celui-ci ainsi que l'état futur et la liste des états commandables ou non à partir de cet état courant. Puisque notre travail s'inscrit dans une architecture orientée ressource, un lien est également présent pointant vers l'URI de la ressource parente $R_A/$. La représentation de R_A étant un sur-ensemble de la représentation de R_A/cs afin de permettre aux applications d'obtenir toutes les informations à propos d'une entité en une seule requête.

Nous en illustrons un exemple dans la figure 5.15 où une requête GET est envoyée afin d'obtenir la représentation de la ressource R_{PR}/cs. La ligne 18 montre un lien vers la ressource R_{PR} afin que l'application puisse à tout moment obtenir l'intégralité des informations liées à l'entité PR. L'état courant de l'entité PR est présent, à savoir MOVING UP (lignes 5 à 9). Puisque l'état courant est un état intermédiaire, un état futur est présent à savoir l'état UP (lignes 14 à 17). L'application peut utiliser l'état DOWN indiqué comme étant commandable pour modifier l'état de l'entité PR. Il doit envoyer une requête de mise à jour vers l'URI indiqué à la ligne

8.

```
1 --->
2 GET /PR/cs
3 --->
4 <----
5     "current_state": {
6         "current_state_uri": "/PR/cs",
7         "state_name": "MOVING UP",
8         "state_uri": "/PR/s_moving_up"
9     },
10    "controlable_state": [{
11        "state_name": "DOWN",
12        "stateUri": "/PR/s_down"
13    }]
14    "future_state": {
15        "state_name": "UP",
16        "stateUri": "/PR/s_up"
17    }
18    "entity": "/PR"
19<----
```

FIGURE 5.15 – Représentation de la ressource R_{PR}/cs dont l'état courant est MOVING UP.

Nous illustrons cette modification de la représentation de la ressource R_A/cs dans la figure 5.16. L'application envoie une requête PUT pour mettre à jour la représentation d'une ressource (ligne 2). Le contenu de cette mise à jour comprend le nom de l'état à changer, ici l'état DOWN (lignes 3 à 5). Nous verrons en section 5.5.2.4 que les acquittements présents dans l'automate correspondent aux codes de réponse du protocole utilisé pour les interactions avec les ressources REST.

```
1 --->
2 PUT /PR/cs
3     "current_state": {
4         "state_name": "DOWN"
5     }
6 --->
```

FIGURE 5.16 – Modification de la représentation de la ressource R_{PR}/cs afin que l'état courant de PR soit DOWN.

Sous-ressources associés aux états La représentation d'une sous-ressource R_A/s_i contient le nom de l'état et sa valeur. Lorsque l'état associé à cette ressource

est l'état courant, un lien pointant vers l'URI R_A/cs apparaît afin d'obtenir plus de détails à propos de l'état courant et des états successeurs (futur, commandable, non-commandable). Un URI vers la ressource parente R_A est également présent afin d'obtenir toutes les informations liées à l'automate A.

L'objectif principal d'une sous-ressource R_A/s_i est d'identifier un état s_i auquel il est associé pour que la sémantique de ce dernier puisse être décrite (cf. section 5.5.3).

A titre d'exemple, nous illustrons la représentation de la sous-ressource R_{PR}/s_{up} en figure 5.17. Une requête GET est envoyée (ligne 2) permettant d'obtenir des informations à propos de l'état UP de PR. La valeur de cet état est renvoyée (ligne 6) et, puisque cet état est l'état courant, un lien vers l'URI de la sous-ressource associée à l'état courant de PR est également renvoyé. Cela permet aux applications d'aller chercher plus de détails à propos des modifications qu'elles peuvent effectuer sur l'état courant de PR.

```
 1 --->
 2 GET /PR/s_up
 3 --->
 4 <----
 5     "state": {
 6         "state_name": "UP",
 7         "current_state_uri": "/PR/cs"
 9     },
10     "entity": "/PR"
11<----
```

FIGURE 5.17 – Représentation de la ressource R_{PR}/s_{up} lorsque l'état courant est UP.

Sous-ressources associées aux variables La représentation d'une sous-ressource R_A/v_i comprend le nom, la valeur de la variable v_i de l'automate A. Un lien vers l'URI de la ressource parente R_A est également présent pour les mêmes raisons qu'évoquées précédemment.

Nous illustrons un exemple en figure 5.18 d'une requête d'obtention de la représentation de la ressource R_{PAD}/v_{msg}. Cette ressource est associée à la variable commandable msg du modèle de l'entité PAD. La valeur et le nom de cette variable sont présents (lignes 6 et 7). Puisque cette variable est commandable (ligne 5), elle peut faire l'objet d'une requête de mise à jour. L'URI présent à la ligne 8 indique la ressource vers laquelle la requête de mise à jour doit être envoyée.

Nous illustrons dans la figure 5.19 une requête de mise à jour de la variable msg. Il s'agit d'une requête PUT (ligne 2) contenant la nouvelle valeur de la variable qui doit être mise à jour (lignes 3 à 5). De la même façon que pour un changement d'état, les codes de réponses sont standardisés pour les requêtes de mise à jour des variables.

```
1 --->
2 GET /PAD/v_msg
3 --->
4 <----
5     "controlable_var": {
6         "var_name": "msg",
7         "var_value": "hello world",
8         "var_uri": "/PAD/v_msg"
9     },
10    "entity": "/PAD"
11 <----
```

FIGURE 5.18 – Représentation de la sous-ressource R_{PAD}/v_{msg} associée à la variable commandable msg du modèle de l'entité PAD.

```
1 --->
2 PUT /PAD/v_msg
3     "controlable_var": {
4         "var_value": "new message"
5     }
6 --->
```

FIGURE 5.19 – Modification de la représentation de la sous-ressource R_{PAD}/v_{msg} afin de modifier la variable commandable msg.

5.5.2.4 Acquittements et codes de réponse

Dans une architecture orientée ressource, le protocole utilisé pour les interactions possède des verbes et des codes de réponses destinés aux applications (cf. section 2.1.1.1). Nous avons mis en correspondance les acquittements avec les différents codes de réponse du protocole utilisé, qui est CoAP dans notre implémentation (cf. section 7.2.1).

Nous avons transposé la signification des codes de réponses standards avec la signification des acquittements que nous générons. Par exemple, une application obtiendra le code de réponse standardisé "2.04 CHANGED" lorsque sa requête PUT de changement d'état a été correctement effectuée. Ici, l'acquittement ack_ok présent dans les différents automates a pour signification un succès de changement. De ce fait, il correspond au code de réponse standardisé "2.04 CHANGED" qui signifie le succès d'une requête PUT. Il en va de même pour les autres acquittements (ack_ko, ack_processing, etc.) qui ont chacun une correspondance avec un code de réponse standardisé.

Les interactions entre les applications et les ressources sont ainsi standardisées. Nous donnons plus d'informations concernant les codes de réponses que nous avons utilisés dans la présentation de notre preuve de concept (cf. section 7.2.5).

5.5.3 Annotations sémantiques des ressources

Nous avons choisi de générer une multitude de ressources et d'ajouter des liens dans les représentations des ressources pour pouvoir intégrer des annotations sémantiques. Les annotations sémantiques permettent de décrire la signification des ressources et des liens qui existent entre les ressources (cf. section 5.5.2.2 et 5.5.2.3). Ces annotations sémantiques sont issues d'ontologies (cf. section 2.3).

Par exemple, la sémantique du lien permettant aux applications de changer l'état d'une entité pourrait être décrite. Cette sémantique permettrait aux applications de connaître la nature du lien et de savoir d'elles-mêmes qu'il s'agit d'un lien permettant de changer l'état d'une entité. Ainsi, les applications ne s'appuieraient plus sur des connaissances a priori pour envoyer des requêtes de changement d'état.

De même, les ressources étant identifiées par des URIs, la sémantique de chacune d'entre elles pourrait être décrite. Par exemple, la sémantique des différents états pourrait être décrite pour que les applications sachent à quoi correspondent les états qu'elles supervisent et modifient sans avoir besoin de connaissances a priori à ce sujet.

Nous reviendrons sur ces aspects sémantiques, au travers de la réutilisation d'ontologies, dans les perspectives de notre travail (cf. section 8.2.1.1).

5.6 Conclusion

Dans ce chapitre nous avons choisi de représenter les éléments physiques, qui sont le point commun de chaque ville, par des entités. Les entités sont une abstraction à différents niveaux de granularité des éléments physiques. Ils permettent de protéger des équipements de la ville contre de mauvaise utilisation. De plus, ils sont généralisables à toute instance de la ville intelligente.

Pour représenter les entités, nous avons utilisé des modèles issus des systèmes réactifs puisque la ville intelligente est semblable à ceux-ci. Les entités ont des états qui peuvent être contrôlés et supervisés par les applications. L'état d'une entité varie suivant les valeurs des capteurs récoltées et les requêtes envoyées par les applications. L'évolution de l'état d'une entité est décrite par le modèle de l'entité c'est-à-dire son automate.

Nous avons décrit la démarche d'invention des entités en présentant les éléments des automates. Certains de ces éléments sont destinés aux applications, à savoir des entrées pour satisfaire des requêtes de changement d'état ainsi que des acquittements pour notifier les applications. Les applications peuvent être restreintes à ne pas pouvoir changer certains états pour la protection des équipements. Par ailleurs, nous avons présenté les états des modèles. Ils peuvent être issus d'unité de temps écoulée ou d'une consolidation de données indépendantes du type de capteur. Les états d'un modèle peuvent être intermédiaires pour représenter des contraintes physiques/fonctionnelles des équipements. Les états peuvent être également agrégés ensemble pour représenter une entité de forte granularité.

Enfin, nous avons décrit la structure particulière des automates qui ont des

états commandables ou non par les applications. Cette structure particulière est exploitée pour générer automatiquement des ressources REST. Nous avons remarqué que le choix d'une architecture orientée ressources était cohérent avec le choix de nos modèles compte-tenu des mêmes notions de liens/transitions qui existent dans ces derniers. Après avoir généré des ressources REST qui peuvent être annotées sémantiquement, nous avons décrit comment celles-ci peuvent être utilisées pour superviser et contrôler la ville intelligente.

Chapitre 6

Mécanismes de base pour le contrôle partagé d'entités

Ce chapitre décrit l'identification des défis qu'engendre le partage d'entités contrôlables dans les systèmes cyber-physiques et propose des solutions pour les relever.

Nous décrivons en section 6.1.1 que des conflits peuvent se produire lors du contrôle d'une entité. Un conflit peut se produire si une entité reçoit une requête de changement d'état alors qu'elle traite déjà une autre requête, ou si des requêtes lui arrivent avant qu'elle ait commencé à en traiter une. Pour y remédier, nous proposons l'exclusion mutuelle des requêtes de changement d'état (cf. section 6.2).

Nous avons identifié également qu'un mécanisme doit permettre aux applications d'effectuer des séquences de changement d'états, comme s'il s'agissait d'une seule et unique opération (cf. section 6.1.2). Les équipements n'étant pas tous reliés physiquement les uns aux autres, un moyen de les coordonner doit être fourni. Nous proposons ainsi un mécanisme de coordination atomique permettant cela.

Nous avons identifié que compte-tenu des caractéristiques de l'environnement, des opérations de recouvrement tel que le rollback [67] ou la compensation [60] n'ont pas de sens d'être présentes dans la conception d'un algorithme de coordination d'entités (cf. section 6.3).

En section 6.4, nous présentons notre algorithme de coordination et nous en montrons la validation effectuée via l'outil de model-checking SPIN.

Sommaire

6.1 Défis identifiés pour l'accès aux entités

6.1.1 Contrôle de concurrence d'une seule entité contrôlable

Sur la base de l'exemple décrit en section 4.3, nous décrivons un cas de conflit lorsque l'entité PR, un plot rétractable, reçoit deux requêtes de changement d'état émises par deux applications concurrentes. Pour rappel, la figure 6.1 illustre son automate à états finis.

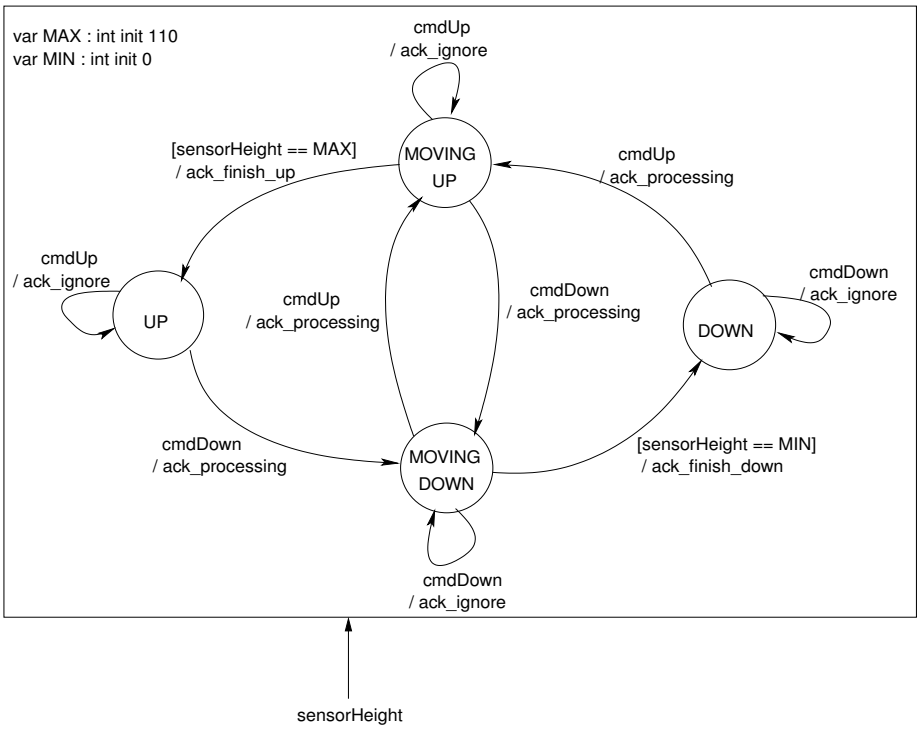


FIGURE 6.1 – Modèle de l'entité associée au plot rétractable PR

Le diagramme de séquence illustré en figure 6.2 montre la concurrence de deux requêtes de changement d'état vers PR, notées R_1 et R_2, et visant respectivement à atteindre l'état DOWN depuis UP et l'état UP depuis MOVING DOWN. Ces requêtes sont émises par des applications différentes, notées App_1 pour R_1 et App_2 pour R_2. Les notes de commentaire présentes sur la droite de la figure indiquent l'état courant de l'entité PR à différents instants.

Apparition d'un conflit PR reçoit et accepte R_1 puis envoie l'acquittement ack_processing à App_1, suite à quoi l'état MOVING DOWN est atteint. Un conflit

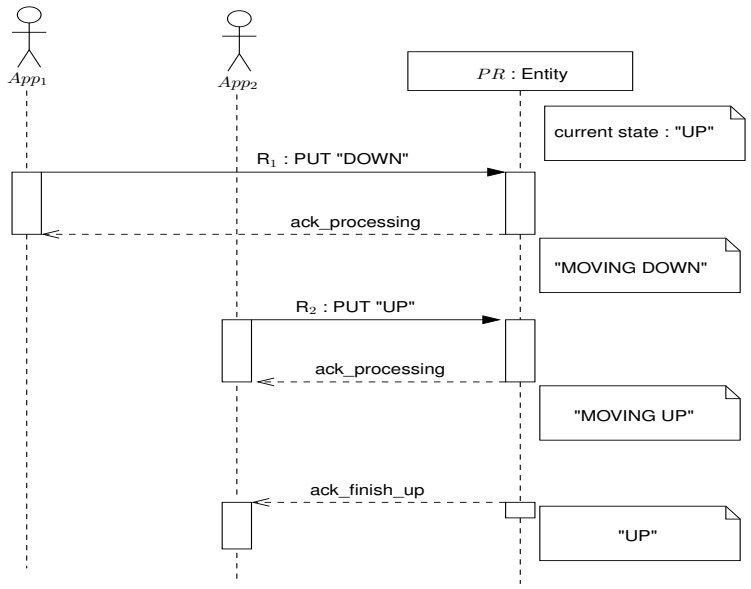


FIGURE 6.2 – Cas de conflit entre les applications App_1 et App_2

apparaît lorsque PR reçoit la requête R_2 alors que l'état DOWN n'a pas encore été atteint. Bien que ce changement d'état soit valide, puisqu'une transition existe depuis l'état MOVING DOWN vers l'état UP via MOVING UP, l'application App_1 est lésée puisqu'elle voit sa requête de changement d'état R_1 interrompue.

Pour chaque entité contrôlable, il existe un laps de temps avant d'atteindre un état cible. Ce laps de temps est dû à la communication avec les actionneurs sous-jacents et à la durée de l'état intermédiaire (ici, MOVING DOWN) précédant l'état cible. Un conflit se produit lorsqu'une requête de changement d'état est acceptée pendant ce laps de temps (ici, R_2, cette requête étant envoyée par une application pouvant être différente ou non (ici, App_2) de celle qui a envoyée la requête en cours d'exécution.

Conséquence d'un conflit Après avoir accepté la requête R_2, PR envoie un acquittement ack_processing à l'application App_2, suite à quoi l'état MOVING UP est atteint. Une fois la durée de l'état temporisé terminée, PR envoie l'acquittement ack_finish_up à App_2. Cet exemple montre que App_1 ne recevra jamais l'acquittement ack_finish_down bien qu'elle ait reçue précédemment l'acquittement ack_processing, puisque R_2 a écrasé sa requête.

De manière générale, la conséquence d'un conflit est l'écrasement du changement d'état en cours par la dernière requête de changement d'état reçu. Ainsi l'entité contrôlable atteint le nouvel état cible, lié à la dernière requête reçue, sans avoir atteint le précédent. La présence de conflits amène les applications à ne jamais avoir de garantie que leur requête de changement d'état aura l'effet escompté, même si elle a été acceptée par l'entité contrôlable.

Par conséquent, nous avons identifié qu'un mécanisme de contrôle de concur-

rence était requis pour permettre aux applications d'effectuer un changement d'état complet sans que des conflits surviennent. Ce mécanisme doit assurer à une application qui a reçu une réponse positive à sa requête de changement d'état que l'état cible correspondant sera atteint. Comme décrit en section 6.2, nous proposons un mécanisme de verrouillage basé sur l'exclusion mutuelle des requêtes de changement d'état. Ce mécanisme de contrôle de concurrence garanti qu'un état sera atteint dès lors que les délais de communication vers les actionneurs sont bornés et garantis.

6.1.2 Coordination de multiples entités contrôlables

La coordination de multiples entités contrôlables résulte d'un besoin des applications (cf. section 4.2.2.3) de pouvoir effectuer une opération globale et atomique impliquant le changement d'état de plusieurs entités contrôlables distinctes. Le mécanisme de contrôle de concurrence que nous proposons n'étant conçu que pour le contrôle unitaire d'entité contrôlable, il n'est pas adapté dans le cas présent.

Groupes d'entités contrôlables Il est plus facile de contrôler certaines entités ensemble plutôt que de manière unitaire, tels que les lampadaires le long d'une route ou les feux d'un carrefour. Dans ce cas, nous regroupons ces entités au sein d'un groupe d'entités contrôlables. Un changement d'état d'un modèle de groupe d'entités contrôlables implique plusieurs changements d'états des entités contrôlables de ce groupe.

Tel qu'illustré dans la figure 6.3 par le groupe d'entités GL, ce dernier permet de contrôler les lampadaires L_1, L_2 et L_3. Le modèle de GL est donné en figure 6.3 : lorsque l'état ALL ON est atteint, les lampadaires sous-jacents sont dans l'état ON et de façon similaire pour les états ALL STAND-BY et ALL OFF.

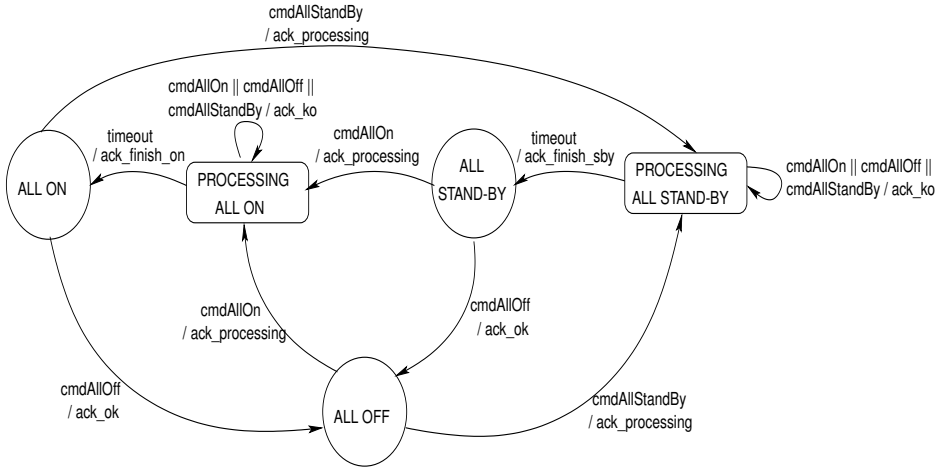


FIGURE 6.3 – Modèle du groupe d'entités contrôlables GL

Pour certains groupes d'équipements courants on peut créer des modèles de groupes, en choisissant a priori les états globaux intéressants (ici, ALL OFF, ALL

ON et ALL STAND-BY). Ces groupes pourraient être hétérogènes mais toujours fixés a priori (cf. section 5.3.3.1).

Cependant, l'utilisation des groupes d'entités contrôlables pour coordonner de manière généralisée les entités ne nous semble pas suffisant. En effet, cela nécessiterait la création de groupes d'entités contrôlables pour chaque configuration de contrôle simultané d'entités en définissant les états de ces groupes. Pour éviter ceci, nous ajoutons la possibilité de créer des groupes à la demande au travers d'un mécanisme de coordination.

Mécanisme de coordination Un mécanisme de coordination doit permettre à une application de définir les entités contrôlables qu'elle souhaite coordonner et la séquence de changement d'états à effectuer de manière atomique. Un tel mécanisme favorise la réutilisation des entités contrôlables puisqu'il permet aux applications de les utiliser en les détournant de leur but principal.

Dans notre exemple typique (cf. section 4.2) l'application qui fluidifie le trafic routier coordonne les feux du carrefour C avec les feux travaux FT_1 et FT_2 illustrés en figure 4.3. Le but principal des feux de travaux est d'alterner le passage des voitures sur un segment de route en travaux, alors que dans le contexte de cette application ils sont utilisés dans un but opportuniste qui est de réguler le trafic sur l'axe routier.

6.2 Gestion de la concurrence sur une entité

6.2.1 Mécanisme de verrouillage

Nous proposons un mécanisme de verrouillage afin de garantir l'exclusion mutuelle des applications. On associe à chaque entité contrôlable e un seul et unique verrou V_e. Afin de changer l'état d'une entité contrôlable e, une application doit préalablement obtenir le verrou V_e qui lui est associé. En revanche, toute application peut interroger l'entité sans verrouillage.

Période de validité d'un verrou Un verrou n'est détenu par une application que durant un certain laps de temps, appelé *période de validité*, après quoi il est relâché automatiquement. Nous avons choisi la mise en place d'une expiration automatique de chaque V_e afin d'éviter que les applications ne monopolisent les entités contrôlables.

Cette durée de validité démarre lors de l'obtention du verrou par une application, et doit être configurée pour être supérieure au temps que prend l'entité e pour effectuer son changement d'état le plus long. Ce temps inclut également les délais de communication avec les actionneurs dont nous supposons qu'ils sont bornés et garantis (cf. section 7.4). La période de validité est donc différente d'une entité contrôlable à une autre.

Une application détentrice d'un verrou est assurée d'effectuer au moins un changement d'état, celui prenant le plus de temps, et peut en effectuer davantage tant

que la période de validité n'a pas expiré. Dans la figure 6.1 illustrant le plot rétractable PR, la durée de validité d'utilisation du verrou est supérieure au délai que met PR pour atteindre l'état DOWN depuis l'état UP, en prenant également en compte les délais de communication avec les actionneurs. On considère que cette durée est la même que pour atteindre UP depuis DOWN.

Obtention et utilisation du verrou par une application Les demandes d'obtention de V_e émises par des applications sont sérialisées par la ressource responsable du mécanisme de contrôle de concurrence (cf. section 7.2.4.1). Elles mènent à deux cas de figure.

Premièrement, si V_e est déjà utilisé, l'application voit sa demande d'obtention rejetée et en est prévenue. Il est de son ressort de re-demander ultérieurement le verrou. Ce choix a été fait car nous considérons que les besoins des applications, pour le contrôle des entités contrôlables, sont variables : certaines requièrent de contrôler immédiatement une entité et d'autres non. Par conséquent, la mise en attente automatique de ces requêtes ne serait pas toujours le bon choix.

Deuxièmement, si V_e n'est pas en cours d'utilisation, un *token* est généré puis renvoyé en réponse à l'application. Un token est un identifiant unique qui est lié au cycle de vie de V_e. Il est créé lorsque V_e est obtenu et est détruit lorsque la durée de validité d'utilisation de V_e expire. Ce token permet d'identifier l'application détentrice de V_e et doit être indiqué dans une requête de changement d'état pour vérifier si l'émetteur est le propriétaire de V_e. Ce choix a été fait afin de respecter les principes d'une architecture REST où le serveur ne garde pas d'informations liées aux clients (statelessness). C'est le client qui stocke cette information.

Il est à noter que pour se prémunir contre l'usurpation d'identité, nous supposons que les applications sont authentifiées par la plateforme avant que la vérification du token ait lieu.

Enfin, si une application tente d'effectuer un changement d'état sans avoir fourni de token ou avec un token dont elle n'est pas le propriétaire, alors la requête est rejetée en indiquant à l'application qu'elle doit fournir un token valide. En considérant dans la figure 6.2 que App_1 vient d'obtenir le verrou, la requête R_2 de App_2 serait rejetée car elle n'inclut pas de token ou inclut un token ayant expiré.

Comme nous le verrons en section 7.3, l'obtention ou non d'un token a une influence sur le modèle de programmation des applications.

Lecture pendant un verrouillage Des applications peuvent continuer à superviser l'entité e bien que V_e soit en cours d'utilisation. Le choix a été fait de ne pas laisser en suspens les requêtes de lecture des applications et donc d'autoriser les lectures dites "fantômes" des états. La conséquence de ce choix étant que les applications sont susceptibles de lire une valeur de l'état qui est sur le point de changer. Les applications sont informées, dans le contenu de la réponse reçue, si un verrou est en cours d'utilisation ou non. Par conséquent, en fonction de ses besoins, une application est laissée juge de la pertinence d'une lecture fantôme.

6.2.2 Exemple d'utilisation d'un verrou

La figure 6.4 illustre un fragment de code d'une application qui envoie une requête de lecture de l'état de l'entité contrôlable PR (ligne 2). Les lignes 6 et 7 indiquent respectivement le succès de la requête, via le code de réponse "2.05 - Content" retourné, ainsi que le début du corps de la réponse. Outre le nom de l'état courant et son URI indiqués aux lignes 11 à 12, les états possibles à atteindre figurent aux lignes 16 à 17. L'attribut "token_uri", à la ligne 19, indique aux applications l'URI de la ressource en charge de la création d'un token.

Puisque nous avons choisi d'adhérer aux principes REST préconisant le principe HATEOAS (cf. section 2.1.1.1), l'attribut "token_uri" est vide lorsque le verrou est en cours d'utilisation. A l'inverse un URI est présent lorsque le verrou est disponible. L'URI permettant d'obtenir un token change à chaque fois qu'un verrou redevient disponible. Ce choix s'explique par le fait que nous voulons éviter qu'une application stocke l'URI de génération des tokens, ce qui pourrait avoir comme conséquence la monopolisation des tokens par une application. Nous verrons que ce choix a une influence sur le modèle de programmation des applications décrit en section 7.3.

```
1 --->
2 GET /statemachines/1
3 --->
4
5 <----
6 Status code: 2.05 - Content
7 Body:
8     [...]
9     "current_state": {
10        "current_state_uri": "/currentState",
11        "state_name": "DOWN",
12        "state_uri": "/states/1"
13    },
14    [...]
15    "reachable_state": [ {
16        "state_name": "UP",
17        "stateUri": "/states/2"
18    } ]
19    "token_uri": "/token985240acvDfz"
20 <----
```

FIGURE 6.4 – Lecture de l'état de PR lorsque son verrou est disponible

L'exemple de code donné en figure 6.5 illustre une requête générant un token afin d'obtenir le verrou associé à l'entité contrôlable PR. La requête ayant été traitée avec succès (code de réponse à la ligne 6), un token unique est renvoyé à l'application via l'attribut "id_token" à la ligne 8.

```
1 --->
2 POST /statemachines/1/token985240acvDfz
3 --->
4
5 <---
6 Status code: 2.04 - Changed
7 Body:
8     "id_token": {  "abcdefg98765"  }
9 <----
```

FIGURE 6.5 – Requête d'obtention du token

Enfin, la figure 6.6 montre un exemple d'utilisation de ce token au travers d'une requête de changement d'état illustrée à la ligne numéro 2. L'URI de cette requête correspond à l'état cible "DOWN" tel que décrit dans le corps de la réponse de la figure 6.4.

```
1 PUT /statemachines/1/currentstate
2 Header:
3     id_token = abcdefg98765
4 Body:
5     "current_state": {  "state_name": "DOWN"  }
```

FIGURE 6.6 – Requête de changement d'état utilisant un token

6.3 Pertinence des opérations de recouvrement dans un système cyber-physique

Comme nous l'avons identifié en section 6.1.2 les applications doivent pouvoir effectuer des opérations globales via la coordination de plusieurs entités. Dans le domaine des bases de données, ce type d'opération globale est appelée transaction [66]. Nous pourrions nous en inspirer, cependant les propriétés ACID [72](Atomicité, Cohérence, Isolation, Durabilité) respectées par les transactions ne sont pas applicables à la nature d'un système cyber-physique, en particulier la propriété d'isolation (cf. section 6.2.1).

Par ailleurs, la coordination d'entités peut échouer à cause de divers problèmes : réseau temporairement indisponible, matériels défaillants, bugs logiciels. Afin d'éviter que le système soit dans un état instable suite à l'échec d'un processus de coordination, les mécanismes traditionnels de transactions utilisent une phase de recouvrement [67] afin de défaire, de refaire ou de compenser [60] les opérations précédemment réalisées. Dans notre contexte nous avons identifié que ces opéra-

tions n'avaient pas de sens compte tenu du système cyber-physique dans lequel est effectué un processus de coordination.

6.3.1 Relations entre éléments physiques

Dans un environnement physique, les éléments physiques ont des relations entre elles et agissent de manière indirecte les unes sur les autres. De ce fait, le contrôle d'une entité par une application peut agir de manière indirecte sur une seconde entité qui est supervisée par une autre application.

Exemple de relations Nous prenons l'exemple du feu de travaux FT_2 contrôlé par l'application App_1 et se situant sur la route R_2 (cf. figure 4.3 de la section 4.2), R_2 étant supervisée par l'application App_2.

La figure 6.7 illustre les relations (flèches pointillées) entre éléments physiques (rectangles) qui sont supervisés ou contrôlés par des applications (hexagones).

Ici, l'état du feu de travaux FT_2 a une influence directe sur les automobilistes à proximité qui, à leur tour ont une influence sur le taux d'occupation de la route R_2. Par exemple, lorsque App_1 change l'état de FT_2 à GREEN, les automobilistes se mettent à circuler et par conséquent cela modifie le taux d'occupation de la route R_2 supervisée par App_2.

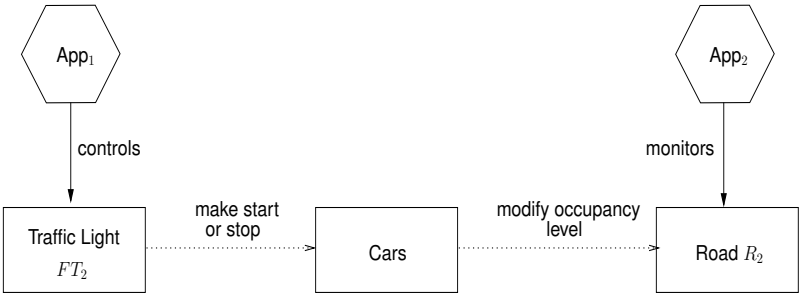


FIGURE 6.7 – Influence de l'application App_1 sur l'application App_2 via les diverses relations existantes entre le feu de travaux FT_2, les automobilistes et la route R_2

La nature des multiples relations existantes entre les éléments physiques peut être décrite dans une ontologie (cf. section 2.3). Sur la base de cette méthode, nous donnons en sous-section suivante, une idée d'exploitation que peuvent en faire les applications.

Propriété d'isolation de la coordination d'éléments physiques La propriété d'isolation désigne le fait que toutes actions effectuées au sein d'une transaction ne sont rendues visibles qu'à la fin de la transaction. Par exemple, lors d'une transaction entre deux comptes bancaires, les opérations de débit et de crédit des comptes ne sont visibles que lorsque la transaction est terminée.

Dans notre contexte, compte-tenu des relations qui existent entre les éléments physiques, un mécanisme de coordination opérant sur des éléments physiques ne

peut garantir la propriété d'isolation des actions effectuées. En effet, les effets des actions ne peuvent être rendus invisibles puisque le changement d'état d'une entité contrôlable produit des effets visibles dans l'environnement physique ayant des conséquences à la fois sur d'autres éléments physiques et sur d'autres applications.

6.3.2 Opérations de recouvrement

Pertinence des opérations de recouvrement Une opération de recouvrement ne peut pas défaire un changement d'état effectué précédemment, car comme décrit plus haut, ce dernier a déjà produit des effets et entraîné des conséquences dans l'environnement physique.

Par exemple, le feu de travaux FT_2 qui est passé dans l'état GREEN pourrait être remis dans l'état RED. Cependant, compte tenu des relations décrites en sous-section 6.3.1, le fait que FT_2 soit passé dans l'état GREEN a amené les conducteurs à circuler et à modifier le taux d'occupation de la route. Ces conséquences ne peuvent, elles, pas être défaites. Par conséquent, une opération de recouvrement n'est pas applicable dans notre contexte.

Pertinence des opérations de compensation Une opération de compensation n'a pas de sens dans notre contexte. Nous pourrions penser à utiliser l'état *fail-safe* d'une entité contrôlable (cf. section 5.3.3), si elle en possède un, pour compenser le changement d'état effectué. Cela reviendrait à affirmer que la compensation effectuée sur chaque entité contrôlable impliquée dans un processus de coordination équivaut à la compensation globale du processus de coordination. Cette affirmation est inexacte, en effet il est impossible de définir automatiquement la compensation globale d'un processus de coordination.

Les opérations de compensation sont propres aux objectifs qu'une application vise à atteindre lorsqu'elle coordonne des entités. Ces objectifs pouvant être différents d'une application à une autre, il n'est pas possible de définir automatiquement des opérations de compensation quelque soit l'application.

Par exemple, atteindre l'état *fail-safe* BLINKING du feu de travaux FT_2 aurait du sens pour une application de trafic routier. En effet, l'objectif de cette application est d'éviter l'apparition de bouchons. En revanche, atteindre l'état BLINKING n'a pas de sens pour une application qui vise à bloquer la circulation. En effet, afin de remplir l'objectif visé par cette application, la compensation devrait plutôt être de changer l'état de FT_2 en RED.

Informations aux applications lors d'un échec Un mécanisme de coordination doit laisser à l'application le choix d'effectuer les opérations de changement d'état qu'il semble, selon elle, raisonnable d'effectuer lors d'un échec.

Des informations doivent être données aux applications afin qu'elles puissent évaluer l'impact des conséquences de l'échec de la coordination. Puisque nous générons des ressources REST qui sont connectées entre elles, nous pourrions exploiter

la sémantique de leurs liens (cf. section 5.5.3). Par exemple, la sous-ressource associée à l'état vert de l'entité de FT_2 pourrait comporter un lien sémantique vers la ressource de l'entité de la route R_2 indiquant qu'il existe une influence sur l'augmentation taux d'occupation de la route. De ce fait, lorsque la coordination des feux FT_2 et FT_1 échoue et amène à ce que les deux feux soient verts, l'application serait notifiée des conséquences provoquées sur la route R_1 et R_2 et pourrait effectuer les choix qu'il lui paraît judicieux.

Il est à noter que l'exploitation de méta-données sémantiques est une des perspectives de notre travail (cf. section 8.2.1.1).

6.4 Coordination de plusieurs entités

6.4.1 Principes du mécanisme de coordination proposé

Afin d'illustrer les principes de notre mécanisme de coordination, nous prenons l'exemple de la coordination des feux de travaux FT_1 et FT_2 et du carrefour C. La coordination de ces entités contrôlables est utilisée afin de fluidifier la circulation sur l'axe routier, en changeant l'état de FT_1 à GREEN, l'état de FT_2 à RED et l'état de C à N-S RED E-W GREEN. Pour rappel, les figures 6.8 et 6.9 illustrent leur modèle d'entité respectif.

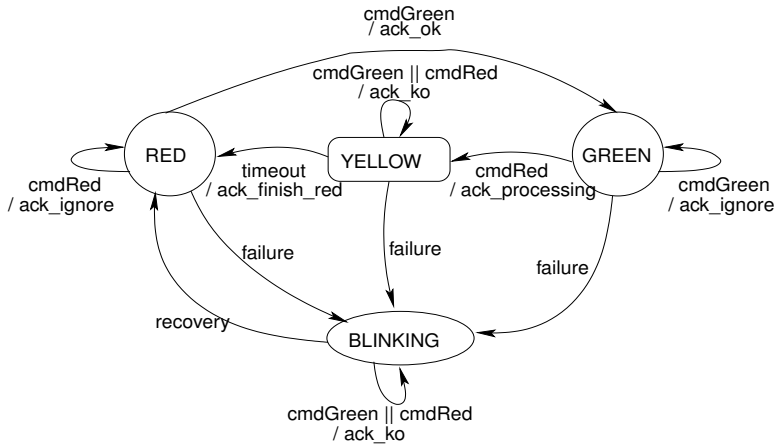


FIGURE 6.8 – Modèle d'entité d'un feu de travaux

Exécution d'un processus de coordination Nous proposons la création d'un coordinateur après réception d'une requête d'une application pour effectuer la coordination de plusieurs entités contrôlables. Un coordinateur est responsable d'un seul processus de coordination, les entités contrôlables ne peuvent être impliquées que dans un seul processus de coordination afin d'éviter les potentiels conflits entre plusieurs coordinateurs.

La requête d'une application pour créer un processus de coordination contient des couples $\{(E_1, etat_1), (E_2, etat_2), ..., (E_n, etat_n)\}$ dans lesquels un état cible

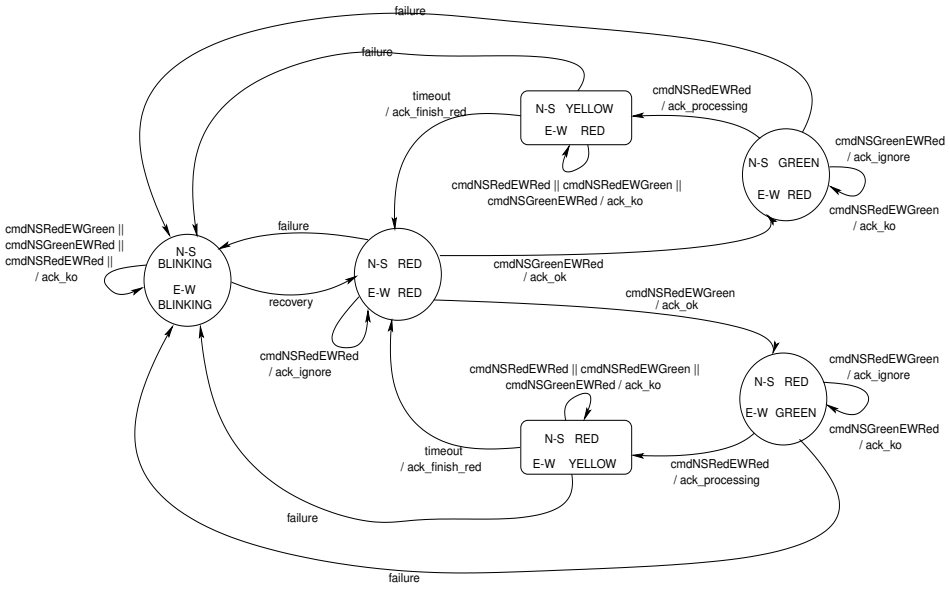


FIGURE 6.9 – Modèle d'entité du carrefour C

est associé à chaque entité contrôlable. Cette requête dans l'exemple serait $\{(FT_1, GREEN),\ (FT_2, RED),\ (C, N{-}S\ \ RED\ \ E{-}W\ \ GREEN)\}$. Par ailleurs, la position des couples dans la requête n'a pas d'importance dans l'exécution d'un processus de coordination.

Notre mécanisme de coordination permet seulement d'exécuter des séquences finies de changement d'états. Un seul changement d'état est associé à une seule entité contrôlable. Les mêmes principes pourraient être utilisés pour fournir des mécanismes plus sophistiqués, de la même façon que des orchestrateurs tel que BPEL [120] (Business Process Execution Language), pour exécuter des mini-programmes permettant d'enchaîner de diverses façons les opérations de changement d'état (cf. section 8.2.1.3).

Défaillance lors d'une coordination La défaillance d'une entité contrôlable au sein d'un processus de coordination est détectée par le coordinateur lorsque la réponse d'une entité contrôlable n'est pas reçue dans un temps imparti. La défaillance d'une entité contrôlable pouvant être due, par exemple, à une indisponibilité temporaire du réseau, du matériel ou à une panne du matériel.

Une défaillance du coordinateur peut également avoir lieu, mais cette dernière ne peut être palliée par l'algorithme de coordination que nous présentons en section 6.4.2. En revanche, ce problème peut être résolu en fonction du choix effectué concernant le déploiement du coordinateur dans une infrastructure distribuée (cf. section 7.4).

Atomicité de la coordination Un coordinateur doit assurer l'atomicité de la coordination en veillant à ce que le verrou V_e de chaque entité e impliquée ne soit

pas relâché avant la fin du processus de coordination. De ce fait, un coordinateur ne peut utiliser un verrou V_e expirant automatiquement.

D'une part cela pourrait engendrer des conflits puisqu'un V_e pourrait être relâché avant la fin de la coordination. Dans l'exemple, si le verrou de FT_2 est relâché avant la fin du processus de coordination, une application pourrait changer de nouveau l'état de FT_2 à GREEN et ainsi interrompre l'opération globale qui devait être accomplie par le coordinateur. D'autre part, la monopolisation d'une entité contrôlable n'est plus un problème puisqu'un coordinateur peut être supposé intrinsèquement collaboratif (il n'est pas écrit par un développeur d'application).

Par conséquent, nous avons choisi de laisser un coordinateur acquérir et relâcher un verrou V_e explicitement. Le verrou de FT_2 sera relâché lorsque toutes les entités (FT_1, C, FT_2) auront atteint leur état cible respectif. De façon similaire, les verrous de FT_1 et C seront également relâchés à ce moment-là.

Une entité contrôlable ne doit jamais rester verrouillée après la fin d'un processus de coordination même si des défaillances surviennent, telle qu'une indisponibilité momentanée du réseau. Le feu de travaux FT_1 devrait pouvoir être remis dans l'état RED par une application qui le souhaite, à partir du moment où il n'est plus impliqué dans un processus de coordination et que la coordination est donc terminée.

Pour empêcher qu'une entité contrôlable reste indéfiniment verrouillée après la fin d'un processus de coordination, nous proposons que chaque entité contrôlable impliquée au sein d'une coordination stocke l'identité du coordinateur pour pouvoir le recontacter après un délai raisonnable pour demander à être déverrouillée.

Résultat d'une coordination Le coordinateur envoie le résultat du processus de coordination à l'application qui l'a créé. Cette réponse contient les couples $\{(E_1, etat_1, statut_1), (E_2, etat_2, statut_2), ...(E_n, etat_n, statut_n)\}$. Le statut permet d'indiquer à l'application si l'état d'une entité a été changé, est resté tel quel, ou n'a pas pu être changé à cause d'une défaillance. Cela permet à l'application d'effectuer les actions appropriées.

Si FT_2 n'a pas répondu à une requête du coordinateur, conduisant le processus de coordination à échouer, et que FT_1 a atteint son état cible mais que l'entité C n'a elle pas changé d'état, alors le résultat envoyé par le coordinateur à l'application est le suivant $\{(FT_1, GREEN, changed), (FT_2, RED, failed), (C, N\text{–}S\ RED\ \ E\text{–}W\ GREEN, remained)\}$.

Lecture pendant une coordination active Nous autorisons les lectures dites "fantômes" des états d'une entité impliquée dans un processus de coordination, pour des raisons analogues à celles décrites en section 6.1.2. Les applications sont notifiées que le verrou est actuellement détenu et peuvent juger de la pertinence de la lecture. L'information concernant la provenance du détenteur du verrou, une application ou un coordinateur, n'est pas communiquée à l'application puisque cette information ne donne aucune garantie sur le moment où sera relâché le verrou.

6.4.2 Algorithme de coordination

La figure 6.10 est un diagramme d'activité illustrant les cinq étapes successives effectuées par un coordinateur pour coordonner des entités contrôlables.

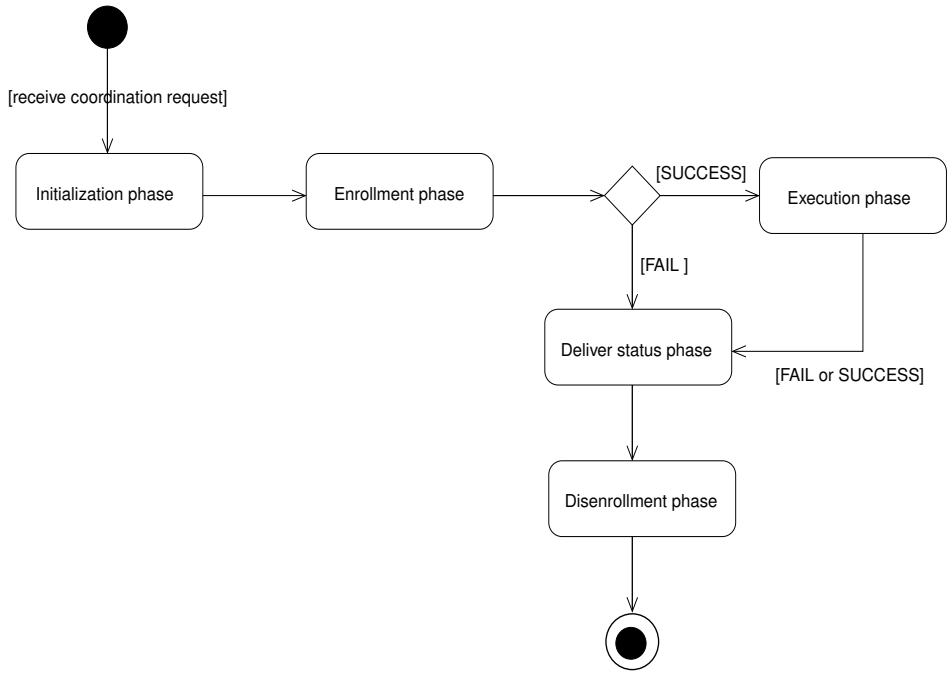


FIGURE 6.10 – Diagramme d'activité d'un processus de coordination

Phase d'initialisation Un coordinateur initialise le processus de coordination lorsqu'il reçoit une requête d'une application comme le montre la transition partant de l'état initial. Comme décrit en section 6.4.1, cette requête contient les couples $\{(E_1, etat_1), (E_2, etat_2), ..., (E_n, etat_n)\}$ où un état cible $etat_i$ est associé à chaque entité E_i impliquée dans le processus de coordination.

Lors de la phase d'initialisation, le coordinateur génère un identifiant unique *cid* (*coordinator unique identifier*) qui sera utilisé dans chaque requête envoyée au cours des phases suivantes. Lorsque le coordinateur atteint de nouveau la phase d'initialisation, il détruit le *cid* utilisé. Une fois le *cid* généré, la phase d'enrôlement commence.

Phase d'enrôlement Le coordinateur envoie une requête d'enrôlement à chaque E_i et arme pour chaque requête R_i un timer t_i. Chaque requête R_i interroge l'entité E_i afin de savoir si elle est en mesure d'atteindre l'état $etat_i$ en fonction de son état courant. Si aucune réponse n'est reçue avant l'expiration de t_i, le coordinateur considère la réponse de E_i comme négative.

Si E_i déclare pouvoir atteindre $etat_i$, alors le verrou V_{E_i} est automatiquement obtenu. E_i stocke le *cid* du coordinateur qui devient alors le détenteur du verrou,

et envoie une réponse positive à ce dernier. Comme représenté par la transition SUCCESS, la phase d'enrôlement se termine avec succès lorsque le coordinateur reçoit des réponses positives de l'ensemble des entités contrôlables impliquées.

Si E_i ne peut pas atteindre $etat_i$, ou si E_i est déjà verrouillée par une application ou un coordinateur, elle envoie une réponse négative au coordinateur. Comme illustré par la transition FAIL, la phase d'enrôlement échoue dès lors qu'au moins une entité contrôlable ne peut être enrôlée (lorsqu'un t_i expire ou qu'une réponse négative est reçue). Le coordinateur démarre alors la phase d'envoi correspondant à l'envoi du statut de la coordination à l'application initiatrice.

Phase d'exécution Après le succès de la phase d'enrôlement, le coordinateur commence la phase d'exécution. Il envoie une requête d'exécution de changement d'état à chaque E_i et arme un timer t'_i associé à chaque requête. Dès l'expiration d'un t'_i, la phase d'exécution est annulée.

Lorsque E_i reçoit une requête d'exécution, elle vérifie qu'il s'agit du détenteur du verrou via le *cid* envoyé dans la requête. S'il s'agit du détenteur du verrou, E_i procède au changement d'état et, une fois effectué, envoie une réponse positive au coordinateur. Comme illustré par la transition SUCCESS, la phase d'exécution se termine avec succès lorsque le coordinateur a reçu les réponses positives de toutes les entités E_i.

Comme illustré par la transition FAIL, la phase d'exécution échoue dès lors qu'un des timers t'_i expire. La phase d'exécution ne peut échouer autrement puisque l'entité est déjà enrôlée et donc prête à atteindre l'état cible. Lorsque la phase d'exécution échoue, le coordinateur arrête l'envoi des requêtes d'exécution.

Que la phase d'exécution ait été effectuée avec succès ou non, le coordinateur démarre la phase suivante qui correspond à l'envoi du statut de la coordination à l'application initiatrice.

Phase d'envoi du statut Cette phase correspond à l'envoi d'un statut négatif ou positif à l'application initiatrice du processus de coordination afin de lui indiquer le résultat de celui-ci. Un statut négatif est envoyé à l'application si le coordinateur est entré dans cette phase lorsque la phase d'enrôlement ou d'exécution a échoué. A l'inverse il est positif si le coordinateur a effectué avec succès la phase d'exécution.

Une fois le statut envoyé à l'application, le coordinateur démarre la phase de désenrôlement.

Phase de désenrôlement Le coordinateur envoie une requête de désenrôlement à chaque E_i afin de les désenrôler et de libérer leur V_{E_i} respectif. Comme illustré par la flèche END, une fois ces requêtes envoyées, le coordinateur retourne dans sa phase d'initialisation en attendant une nouvelle requête d'une application pour débuter de nouveau un processus de coordination.

6.4.3 Validation via l'outil de model-checking Spin

6.4.3.1 Principes de la validation effectuée

Objectifs Afin de valider notre algorithme via l'outil de model-checking Spin [79], nous avons choisi de modéliser les éléments suivants au sein d'un programme Promela : deux entités contrôlables notées E_1 et E_2, deux coordinateurs notés c_1 et c_2 ainsi que deux fragments d'application notés App_1 et App_2. Les interactions entre les différents éléments sont les suivantes : c_1 et c_2 coordonnent E_1 et E_2 et les deux applications App_1 et App_2 envoient des requêtes de changement d'état à E_1.

Ce choix a été fait afin d'observer toutes les configurations d'interactions possibles auxquelles sont sujettes les entités contrôlables. L'observation des multiples exécutions des coordinateurs et des applications permet de montrer que le processus de coordination ne produit pas d'interblocage ou de conflits sur les entités contrôlables. De même, cela permet de vérifier qu'une entité ne reste jamais indéfiniment verrouillée (et donc inutilisable) même lorsque des pertes de paquets surviennent sur le réseau. De manière générale, cela permet de vérifier que le comportement des différents éléments est cohérent, par exemple si un coordinateur est en phase d'exécution toutes les entités sont bien enrôlées.

Modélisation des processus et des communications On associe à chaque élément modélisé un "active process" défini dans Promela : cela permet à un élément d'avoir son propre process qui lui permet de s'exécuter en parallèle de façon asynchrone. Chaque élément communique, tel que App_1 avec E_1, via des canaux FIFO. Un seul canal est associé à chaque élément par le biais duquel toutes les requêtes sont reçues.

Les pertes potentielles de messages durant la transmission sont modélisées du côté de l'émetteur : l'envoi d'un message consiste en un choix non-déterministe entre l'envoi effectif du message ou la perte de celui-ci. Dans les deux cas, l'émetteur considère que le message a été émis.

Timeouts Nous ne modélisons pas d'une manière quantitative les timeouts. En effet, nous utilisons le mot-clé "timeout" de Promela qui est une condition qui devient vraie lorsqu'aucune transition n'est possible dans la totalité du système. Dans notre cas, ce *timeout* est une protection qui permet à un process de ne pas être bloqué indéfiniment dans un état.

Dans un système réel, cette condition représente un timer dont la durée serait choisie avec justesse lors de l'émission d'une requête. Cette durée n'étant ni trop courte afin de ne pas manquer la réponse, ni trop grande afin de ne pas rester inutilement en attente d'une réponse.

Expiration d'un token Nous utilisons une transition pour simuler la destruction d'un token lorsque la période de validité du token expire. Ce choix s'explique par le fait qu'il n'est pas possible de définir du temps de manière quantitative dans Pro-

mela. Par conséquent, nous nous affranchissons d'une durée quantitative via cette transition qui représente le moment lors duquel a lieu l'expiration du token.

Cette transition nous permet de faciliter la validation du comportement d'une entité, puisque cette transition peut être activée à tout moment après génération du token. Elle permet également de tester plusieurs cas d'usage, comme nous le décrivons dans la sous-section suivante.

6.4.3.2 Automates des éléments

Les éléments Promela sont représentés par des automates décrivant leur comportement. Chaque transition de ces automates est atomique, ceci étantt rendu possible via l'utilisation du mot-clé "atomic" de Promela. Les figures 6.13, 6.11 et 6.12 représentent respectivement les automates du fragment de App_1 et App_2, des coordinateurs c_1 et c_2 et des entités contrôlables E_1 et E_2. Chaque transition est labellisée avec le formalisme Promela : ! désigne l'envoi d'une requête et ? désigne la réception d'une réponse. Afin d'améliorer la lisibilité des figures, nous utilisons les abréviations suivantes :

- !exe(E_i) : envoi d'une requête d'exécution de changement d'état à E_i.
- ?ok_exe(E_i) : réception d'une réponse positive d'une requête d'exécution, envoyée par E_i
- ?ko_exe(E_i) : réception d'une réponse négative envoyée par E_i, à une requête d'exécution. Celle-ci indique que l'état cible n'est pas atteignable.
- ?ko2_exe(E_i) : réception d'une réponse négative envoyée par E_i, à une requête d'exécution. Celle-ci indique que le token n'est plus valide bien que l'état cible soit potentiellement atteignable.
- !enr(E_i) : envoi d'une requête d'enrôlement à destination de E_i
- ?ko_enr(E_i) : réception d'une réponse négative d'enrôlement envoyée par E_i
- ?ok_enr(E_i) : réception d'une réponse positive d'enrôlement envoyée par E_i
- !tok(E_i) : envoi d'une requête de demande de verrou à E_i.
- ?ko_tok(E_i) : réception d'une réponse négative à une demande de verrou, envoyée par E_i.
- ?ok_tok(E_i) : réception d'une réponse positive à une demande de verrou, envoyée par E_i.
- !dis(E_i) : envoi d'une requête de désenrôlement à E_i.
- tim_exe(E_i) : expiration du timer associé à une précédente requête d'exécution. De la même façon tim_enr(E_i) désigne l'expiration du timer associé à une précédente requête d'enrôlement.
- tok_expired : expiration du token précédemment généré.

Automate des coordinateurs La figure 6.11 illustre le processus de coordination décrit en section 6.3.1. La requête provenant d'une application n'est pas modélisée dans l'automate, puisque la validation porte sur l'algorithme de coordination. Par conséquent, l'hypothèse est faite que lorsque l'automate de la figure 6.11 est dans l'état 0, une requête d'une application pour coordonner E_1 et E_2 a déjà été reçue.

L'état 0 correspond à la phase d'initialisation, où est généré le *cid* qui est envoyé dans les requêtes suivantes du processus. Les états 1 à 4 représentent la phase d'enrôlement dans laquelle les requêtes d'enrôlement sont émises et leurs réponses reçues par le coordinateur. Cette phase échoue lorsque le coordinateur atteint l'état 10 suite à l'échec d'enrôlement d'une ou des deux entités. A l'inverse, cette phase est accomplie avec succès lorsque le coordinateur atteint l'état 5 suite à la réception d'une réponse positive de la part de chaque entité.

Les états 6 à 7 correspondent à la phase d'exécution dans laquelle les requêtes d'exécution sont émises. Quelque soit le statut de cette phase où les requêtes peuvent aboutir à un changement d'état des entités ou expirer, le coordinateur atteint l'état 10.

Les états 10 et 11 représentent la phase de désenrôlement où les requêtes de désenrôlement sont envoyées à destination de E_1 et E_2. Cette phase conduit le coordinateur à retourner dans l'état 0 et à pouvoir démarrer une nouvelle fois un processus de coordination. Bien que non représentées sur la figure, les réponses reçues d'une précédente coordination sont simplement ignorées par le coordinateur en contrôlant le *cid*.

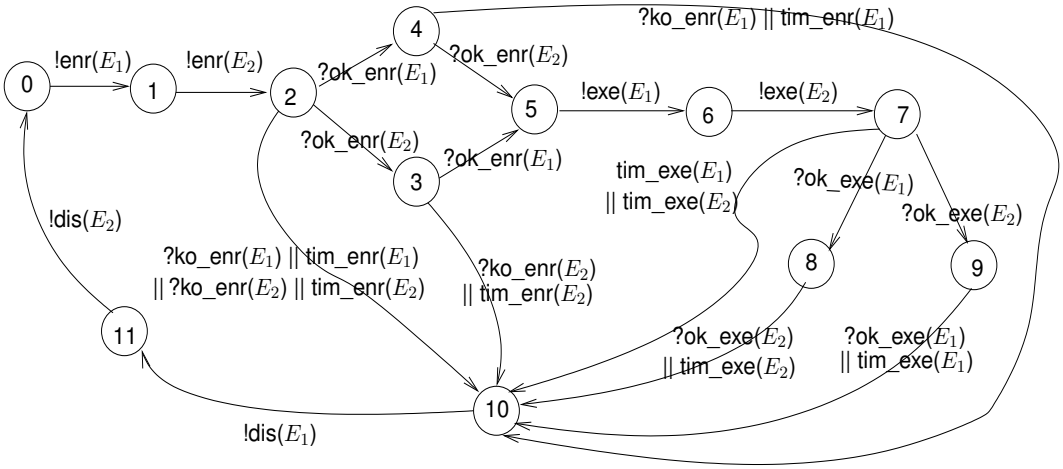


FIGURE 6.11 – Modélisation Promela des coordinateurs c_1 et c_2. || représente la disjonction booléenne

Automate des entités contrôlables La figure 6.12 illustre l'automate Promela des entités contrôlables E_1 et E_2, qui est composé de trois parties distinctes et indépendantes les unes des autres, décrites ci-dessous.

Génération d'un token Les états 5 à 6 représentent la réception d'une requête d'obtention de token ainsi que la génération de ce dernier qui conduit une application à le détenir. Les états 7 à 8 illustrent les requêtes d'exécution de l'application détentrice du token ainsi que leurs réponses associées émises, pouvant être positives ou négatives si l'état est atteignable ou non.

Comme l'expiration d'un token peut survenir à tout moment après qu'il ait été généré, l'état 0 peut être atteint depuis les états 7 et 8 (via la transition tok_expired). Le verrou est donc relâché lorsque l'état 0 est atteint.

Enrôlement dans un processus de coordination Les états 1 et 2 représentent l'enrôlement de l'entité amenant un coordinateur à détenir le verrou. Les états 3 et 4 montrent la réception d'une requête d'exécution provenant du coordinateur détenant le verrou ainsi que la réponse positive qui lui est retournée. Puisqu'un coordinateur peut envoyer à tout moment une requête de désenrôlement à une entité contrôlable, à cause de l'échec de la phase d'enrôlement, les états 1 à 4 ont une transition portant le label $?dis$ vers l'état 0 conduisant l'entité à être déverrouillée. Les requêtes de désenrôlement provenant d'un précédent processus de coordination sont simplement ignorées telles qu'illustrées avec les transitions self-loop sur les différents états de l'automate.

Cas d'erreurs La réception de requêtes (illustrées en pointillés) provient d'applications ou de coordinateurs qui ne sont pas détenteurs du verrou. Les états 23, 43, 63, 83 et 103 illustrent la réception d'une requête de changement émise par une application qui n'a plus de token et qui recevra alors une réponse négative le lui indiquant. Ces requêtes pouveut être reçues à tout moment lorsque l'entité contrôlable a généré un token ou se trouve dans un processus de coordination.

Comme un token n'est pas renouvelable, les états 62 et 82 illustrent le cas où une requête de demande de token est reçue et où une réponse négative lui est renvoyée. Ces deux états ainsi que les états 22 et 42 permettent également d'illustrer que lorsqu'une application non-détentrice du verrou envoie une requête, l'entité lui répond négativement puisqu'elle est déjà verrouillée par un tiers.

Pour des raisons analogues, représentées par les états 21, 41, 61 et 81, une réponse négative est envoyée au coordinateur qui émet une requête d'enrôlement alors que l'entité est déjà verrouillée.

Automate des applications La figure 6.13 représente un fragment des applications App_1 et App_2. Celles-ci envoient une requête pour obtenir un token tel qu'illustré par l'état 1. Lorsqu'un token est reçu (état 2), une requête d'exécution est envoyée vers E_1 (état 3). Si une application reçoit une réponse lui indiquant que son token n'est plus valide (transition $!ko2\_exec(E_1)$), elle atteint l'état 0 et recommence la totalité du workflow. Si une application reçoit tout autre réponse (positive pour $ok\_exe(E1)$ ou négative pour $ko\_exe(E1)$) ou si le timer lié à cette requête expire ($tim\_exe(E1)$), elle recommence l'envoi d'exécution en atteignant de nouveau l'état 2.

6.4.3.3 Validation

Nous avons utilisé le logiciel Spin en mode interactif pour simuler les comportements des éléments modélisés. Nous avons exécuté une simulation comprenant 10000

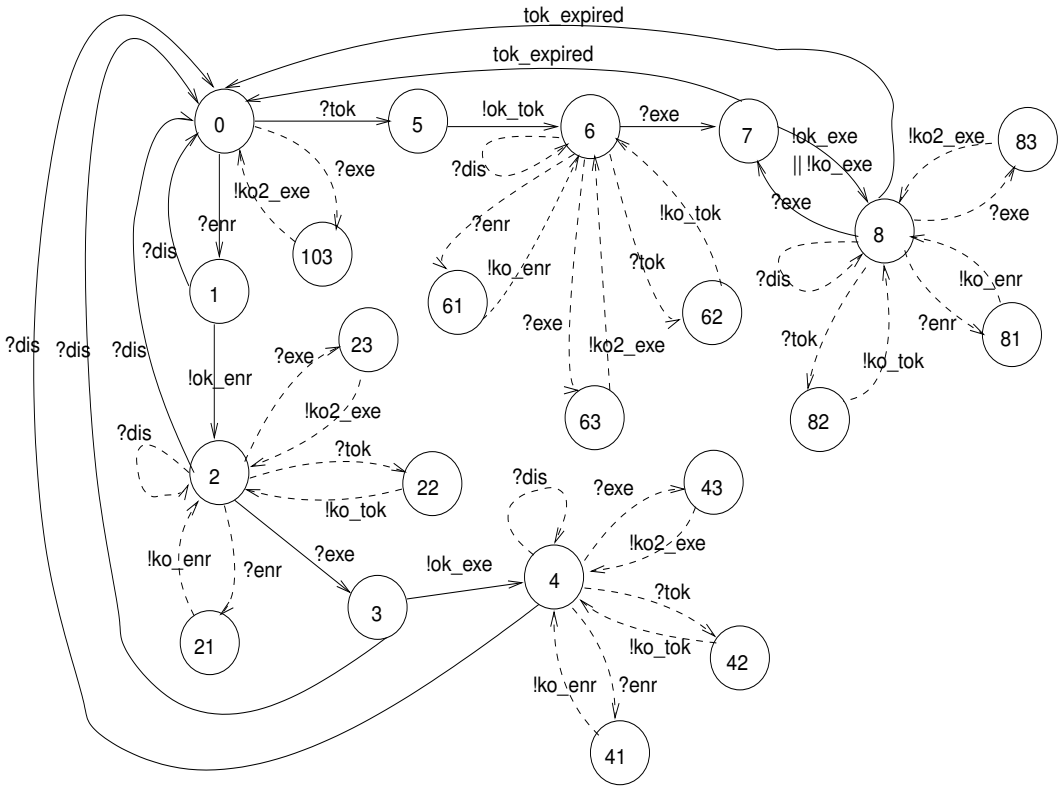


FIGURE 6.12 – Modélisation Promela des entités contrôlables E_1 et E_2. Les flèches pointillées représentent les requêtes (et leurs réponses associées) provenant d'une application ou d'un coordinateur n'étant pas le détenteur du verrou.

pas aléatoirement choisis permettant un choix non-déterministe des transitions des automates. Ces 10 000 pas représentent approximativement 2500 changements d'état sur la totalité des automates. Cela nous a permis d'identifier certains bugs et nous a permis d'observer que notre algorithme ne comportait pas de deadlock, même en cas de perte de messages.

De plus, nous avons observé qu'un élément qui envoie une requête (application ou coordinateur) reçoit toujours une réponse qui est cohérente avec le comportement actuel de l'entité contrôlable réceptrice. Par exemple, une application recevra toujours une réponse négative lorsqu'il souhaite obtenir le token d'une entité qui est enrôlée.

Le cas validé comprenant deux applications, coordinateurs et ressources est représentatif du cas général bien que notre validation ne soit pas généralisable. Par exemple, une application peut tenter d'entrer un conflit avec une application ou un coordinateur, peu importe le nombre existant de coordinateurs et d'applications. De la même manière, une entité contrôlable enrôlée peut recevoir 1 ou n requêtes de la part d'autres applications ou d'autres coordinateurs, quel que soit le nombre d'applications ou de coordinateurs.

Il est possible de formellement généraliser cette validation, tel qu'effectué pour

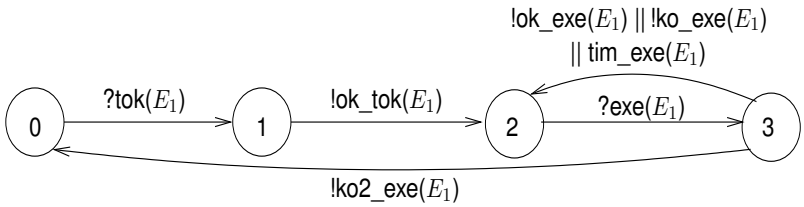


FIGURE 6.13 – Modélisation Promela du fragment des applications App_1 et App_2.

d'autres protocoles [18], à l'aide de propriétés LTL. Il s'agit de l'une des perspectives de notre travail (cf. section 8.2.1.3).

6.5 Conclusion

Dans ce chapitre, nous avons identifié les exigences que soulève le contrôle partagé d'entités puis proposé des mécanismes pour répondre à ces exigences.

La première des exigences était de garantir aux applications que l'état cible de leur requête soit atteint dès lors que celle-ci est acceptée, empêchant ainsi que des conflits surviennent. Pour cela, nous avons proposé un mécanisme d'exclusion mutuelle, basé sur des verrous, où un verrou est associé à chaque entité contrôlable. Les applications pouvant potentiellement monopoliser ces verrous, nous avons mis en place une période de validité associée à chaque verrou. Cette période de validité implique l'expiration du verrou après un certains laps de temps et permet une utilisation équitable des verrous, de la part des applications.

Pour permettre le contrôle atomique de plusieurs entités, non reliées physiquement de manière sous-jacente, mais également pour favoriser l'utilisation opportuniste des entités, nous avons identifié qu'un mécanisme de coordination était nécessaire. Compte-tenu de la nature du système cyber-physique dans lequel nous évoluons, nous avons montré qu'une phase de recouvrement n'a pas de sens au sein d'un mécanisme de coordination. La compensation ne peut pas être définie automatiquement et le recouvrement n'est pas applicable puisqu'un changement d'état d'une entité affecte de manière indirecte d'autres entités.

En tenant compte de ces conclusions, nous avons proposé un algorithme de coordination permettant d'effectuer des séquences de changement d'états. Afin de vérifier que le comportement de notre algorithme était celui attendu, nous l'avons validé via l'outil de vérification SPIN. Pour cela, nous avons modélisé sous forme d'automates des entités, des applications ainsi que des coordinateurs et exécuté manuellement diverses simulations en analysant celles-ci pour vérifier le comportement global.

Chapitre 7

Implémentation d'une preuve de concept

Dans ce chapitre, nous décrivons l'implémentation d'une preuve de concept autour de l'exemple que nous avons élaboré (cf. section 4.2).

Nous commençons par présenter ce que comporte notre implémentation en définissant le périmètre de notre preuve de concept (cf. section 7.1.1) qui comprend les mécanismes de contrôle proposés, un ensemble d'entités, des applications ainsi qu'un simulateur des capteurs réseaux et des actionneurs. Cela nous permet de décrire les objectifs que nous souhaitons atteindre au travers du travail technique réalisé (cf. section 7.1.2).

Nous détaillons les divers choix techniques que nous avons effectués pour réaliser notre implémentation (cf. section 7.2.1). Puis, nous présentons notre implémentation des modèles d'entités et de leurs ressources REST (cf. section 7.2.2). Afin d'exécuter les modèles d'entités créés, nous simulons les capteurs et les actionneurs qu'ils utilisent (cf. section 7.2.3). Nous décrivons également notre implémentation des divers mécanismes de contrôle (cf. section 7.2.4) ainsi que les différentes applications de démonstration créées (cf. section 7.2.5).

Afin de superviser et contrôler les entités, nous présentons un modèle de programmation que les développeurs doivent suivre ainsi que différentes recommandations à propos des actions qu'elles doivent entreprendre (cf. section 7.3).

Enfin, nous proposons des recommandations à propos du déploiement de notre implémentation sur une infrastructure distribuée précédemment identifiée (cf. section 4.1.4). Ces recommandations concernent le déploiement du code des entités (cf. section 7.4.1), le mécanisme de coordination (cf. section 7.4.2) ainsi que le code des applications (cf. section 7.4.3).

Sommaire

7.1 Définition de l'implémentation réalisée

7.1.1 Périmètre de la preuve de concept

Nous proposons une implémentation des modèles d'entités (cf. chapitre 5), des mécanismes de contrôle de concurrence et de coordination et des applications interagissant avec différentes entités. Nous avons créé un simulateur afin de générer les valeurs des capteurs, les unités de temps physiques écoulées ainsi que les latences réseaux des commandes envoyées aux actionneurs et leurs défaillances.

Notre implémentation des entités et des applications est une preuve de concept qui se base sur l'exemple typique du carrefour (cf. section 4.2). Nous avons conçu les modèles d'entités liés aux éléments physiques présents dans cet exemple. Les différentes données des capteurs présentes dans cet exemple sont simulées. Nous injectons également des pannes et simulons les latences réseaux liées à des commandes envoyées aux actionneurs. Enfin, toujours sur la base de notre exemple, nous avons implémenté des applications de démonstration qui supervisent et contrôlent ces entités.

7.1.2 Objectifs de l'implémentation

Notre implémentation vise à illustrer trois objectifs : permettre l'intégration de nouveaux capteurs et actionneurs, permettre l'intégration de nouvelles applications et présenter le modèle de programmation que les développeurs d'applications doivent adopter.

Concernant les nouvelles applications, notre objectif est de montrer comment elles doivent interagir avec la plateforme pour superviser et pour contrôler la ville intelligente. Pour cela, elles doivent tenir compte des mécanismes de contrôle de concurrence et de coordination.

Dans un second temps et de manière plus générale, les applications doivent adopter un modèle de programmation en tenant compte des divers codes de réponse qui leur sont retournés.

Concernant les nouveaux capteurs et/ou actionneurs, notre objectif est de montrer la démarche qu'un nouvel opérateur doit suivre pour pouvoir les intégrer une infrastructure déjà gérée par notre plateforme.

Nous ne mettons pas l'accent sur les choix technologiques effectués dans notre implémentation. Ces choix technologiques sont suffisants pour notre preuve de concept et nous permettent de proposer une implémentation prête à être intégrée dans la plateforme fédératrice FIWARE. Les objectifs de notre implémentation sont d'ordre qualitatif, à savoir qu'ils ne peuvent être mesurés par des métriques. De ce fait, il ne présente pas d'intérêt d'évaluer quantitativement notre travail, par exemple avec des métriques de performance.

7.2 Conception de la preuve de concept

La preuve de concept a été réalisée en langage Java en utilisant l'environnement de développement intégré Eclipse et Maven3. Le nombre total de lignes de code écrites est de 7000 lignes de code, d'après notre analyse faite au moyen de commandes bash et d'expressions régulières.

7.2.1 Choix technologiques

Le choix du protocole entre les applications et les ressources doit remplir deux critères, à savoir être compatible avec une architecture REST et permettre l'envoi de notifications. De ce fait, nous avons choisi le protocole CoAP [152]. Celui-ci remplit ces deux critères puisqu'il s'agit d'un protocole très similaire à HTTP (cf. section 2.1.2) et présente l'avantage d'avoir un verbe supplémentaire OBSERVE [78] permettant aux clients de s'abonner à des ressources REST dont la représentation change.

Nous avons utilisé l'implémentation open-source Eclipse Californium [46] qui est écrite en Java pour la création de serveurs et d'applications. Californium a été initialement développé par des chercheurs de l'école polytechnique fédérale de Zurich [1]. Désormais, cette implémentation fait partie d'un ensemble de projets open-source, appartenant à la fondation Eclipse [56], relatifs au domaine de l'IoT. La raison de notre choix concernant Californium est qu'il s'agit de l'implémentation la plus aboutie, en langage Java, pour créer des clients et des serveurs basés sur CoAP tel que décrit sur le site dédié à ce protocole [2].

Afin de visualiser et d'interagir avec les représentations des ressources que nous créons, nous avons utilisé l'outil Copper [92]. Il s'agit d'un plugin disponible sous le navigateur web Firefox qui permet d'interagir avec des ressources REST en utilisant le protocole CoAP.

Pour sérialiser les échanges entre les applications et le serveur CoAP, nous avons opté pour le format JSON [30]. Ce choix s'explique car JSON est un format communément utilisé compte tenu de sa structure facile à appréhender par des développeurs d'application. Ce format n'intègre pas d'éléments de syntaxe pour annoter sémantiquement les divers champs en les référençant au vocabulaire d'ontologie. Cependant, nous avons identifié que le format JSON-LD [162] (Linked Data), qui comprend une

1. http://people.inf.ethz.ch/mkovatsc/californium.php
2. http://coap.technology/impls.html

syntaxe étendue de JSON, permet de satisfaire ce critère. Similairement, puisque JSON n'est pas un format hypermédia, car il ne possède pas d'éléments de syntaxe décrivant des liens hypermédia, nous avons identifié que le format hypermédia Hydra [94] était un choix idéal. Il est à noter qu'utiliser le format hypermédia Hydra est compatible avec l'utilisation du format JSON-LD, puisque Hydra se base sur JSON-LD. Nous donnerons quelques détails à propos de ces formats dans les perspectives de notre travail (cf. section 8.2.1.1).

Enfin, la librairie Java nommée google-gson [65] a été utilisée afin de sérialiser et désérialiser les modèles Java vers le format JSON (vice-versa). Le choix de cette librairie est motivé par le fait que celle-ci est facile d'utilisation.

7.2.2 Implémentation des modèles

7.2.2.1 API des modèles à états finis

Création des modèles L'API que nous avons conçue permet de créer des automates à états finis et d'y définir leurs entrées, sorties, variables, états et transitions. Notre API n'est pas restreinte à créer un modèle d'entité spécifique.

Entrées Chaque entrée possède un nom unique, via lequel elle est identifiée ainsi qu'une valeur. Une entrée peut être définie comme étant liée à une valeur de capteur, une interface devra être implémentée pour obtenir celle-ci. Une entrée qui représente une unité de temps physique (top) est automatiquement ajoutée à un automate lorsque celui a été défini comme comportant au moins un état temporisé. Enfin, les entrées utilisées par les applications sont créées automatiquement lors de l'ajout de transitions entre deux états.

Etats et variables Les états ont un nom unique et peuvent être optionnellement définis comme étant intermédiaires et temporisés. L'ordre de création des états est important. Lors de la création d'un état intermédiaire, l'état atteint par cet état intermédiaire doit être précisé. De plus, une interface doit être implémentée et décrire les conditions (c'est-à-dire la transition) permettant d'atteindre cet état, via l'état intermédiaire. Le programmeur doit également spécifier le nombre d'unités de temps liées à un état temporisé.

Enfin, les variables ont un nom unique et doivent être définies comme étant commandables ou non. Lorsqu'une variable est définie comme étant commandable par les applications, une entrée utilisée par les applications est créée.

Transitions Suite à l'ajout d'états et de variables, les transitions doivent être définies. Celles-ci comportent une ou plusieurs conditions ainsi qu'une sortie liée à la mise à jour de variables. Les conditions d'une transition et la sortie rattachée à celle-ci doivent être définies au moyen d'une interface.

Des transitions utilisées lors des requêtes de changement d'état peuvent également être définies. Leur sortie correspond à une interface qui doit être implémentée

afin d'envoyer une ou plusieurs commandes à un ou plusieurs actionneurs. Une entrée est associée automatiquement à ces transitions qui sont elles-mêmes créées si elles n'existent pas.

Dans l'exemple, une fois les entrées, états et variables du modèle du plot PR créés, une transition utilisée par les applications est définie entre l'état UP et l'état MOVING DOWN. Cette transition a automatiquement pour condition, l'entrée "cmdDown==true" nouvellement créée. La sortie de cette transition permettant d'abaisser le plot PR doit être implémentée par du code.

Ce type de transition peut être attaché à une variable. Ici, les conditions doivent être définies comme étant celles permettant la mise à jour de la variable, et la sortie de cette transition via une interface à implémenter pour envoyer la ou les commande(s). Par exemple, la transition liée à la variable message de l'automate du panneau PAD a pour condition une longueur de chaîne de caractères à ne pas dépasser. Si cette condition est respectée, le message sera mis à jour via le code de l'interface implémentée.

Finalisation du modèle La finalisation du modèle doit être effectuée de manière explicite via l'API.

Dans un premier temps, des transitions "self" (sortante et entrante d'un même état) sont ajoutées automatiquement à certains états du modèle. Les états cibles de cet ajout sont ceux manquants d'une transition sortante ayant pour condition une ou plusieurs entrées utilisées par les applications. Dans l'exemple, les états MOVING DOWN et DOWN du modèle de PR n'ont pas pas de transition sortante ayant pour condition l'entrée cmdDown. De même pour les états MOVING UP et UP et l'entrée cmdUp. Ainsi, une transition "self" comportant l'entrée en question sera ajoutée à chacun de ces états.

Une fois ces nouvelles transitions créées, les acquittements à envoyer aux applications sont inférés à partir de la structure de l'automate (cf. section 5.3) en les ajoutant aux sorties des différentes transitions.

Exécution des modèles L'API conçue permet d'exécuter des automates de manière générique. Nous en illustrons le principe général par le pseudo-code présent dans la figure 7.1.

```
1  current state = init
2  while (true) {
3    get inputs
4    if (condition satisfied on current state available transition) {
5      actuator error = set output to actuators
6      if(!actuator error)
7        modify current state
8      set output to application
9  } }
```

FIGURE 7.1 – Étapes suivies lors de l'exécution des automates à états finis

Chaque automate commence par initialiser son état (ligne 1) puis s'exécute dans une boucle infinie (ligne 2). L'automate commence par lire les entrées disponibles (ligne 3) : données de capteurs, entrées utilisées par des applications ou temps écoulé. Par exemple, si l'état de l'automate relatif au plot PR est MOVING DOWN, on pourra obtenir l'entrée permettant de relever le plot ou la valeur du capteur mesurant la hauteur du plot.

Lorsque la condition d'une transition de l'état courant est satisfaite pour cette entrée (ligne 4), les sorties associées à cette transition sont exécutées. Des commandes sont envoyées aux actionneurs lorsque nécessaire (ligne 5) et une récupération générique des erreurs liées à cet envoi est effectuée. Si les commandes ont été effectuées avec succès, l'état courant est modifié (ligne 7). Enfin, un acquittement est retourné à l'application pour l'informer du statut de sa requête (ligne 8). Si une erreur s'est produite lors de l'envoi des commandes aux actionneurs (obtenue en ligne 5), un acquittement générique d'erreur est renvoyé à l'application.

7.2.2.2 Bibliothèque de modèles d'entités

Afin de permettre à un opérateur d'intégrer ses capteurs et actionneurs à notre plateforme, nous avons développé une bibliothèque de modèles d'entités se basant sur l'API précédente. L'implémentation actuelle de cette bibliothèque comporte les modèles d'entités suivants : un plot rétractable, une route, un carrefour, un panneau digital, un feu tricolore, des lampadaires d'une rue et un ensemble de places de parking. A titre d'exemple, le modèle d'entité d'une route de cette bibliothèque a été créé via l'API de création. Il comporte trois états (LOW TRAFFIC, MEDIUM TRAFFIC, HIGH TRAFFIC), des variables non-exposées aux applications (nbCars, MEDIUM, HIGH) et des transitions (avec hystérésis, prenant en compte nbCars ainsi que MEDIUM et HIGH). Ce modèle comporte des entrées relatives aux capteurs magnétiques (sensorIn, sensorOut). Il pourrait utiliser d'autres entrées de capteurs associés à d'autres types de capteurs.

La figure 7.2 est un diagramme de cas d'usage permettant d'illustrer les différentes étapes à effectuer par un opérateur pour intégrer ses capteurs et/ou actionneurs à la plateforme, via l'utilisation des modèles d'entités. Nous revenons sur ces

étapes dans les paragraphes suivants.

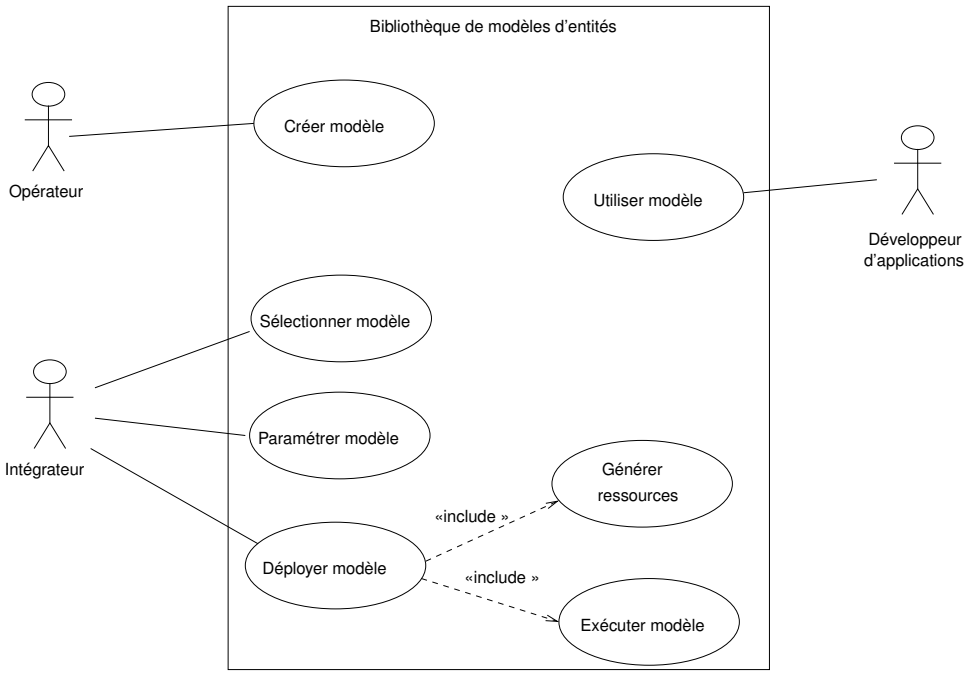


FIGURE 7.2 – Intégration des capteurs et actionneurs dans la bibliothèque de modèles d'entités dans le but de les partager

L'opérateur a un rôle d'intégrateur lorsqu'il souhaite partager ses capteurs et actionneurs. Il doit sélectionner un modèle parmi ceux présents dans la bibliothèque. Ce choix doit se faire en fonction de la spécification des modèles et des nouveaux équipements ou capteurs à intégrer. Par exemple, lorsque de nouveaux lampadaires sont installés, le rôle d'intégrateur consister à consulter la bibliothèque. Il est envisageable que ces lampadaires aient des modes de fonctionnement spécifiques et ne correspondent pas au modèle d'entité des lampadaires génériques présents dans la bibliothèque. Dans ce cas, un nouveau modèle d'entité doit être créé et ajouté à la bibliothèque. Il s'agit d'un rôle à part puisqu'il est de la responsabilité de l'opérateur d'effectuer cette tâche en suivant la démarche d'invention présentée en section 5.3. Cependant, nous n'en traiterons pas ici et nous considérons que le modèle choisi correspond aux attentes de l'opérateur.

Après avoir choisi un modèle d'entité, l'intégrateur doit le paramétrer. Ce paramétrage comprend la définition des valeurs des différentes variables contenues dans le modèle de l'entité choisie. Par exemple, si le modèle d'une voie est choisi, les seuils de celui-ci doivent êtres définis. Ces seuils correspondent au taux d'occupation moyen (variable MEDIUM) et élevé (variable HIGH) d'une route. Similairement pour le modèle du plot rétractable, la hauteur maximale du plot doit être paramétrée.

Ce paramétrage comprend également l'écriture du code pour acquérir les valeurs de capteurs et/ou envoyer des commandes aux actionneurs ; ce code est indépendant du modèle puisqu'il s'agit d'interfaces à implémenter. Pour envoyer des commandes,

l'intégrateur doit suivre le patron de conception Command [57] que nous avons utilisé pour sa simplicité : une interface doit être implémentée pour chaque commande envoyée aux actionneurs. Par exemple, pour intégrer des lampadaires nouvellement installés, les commandes requises pour allumer, éteindre et mettre en veille doivent être implémentées.

Afin d'utiliser des capteurs, l'intégrateur doit implémenter des interfaces spécifiques au modèle d'entité choisi. Ces interfaces doivent contenir le code qui permet d'obtenir des valeurs de capteurs utilisées par le modèle lors de son exécution à chaque tour d'exécution (cf. section 7.2.2.1). Pour le modèle d'entité d'une route, il peut y avoir plusieurs interfaces correspondant à différents types de capteurs utilisés. Dans notre exemple, nous avons choisi seulement des données de capteurs magnétiques.

Enfin, la dernière étape consiste à déployer le modèle. Le déploiement comporte deux étapes successives qui sont la génération des différentes ressources REST (cf. section 7.2.2.3) suivie de l'exécution du modèle choisi au sein du matériel qu'il juge nécessaire : serveurs dans le cloud ou matériel edge. Nous revenons sur cette étape en section 5.4.1 en exprimant nos préconisations à ce sujet.

Il est à noter que, de manière orthogonale à l'intégration de capteurs et actionneurs, les développeurs d'applications utilisent les modèles présents dans cette bibliothèque. Ces modèles sont utilisés au travers des ressources REST générées automatiquement afin de superviser et contrôler la ville.

7.2.2.3 Génération des ressources REST

Notre implémentation embarque un serveur CoAP, créé via l'API de Californium, qui est préalablement démarrée et qui rend accessible les ressources REST générées automatiquement (cf. section 5.5.2) aux applications. Les ressources que nous générons sont spécifiques au projet Californium et sont liées à l'implémentation de ce serveur CoAP. La génération de ressources REST pourrait également produire des ressources spécifiques à d'autres serveurs web, tels que CoAP (par exemple, nCoap [117]) ou HTTP (Jersey [86], RESTeasy [143], Apache CXF [43], etc.).

Nous avons également créé un annuaire générant des URIs uniques associés aux ressources nouvellement créées et aux ressources liées au contrôle de concurrence et à la coordination. Conformément à ce qui a été débattu par le W3C[98], nous distinguons les URIs des ressources REST générés (dits "informationnels") des URIs des éléments physiques (dits "non informationnels"). Ce choix a été fait pour permettre à une plateforme fédérant des données sémantiques de différencier les éléments physiques et les modèles que nous proposons. L'intégration d'une telle plateforme souhaitant se baser sur la nôtre est ainsi facilitée. Par conséquent, les URIs des ressources n'ont pas de sémantique par rapport aux éléments physiques. Par exemple la ressource R_A générée à partir du modèle de l'entité de la route R_1 a pour URI "/statemachines/18" ; l'URI "/road/1" étant lié à la route R_1.

Chaque ressource générée (R_A, R_A/cs, R_A/v_1, etc.) possède un nombre de pool de threads qui détermine le nombre de requêtes qu'elle peut traiter en parallèle. Ce

nombre est configurable mais nous ne nous en préoccupons pas puisqu'il est dépendant des ressources matérielles. Les requêtes reçues par une ressource sont mises en attente lorsqu'il n'y a plus de thread disponible. Les requêtes GET peuvent être ainsi parallélisées. Pour les requêtes PUT cela ne change rien car les changements d'états sont séquentiels, une requête PUT doit attendre que la précédente soit terminée pour s'exécuter.

Enfin, pour notifier les applications du format utilisé, nous intégrons l'option "Content-Format" dans les réponses envoyées qui a pour valeur le numéro identifiant relatif au format JSON.

Nous verrons en section 7.2.5 les codes de réponse CoAP que nous retournons aux applications lorsqu'elles supervisent et/ou contrôlent des entités. Ces codes de réponses CoAP correspondent aux acquittements des automates (cf. section 5.3.1.3)

7.2.3 Simulateur

Le simulateur développé possède une console d'administration dans laquelle deux modes peuvent être sélectionnés afin d'interagir avec les entrées et les sorties des modèles d'entités en cours d'exécution. Nous les décrivons ci-dessous.

7.2.3.1 Simulation des entrées des automates

Le premier mode du simulateur permet de simuler des entrées liées à chaque automate en cours d'exécution : valeur de capteur, unité de temps ou commande provenant des applications. L'interface est une console dans laquelle un automate doit être choisi parmi ceux en cours d'exécution. Une fois sélectionné, une saisie manuelle doit être effectuée comprenant le nom de l'entrée à simuler et sa valeur. Une fois la saisie effectuée, l'entrée sera prise en compte par l'automate dès qu'il atteindra l'étape relative à l'obtention des entrées (cf. ligne 3, figure 7.1).

Par exemple, si le modèle de l'entité de R_1 et du feu FT_1 sont en cours d'exécution, l'un d'entre eux peut être choisi dans la console. Si l'automate de R_1 est sélectionné, la saisie de "sensorIn=true" simulera la valeur d'un des capteurs magnétiques. Si l'automate de FT_1 est sélectionné, saisir "top=true" simulera une unité de temps (utilisée lorsque l'état est YELLOW) et saisir "cmdGreen=true" simulera une commande envoyée par une application détenant un verrou.

Il est possible d'injecter plusieurs autres entrées à l'automate choisi ou de choisir un autre automate en cours d'exécution.

7.2.3.2 Simulation des pannes et des latences réseaux des commandes envoyées

La console d'administration comporte un second mode qui permet de simuler la latence réseau ou bien la défaillance d'une commande envoyée aux actionneurs. Il est à noter qu'il est possible de changer de mode en passant de l'injection de pannes et de latences réseaux à l'injection des entrées (vice-versa).

La latence réseau correspond au délai aller/retour des commandes envoyées aux actionneurs. Cette latence est injectée à travers l'interface implémentée pour chaque envoi de commande. Un délai correspondant à cette latence est respectée à chaque envoi de la commande, tant que celui-ci n'est pas remis à zéro. Une fois le second mode sélectionné dans la console, un modèle d'entité en cours d'exécution doit être choisi. Une fois ce choix fait, le nom des différentes commandes de l'automate apparaît (nom de leurs classes). Une commande doit être choisie et doit être accompagnée d'une saisie manuelle indiquant le temps (en millisecondes) de la latence associée à cette commande. L'injection de cette latence prendra effet immédiatement.

De la même manière que pour la simulation de latences, une panne est associée à une commande. Une panne d'un ou plusieurs actionneurs correspond à l'échec d'envoi d'une commande à celui/ceux-ci, pour des raisons variées (pannes réseaux, actionneurs défaillants). Il est possible de simuler de manière combinée la latence réseau et la panne d'une commande. Par exemple, cela se traduit par une panne survenant après un délai correspondant à la latence injectée.

7.2.4 Implémentation des mécanismes pour le contrôle d'entités

Pour l'implémentation des mécanismes de contrôle de concurrence et du coordinateur, nous nous sommes basés sur la conception préalablement effectuée pour la validation de ces mécanismes dans l'outil SPIN. Dans un but d'uniformité des interactions avec les applications, les mécanismes sont également des ressources REST.

7.2.4.1 Mécanisme de contrôle de concurrence

Dans notre implémentation, une ressource R_A/tok est également instanciée lorsqu'un modèle d'entité controlable voit ses ressources REST générées. Cette ressource comporte l'implémentation de notre mécanisme de concurrence 6.1.

Nous avons implémenté de manière atomique l'acquisition d'un verrou et la génération d'un token. Seules des requêtes POST peuvent envoyées à la ressource R_A/tok qui sérialise ces dernières afin qu'un seul token soit généré. Une chaîne de caractères aléatoires est générée et envoyée en réponse à une requête POST lors d'une obtention du token. Un timer pré-défini est exécuté de manière indépendante. Son expiration correspond à la fin de la période de validité du verrou et donc à l'invalidation de la chaîne de caractères précédemment générée. La durée de ce timer est configurable lors de la création et de l'ajout d'un nouveau modèle d'entité dans la bibliothèque de modèles d'entités.

L'URI d'une ressource responsable du contrôle de concurrence est une chaîne de caractères aléatoires que nous générons à chaque obtention du verrou. Cet URI devient visible dans les réponses de R_A à des requêtes GET, dès que le verrou est disponible. Dans notre implémentation, l'URI d'une ressource R_A/tok est de la forme suivante (pour une entité quelconque) "/statemachines/5/tokA89zvRe" ou "/statemachines/5/tokRasV82" à un autre moment, lorsque le verrou est de nouveau disponible.

A l'aide d'un fragment d'application que nous avons conçu, nous présenterons en section 7.2.5.3 les différents codes de réponse CoAP qu'une ressource R_A/tok envoie lorsqu'elle traite une requête d'une application.

7.2.4.2 Mécanisme de coordination

Ressources d'enrôlement et de désenrôlement Les ressources R_A/erl et $R_A/derl$ (enrôlement/désénrôlement d'une entité) sont également instanciées dès que les ressources d'un automate A sont générées. Leur URI est unique, par exemple "/statemachines/4/enroll" pour R_A/erl et "/statemachines/4/disenroll" pour $R_A/derl$. Ces ressources peuvent seulement être utilisées par le coordinateur, les applications ne connaissent pas leurs URIs.

Concernant l'enrôlement, un coordinateur envoie une requête POST à une ressource R_A/erl qui contient l'état cible et le *cid* (cf. phase d'enrôlement, section 6.4.2). Dans notre implémentation, un acquittement positif se traduit par le code de réponse CoAP "2.04 CHANGED", précédé de l'obtention atomique du verrou et du stockage du *cid*. Nous avons considéré que si l'entité est déjà dans l'état souhaité, elle retournera également ce code de réponse. Pour un acquittement négatif il s'agit du code de réponse "4.02 BAD OPTION" si l'entité est déjà verrouillée ou "4.00 BAD REQUEST" si l'état cible n'est pas atteignable.

Pour le désenrôlement, un coordinateur envoie une requête POST à une ressource $R_A/derl$ contenant son *cid*. Au contraire de toutes les autres requêtes, les requêtes CoAP de désenrôlement sont de type "Non-Confirmable" (NON) puisqu'aucun acquittement n'est attendu par le coordinateur.

Ressource de coordination Une seule ressource est responsable de la coordination des entités, elle génère un processus de coordination lorsqu'elle reçoit une requête d'une application. Dans notre implémentation, l'URI de cette ressource est de la forme suivante "/coordinator" et est connu par les applications.

Des requêtes POST peuvent être envoyées à cette ressource contenant les couples $< URI\_R_A, etat\_cible >$. Dans notre implémentation, plusieurs coordinateurs peuvent être lancés simultanément où chacun détient un *cid* unique généré atomiquement. Pour accomplir les différentes phases du processus (cf. section 6.4.2), les URIs des ressources R_A/erl (enrôlement), R_A/cs (exécution) et $R_A/derl$ (désenrôlement) sont construits à partir de l'URI de chaque R_A contenu dans la requête de l'application. La construction de ces URIs à partir de l'URI de chaque R_A permet une flexibilité dans le déploiement du coordinateur (cf. sous-section 7.4.2)

Notre implémentation du processus de coordination comporte l'utilisation du verbe POST pour enrôler et désenrôler des entités (cf. paragraphe précédent), à partir des URIs construits. La phase d'exécution s'effectue de la même manière qu'un changement d'état traditionnel opéré par une application, c'est-à-dire en envoyant une requête PUT contenant l'état cible à la ressource R_A/cs. Dans notre implémentation, le coordinateur est notifié que l'état cible est atteint via un code de réponse de la classe 2.xx. Enfin, les différentes requêtes d'un processus de coor-

dination comportent l'option "cid" qui, comme son nom l'indique, a pour valeur le *cid* afin d'identifier le coordinateur.

Nous présenterons le fragment d'une application qui nous permet d'illustrer les différents codes de réponse CoAP renvoyés par une ressource de coordination (cf. section 7.2.5.5).

7.2.5 Implémentation des applications de démonstration

7.2.5.1 Conception des applications

Afin de créer les applications de démonstration, nous avons utilisé l'API cliente de Californium qui permet de créer, configurer et exécuter des clients CoAP. Nous ne nous préoccupons pas la configuration des applications (numéro de port, nombre de retransmissions, durée liée au timeout des requêtes, etc.) puisqu'elle est propre aux besoins de chacune d'entre elles.

Dans notre implémentation, nous partons du principe que les applications connaissent les URIs des ressources des entités et du coordinateur. Ces URIs seraient typiquement obtenus par des registres ou annuaires qui ne sont pas couverts par le travail de la thèse. De même, les applications connaissent la sémantique des représentations des ressources qui leurs sont envoyées. Ainsi, elles connaissent la signification de l'URI token_uri qui permet d'obtenir un token et le verbe à utiliser pour obtenir celui-ci (POST) mais également le sens des états et des variables ainsi que les verbes à utiliser pour modifier un état et/ou une variable (PUT) ou le lire (GET). Comme nous l'évoquions en section 4.1.3, l'utilisation de méta-données n'est pas un des éléments couvert par notre travail.

Pour réaliser notre démonstrateur, nous avons simulé une partie du comportement de certaines d'entre elles. L'implémentation de l'application gérant les accès à la zone de marchandises ne comporte que le contrôle du plot PR. L'identification d'un camion de marchandises, via la borne RFID, est simulée par une saisie manuelle dans la console. De même, l'application des travaux publics contrôle les feux FT_1 et FT_2 après qu'une saisie manuelle ait été effectuée. Enfin, la luminosité utilisée par l'application d'éclairage est directement simulée au sein de l'application par une saisie numérique dans la console de celle-ci.

Le fonctionnement des différentes applications que nous avons implémentées est le suivant :

— Application de contrôle d'accès : abaisse puis relève le plot PR après quelques secondes, suite à une saisie manuelle.

— Application des travaux publics : fait passer le feu FT_1 au vert et le feu FT_2 au rouge, suite à une saisie manuelle. Une saisie sur deux, la couleur des feux est inversée.

— Application gérant les places de parking : souscrit des notifications auprès de l'entité associée aux places de parking PP_1 à PP_8 via une requête OBSERVE et met à jour le message du panneau PAD à chaque notification reçue.

— Application d'éclairage public : supervise périodiquement la route R_1 , quand le taux d'occupation ainsi que la luminosité sont faibles alors elle éteint les lampadaires L_1 à L_4.
— Application informant l'occurrence de bouchons et d'un itinéraire de délestage : supervise périodiquement les routes R_1 et de R_2, lorsqu'elles sont toujours fortement occupées après plusieurs requêtes GET, un message est affiché sur le panneau PAD et le plot, de manière indépendante, est baissé.
— Application de fluidification du trafic : supervise périodiquement la route R_1 et coordonne les feux FT_1 et FT_2 avec le carrefour C, de sorte que FT_1 soit vert, FT_2 rouge et C soit passant d'Est en Ouest.

Ces scénarios nous permettent de présenter de manière exhaustive les différentes interactions qui peuvent exister entre les applications et les entités. Lorsqu'il s'agit de supervision d'entités, ces interactions comportent des requêtes GET/OBSERVE envoyées par les applications aux entités. Lors du contrôle d'entités, les interactions initiées par les applications tiennent compte des mécanismes de contrôle pour contrôler les entités.

Nous présentons ces diverses interactions dans les sous-sections suivantes via des fragments de ces applications. A partir de ces fragments d'applications et de l'exécution des différentes applications, nous présenterons le modèle de programmation que nous avons défini (cf. section 7.3)

7.2.5.2 Fragments pour la supervision d'entités

Nous montrons dans cette section le pseudo-code permettant de superviser les entités via des requêtes GET ou OBSERVE. Ces requêtes sont envoyées à des ressources R_A puisqu'elles comportent toutes les informations nécessaires pour superviser les entités (cf. sous-section 5.5.2.2)

Requête GET Le pseudo-code de la figure 7.3 illustre la supervision des routes R_1 et R_2 effectuée pas les différentes applications, via une requête GET.

```
1  reqGET.setURI(roadResourceURI);
2  reqGET.setAcceptContent(50); // format JSON
3  respGET = reqGET.sendGET();
4  switch(respGET.getResponseCode()) {
5    case CONTENT:
6       currentState = parseCurrentState(respGET.getBody());
7       uncontrolStates = parseUncontrolStates(respGET.getBody());
8       ... break;
9    // erreur cliente de paramètrage (4.xx)
10 }
```

FIGURE 7.3 – Pseudo-code d'une des applications créées qui envoie une requête GET pour superviser l'entité associée à la route R_1

Les deux premières lignes correspondent au paramétrage de la requête : son URI qui est celui de la ressource générée à partir d'une entité liée à une route, pouvant être R_1 ou R_2, ainsi que le type de format accepté par l'application qui est JSON définie par le numéro unique 50. Après avoir envoyé la requête et reçu la réponse (ligne 3), le code de réponse doit être extrait (ligne 3). Sauf erreur de paramétrage de la requête cliente (URI incorrect, format d'échange non supporté, verbe non accepté, etc.) comme commenté à la ligne 9, une requête GET renvoie le code de réponse CoAP "2.05 CONTENT" correspondant au succès de l'obtention de la représentation de la ressource. Cette représentation de la ressource contient diverses informations (cf. section 5.5.2.2). L'état courant est extrait (ligne 6) ainsi que le ou les états non commandable (ligne 7) par les applications.

Nous avons pu étudier la cohérence des réponses obtenues par les applications par rapport aux valeurs des capteurs simulées. En outre, ceci nous a permis de voir que l'état courant de l'entité est le même que celui obtenu par une application qui supervise l'entité.

Requête OBSERVE De la même façon que dans la section ci-dessus, nous avons souhaité tester le cas dans lequel une entité est supervisée au moyen d'une souscription. Le pseudo-code de la figure 7.4 illustre un fragment de l'application supervisant les places de parking PP_1 à PP_8 en envoyant une requête OBSERVE à la ressource R_{PP}

```
1  reqOBS.setURI(parkingSpotsURI);
2  reqOBS.setAcceptContent(50); // format JSON
3  reqOBS.sendOBS( new asyncHandler(respOBS) {
4     switch(respGET.getResponseCode()) {
5        case CONTENT:
6           availableSpots = parseAvailableSpots(respGET.getBody());
7           occupiedSpots = parseOccupiedSpots(respGET.getBody());
8        // erreur cliente de paramètrage (4.xx)
9   } });
```

FIGURE 7.4 – Pseudo-code de l'application de gestion des places envoyant une requête OBSERVE pour superviser l'entité associée aux places PP_1 à PP_8

Comme dans l'exemple de la figure 7.3, les lignes 1 et 2 correspondent au paramétrage de la requête, qui est ici une requête OBSERVE, à destination de la ressource générée à partir de l'entité représentant un ensemble de places de parking. La différence se trouve à la ligne 3 où après avoir envoyée une requête OBSERVE, une fonction asynchrone doit être passée en paramètre. Celle-ci gère les notifications envoyées par la ressource lorsque l'une des variables de l'entité change. Si le développeur a mal paramétré sa requête (ligne 8 commentée), la souscription à cette ressource est automatiquement annulée par Californium. Si les souscriptions ont un code de réponse signifiant le succès d'une notification (ligne 5), les valeurs des va-

riables associées au nombre de places disponibles et occupées seront extraites du corps de la notification reçue (ligne 6 et 7).

L'implémentation de cette application nous a permis de constater que lorsque nous simulons les valeurs des capteurs magnétiques, lesquels sont associés à chaque place de parking, l'application reçoit effectivement des notifications de la part de la ressource à qui elle a souscrite.

7.2.5.3 Fragment d'obtention d'un token

La figure 7.5 illustre le pseudo-code d'une application envoyant une requête à une ressource R_A/tok afin d'obtenir un token permettant de contrôler une entité. Il peut s'agir de n'importe quelle application (éclairage public, gestion du trafic routier, etc.) contrôlant une entité. Ce pseudo code nous permet de mettre en évidence les codes de réponse CoAP retournés aux applications dans notre implémentation ainsi que leur signification.

```
1  respPOST = reqPOST.setURI(tokenURI)
2  respPOST = reqPOST.sendPOST();
3  switch(respPOST.getResponseCode() {
4     case CHANGED:
5        id_token = parseIdToken(respPOST.getBody());  ... break;
6    case NOT FOUND: ... break;
7    // erreur cliente de paramètrage (4.xx)
8  }
```

FIGURE 7.5 – Pseudo-code d'une des applications de contrôle créé envoyant une requête POST pour obtenir le token d'une entité

Après avoir extrait l'URI de la ressource R_A/tok issu de la représentation de la ressource R_A obtenue par une requête GET ou OBSERVE (cf. section 6.2.2)), une requête POST est envoyée à la ressource R_A/tok (ligne 1 et 2). Hormis les codes de réponses liés à une erreur de paramétrage de la requête cliente (ligne 8), un client peut recevoir le code de réponse CoAP "2.05 CHANGED" (ligne 4) qui lui signifie le succès de sa requête ainsi que le token généré pouvant être extrait (ligne 5). Il peut recevoir le code de réponse "4.04 NOT FOUND" si une autre application est déjà détentrice d'un token valide.

Pour que ces deux codes de réponse puissent être reçus par les diverses applications, nous avons simulé les valeurs des capteurs utilisés par des entités qui sont supervisées par des applications de contrôle. Par exemple, en prenant le cas de l'application de déviation, nous avons simulé les valeurs de la route R_1 afin que l'application contrôle le plot PR et tente d'obtenir le token de l'entité. Dans le même temps, nous avons exécuté l'application gérant la zone restreinte via le contrôle du plot PR qui tente également d'obtenir le token. Cela nous a permis de générer diverses situations dans lesquelles l'une ou l'autre peut échouer ou acquérir le token.

Les interactions des applications pour obtenir un token nous ont conduit à identifier un modèle de programmation que celles-ci doivent respecter pour acquérir un token (cf. section 7.3.1.1).

7.2.5.4 Fragment pour contrôler une entité

Changement d'état Nous illustrons dans la figure 7.6 un fragment de l'application des travaux publics qui change l'état de l'entité associée au feu FT_1 afin que le feu soit rouge. Ce pseudo-code nous permet de montrer les codes de réponse CoAP que les applications peuvent recevoir ainsi que leurs significations. Après avoir extrait l'URI de la ressource R_{FT_1}/cs issu de la représentation de la ressource R_{FT} (cf. section 5.5.2.2) précédemment obtenue via une requête GET ou OBSERVE, celui-ci est stocké dans la variable "currentStateUri".

```
 1 reqPUT.setURI(currentStateURI);
 2 reqPUT.setOptions("token", id_token).setBody("RED");
 3 respPUT = reqPUT.sendPUT();
 4 switch(respPUT.getResponseCode()) {
 5   case CHANGED: ... break; // état cible atteint = ack_ok
 6   case CREATED:  // état intermédiaire atteint = ack_processing
 7     statusURI = respPUT.getOption("Location-path");
 8     respGETStat = reqGETStatus.setURI(statusURI).sendGET();
 9     respGETStat.getBody().getStatus();// état atteint=ack_finish_red
10     ... break;
11   case CONTINUE: ... break; // meme état = ack_ignore
12   case BAD REQUEST: ... break; // état cible non valide = ack_ko
13   case BAD OPTION: ... break; // expiration du token
14   case SERVER ERROR: ... break; // impossible d'exécuter la requete
15 }
```

FIGURE 7.6 – Pseudo-code de la requête PUT de l'application des travaux publics permettant de changer l'état de l'entité associé au feu FT_1

La requête PUT de changement d'état est construite en indiquant l'URI de la ressource R_{FT_1}/cs (ligne 2). Dans cette requête est incluse l'option contenant la valeur du token, précédemment obtenue, et l'état cible au format JSON (ligne 2) qui est contenu dans le corps de la requête (ligne 3). Suite à l'envoi de la requête (ligne 3), une réponse est reçue contenant un code de réponse qui doit être extrait (ligne 4). Peu importe le code de réponse reçu, la représentation de la ressource comporte l'état courant de l'entité qui a changé ou non.

Dans notre implémentation, le code de réponse "2.04 CHANGED" (ligne 5) signifie que l'état cible est atteint, ce qui correspond à l'acquittement ack_ok retourné par l'automate. L'exécution de l'application d'éclairage public nous a permis de vérifier que ce code est bien reçu : l'application est la seule à contrôler l'entité associée

au feu L_1 à L_4.

Le code “2.01 CREATED” (ligne 6) indique le succès de la requête et qu'un état intermédiaire est atteint, ce qui équivaut à l'acquittement ack_processing renvoyé à l'automate. L'exécution des applications contrôlant des entités pourvues d'un ou plusieurs états intermédiaires nous ont permis de vérifier que ce code de réponse était retourné.

Puisque CoAP se base sur un modèle requête/réponse, une seconde réponse correspondant à l'acquittement ack_finish ne peut être obtenue par une application que lorsque cette dernière envoie une nouvelle requête. Dans notre implémentation, nous créons une ressource pour stocker ce résultat, l'URI de celle-ci est indiqué dans l'option “Location-path” (ligne 7) permettant à l'application d'envoyer une requête GET ou OBSERVE (ligne 8). Ce résultat est contenu dans la réponse qui doit être extraite (ligne 9) car CoAP possède un nombre limité de code de réponse. L'état courant de l'automate est indiqué dans ce résultat ainsi qu'un attribut “Status” dont la valeur peut être “finish” si l'acquittement ack_finish a été généré par l'automate ; “running” si l'état courant est toujours l'état intermédiaire et “fail” si l'état atteint n'est pas l'état cible. La simulation du capteur de panne des feux FT_1 et FT_2, lorsque l'application liée aux travaux publics contrôlent ceux-ci, nous a permis d'identifier que les réponses envoyées étaient conformes à nos attentes.

Le code de réponse CoAP “2.31 CONTINUE” (ligne 7) indique le succès de la requête et que l'état courant de l'entité est déjà celui souhaité. Cela correspond à l'acquittement ack_ignore. Un cas d'usage nous a permis d'identifier que ce code était retourné lorsque l'application gérant l'accès à une zone restreinte baisse le plot PR alors que celui-ci est déjà abaissé puisque l'application déviant le trafic routier l'a contrôlé en dernier.

Le code “4.00 BAD REQUEST” (ligne 8) est relatif à l'acquittement ack_ko et indique l'échec de la requête car l'état cible ne peut être atteint. En simulant la valeur du capteur détectant la panne du feu FT_1 suivie d'une requête de changement d'état de l'application des travaux publics, nous avons pu générer un cas amenant ce code de réponse à être reçu.

Lorsque le token est expiré, l'application reçoit le code de réponse “4.02 BAD OPTION” (ligne 9). Ici, la simulation d'une latence des commandes envoyées aux actionneurs nous a permis de nous apercevoir que ce code de réponse était bel et bien retourné. Nous avons simulé une importante latence de la commande permettant d'abaisser le plot PR. Ainsi, lorsque l'application gérant l'accès à une zone restreinte abaisse puis relève le plot PR, la seconde requête aboutit à l'obtention de ce code de réponse puisque le token a expiré au cours de la première requête.

Enfin, en cas de défaillance menant l'entité à ne pas pouvoir effectuer le changement d'état, le code de réponse “5.00 SERVER ERROR” est reçu (ligne 10). La simulation de la défaillance des commandes envoyées aux actionneurs nous a permis d'identifier que ce code était cohérent avec la réalité.

De la même façon que pour l'obtention d'un token, les interactions des applications nous ont permis d'identifier un modèle de programmation à adopter lorsqu'elles reçoivent certain code de réponse (cf. section 7.3.1.2)

Changement de variable Comme le montre le pseudo de la figure 7.7, nous présentons un fragment d'une application qui met à jour une variable. Ici, il s'agit de la mise à jour du message du panneau PAD qui est effectuée par l'application informant les automobilistes de l'état du trafic. Ce fragment est similaire au fragment lié à une requête de changement d'état. Ceci s'explique par le fait que les variables sont une extension de notation des états (cf. section 5.4.2).

```
 1 reqPUT.setURI(varMessageURI);
 2 reqPUT.setOptions("token", id_token).setBody("Déviation à 200 m.");
 3 respPUT = reqPUT.sendPUT();
 4 switch(respPUT.getResponseCode()) {
 5   case CHANGED: ... break; //  variable mise à jour = ack_ok
 6   case CONTINUE: ... break; // meme valeur courante = ack_ignore
 7   case BAD REQUEST: ... break; // valeur incorrecte = ack_ko
 8   case BAD OPTION: ... break; // expiration du token
 9   case SERVER ERROR: ... break; // impossible d'exécuter la requete
10 }
```

FIGURE 7.7 – Pseudo-code de la requête PUT permettant de changer la variable message de l'entité associée au panneau PAD

L'URI de la ressource R_{PAD}/v_1, associé à la variable du message, a été extrait de la représentation de la ressource R_{PAD} (cf. section 5.5.2.2) obtenue via une requête GET ou OBSERVE. Ici, cet URI est stocké dans la variable "varMessageURI"..

La construction de la requête PUT est similaire à la requête PUT de changement d'état (ligne 1 à 3). Celle-ci doit comprendre l'URI de la ressource, l'option comportant la valeur du token ainsi que le contenu de la requête comportant la valeur de la variable.

Avec différentes exécutions de l'application informant les automobilistes de l'état du trafic et de l'application gérant du nombre de places de parking disponibles, nous avons pu vérifier que les différents codes de réponses (lignes 5 à 7) étaient renvoyés. La simulation de panne nous a permis d'identifier que le code de réponse "5.00 SERVER ERROR" était retourné (ligne 9). De même, pour l'application gérant le nombre de places de parking, nous avons simulé les valeurs des capteurs envoyées à l'entité associée à PP_1 à PP_{10} que cette application supervise. En complément, nous avons simulé une forte latence des actionneurs du panneau PAD afin que le token expire et que le code "4.00 BAD OPTION" soit renvoyé (ligne 8).

7.2.5.5 Fragment pour la coordination d'entités

La figure 7.8 illustre un fragment de l'application fluidifiant le trafic routier qui permet de coordonner les entités du feu FT_1, du feu FT_2 et du carrefour C. Ce fragment nous permet de montrer comment doit être paramétrée la requête à destination de la ressource de coordination. Cela nous permet également de montrer

les codes de réponses retournés aux applications ainsi que leurs significations.

```
 1 reqPOST.addBody(addParticipant(trafficLight1URI, "RED"));
 2 reqPOST.addBody(addParticipant(trafficLight2URI, "GREEN" ))
 3 reqPOST.addBody(addParticipant(crossroadURI, "N-S RED  E-W GREEN"))
 4 respPOST = reqPOST.setURI(coordinatorURI).sendPOST();
 5 switch(respPOST.getResponseCode()) {
 6    case CHANGED: ... break; // coordination reussie
 7    case BAD REQUEST: // echec de l'enrolement d'un participant
 8       participantURI = respPOST.getBody().getParticipantURI();
 9       participantStatus = respPOST.getBody().getParticipantStatus();
          ... break;
10    case SERVER ERROR: // echec du changement d'etat d'un participant
11       participantsURI = respPOST.getBody().getParticipantsURI();
12       participantsStatus = respPOST.getBody().getParticipantsStatus();
13       ... break;
14    // cas lie au mauvais parametrage de la requete (4.xx)
15 }
```

FIGURE 7.8 – Pseudo-code de la requête POST de coordination des entités du feu FT_1, du feu FT_2 et du carrefour C envoyé par l'application visant à fluidifier le trafic routier

Les lignes 1 à 3 illustrent le contenu de la requête POST comprenant l'URI ainsi que l'état cible de chaque participant . Cette requête est ensuite envoyée à la ressource de coordination, en ayant spécifié son URI (ligne 4). Une fois la réponse reçue, le code de réponse est extrait (ligne 5).

Si le processus de coordination a réussi, le code de réponse CoAP "2.04 CHANGED" sera reçu par l'application. En exécutant l'application de fluidification du trafic sans qu'elle entre en conflit avec l'application liée aux travaux publics et lorsque les états des entités sont atteignables, nous avons pu générer ce cas.

En revanche, si le processus de coordination a échoué pendant la phase d'enrôlement alors le code de réponse illustré à la ligne 10 est renvoyé à l'application ("4.00 BAD REQUEST"). La réponse associée contient l'URI de la ressource qui n'a pas pu être enrôlé et son statut (lignes 8 et 9) : "locked" s'il est déjà verrouillé par une autre application ou "invalid" si l'état voulu n'est pas atteignable. Ici, il nous a suffit de tenter de provoquer des conflits entre les deux applications contrôlant les feux FT_1 et FT_2 pour générer ce cas.

Si le processus de coordination a échoué durant la phase d'exécution, à cause d'une erreur matérielle (actionneur défaillant) ou d'une panne réseau par exemple, le coordinateur renvoie le code de réponse "5.00 INTERNAL SERVER ERROR" illustré à la ligne 14. La réponse associée à ce code de réponse contient un récapitulatif du statut de chaque participant (lignes 11 et 12) : "changed" si l'état de l'entité a été modifié, "remained" si l'état est inchangé, et "failed" lorsqu'il s'agit de

l'entité qui a échoué à effectuer le changement d'état. Ici, il nous a suffit de simuler la défaillance des actionneurs des feux FT_1 pour générer ce cas.

Nous donnons en section 7.3.2 des indications aux développeurs d'applications afin qu'ils appréhendent les divers codes de réponse décrits ici.

7.3 Applications et modèle de programmation à adopter

7.3.1 Contrôle d'une entité

Cette section vise à montrer le modèle de programmation que les applications doivent adopter pour changer l'état d'une entité. Nous avons inféré ce modèle à partir des exécutions des différentes applications qui contrôlent les mêmes entités.

La figure 7.9 décrit la signature de la fonction contenant le pseudo-code de ce modèle de programmation. Cette fonction est composée d'une partie pour obtenir le token d'une entité (cf. section 7.3.1.1) et d'une autre partie pour changer l'état de l'entité (cf.section 7.3.1.2). Les paramètres de cette fonction sont : l'état cible souhaité par l'application ("targetState"), l'URI de la ressource R_A ("resourceURI") ainsi que deux valeurs de timer ("tokenWaiting" et "failureWaiting") sur lesquelles nous reviendrons. Ces paramètres sont utilisés dans le pseudo-code des sous-sections suivantes.

```
1 function stateChange(targetState, resourceURI, tokenWaiting,
                                    failureWaiting) {
2   // premiere partie: acquisition du token : voir section 7.3.1.1
3   // seconde partie: changement d'etat : voir section 7.3.1.2
4 }
```

FIGURE 7.9 – Signature de la fonction qui contient le modèle de programmation pour changer l'état d'une entité

7.3.1.1 Acquisition du token

La figure 7.10 illustre le pseudo-code typique permettant à une application d'acquérir un token afin d'effectuer un changement d'état.

```
1  id_token = null;
2  while(id_token is null) {
3    respGET = reqGET.setURI.(resourceURI).GET();
4    switch(respGET.getResponseCode()) {
5      case CONTENT:
6        currentStateURI = parseCurrentURI(respGET.getBody());
7        controlStates = parseControlables(respGET.getBody());
8        tokenURI = parseTokenURI(respGET.getBody());
9        if(targetState is in controlStates
                                  AND tokenURI is not null) {
10         respToken = reqPOST.setURI(resourceURI+tokenURI)
                             .sendPOST();
11         if(respToken.getResponseCode() is CHANGED) {
12           id_token = parseIdToken(respToken.getBody());
13         }
14         else if(respToken.getResponseCode() is NOT FOUND) { ... }
15       }
16     // probleme de parametrage de la requete cliente (4.xx)
17   }
18   wait(tokenWaiting);
19 }
```

FIGURE 7.10 – Exemple type de pseudo-code pour acquérir le token d'une entité avant de changer son état

Les lignes 1 et 2 correspondent à l'initialisation du token, qui n'a pas encore été acquis, et au fait que la requête de changement d'état ne sera pas envoyée tant que le token n'est pas obtenu.

Pour obtenir un token, une requête GET doit être envoyée à l'URI de la ressource de l'entité que l'application souhaite contrôler (ligne 3). Après avoir reçu la réponse et extrait le code de celui-ci (ligne 4), le développeur doit effectuer différents traitements. Nous ne nous préoccupons pas des cas liés à un mauvais paramétrage de la requête (ligne 16 commentée), tel qu'un URI incorrect, car cela conduit le développeur à devoir re-paramétrer et recommencer son envoi.

Par conséquent, lorsque la requête a réussi (ligne 5), l'URI de l'état courant (ligne 6), la liste des états commandables (ligne 7) ainsi que l'URI du token (ligne 8) doivent être extraits du corps de la réponse retournée. L'URI de l'état courant sera utilisé par la suite pour envoyer la requête de changement d'état (cf. section 7.3.1.2). La liste des états commandables est consultée afin de savoir si l'état cible est présent parmi celle-ci. De même l'URI du token est consulté afin de savoir si un token est disponible (ligne 9). Si ces deux conditions sont respectées, une requête POST est envoyée à destination de la ressource générant un token, l'URI de celle-ci étant issu d'une concaténation de l'URI absolu de la ressource et de l'URI relatif du token (ligne 10).

Lors de la réception de la réponse, si le code de celle-ci indique une réussite (ligne 11) alors le token doit être extrait du contenu de la réponse (ligne 12). Ainsi, l'envoi d'une requête de changement d'état a lieu dès que le token a été obtenu. En revanche, si le token n'a pas pu être obtenu, tel qu'indiqué par le code de réponse à la ligne 14, le développeur est libre d'effectuer les traitements qu'ils jugent nécessaires. Le développeur sera également libre d'attendre un délai qu'il a lui-même configuré afin de retenter d'acquérir le token (ligne 18).

7.3.1.2 Requête de changement d'état

Nous décrivons ici le modèle de programmation à adopter lorsqu'une application souhaite effectuer un changement d'état, après avoir acquis un token. Nous illustrons nos propos via le pseudo-code de la figure 7.11. Celui-ci vient chronologiquement après le pseudo-code de la figure 7.10 décrit dans la sous-section précédente.

```
 1  reqPUT.setURI(currentStateURI).setOption("token", id_token);
 2  respPUT = reqPUT.setBody(targetState).sendPUT();
 3  switch(respPUT.getResponseCode()) {
 4    case CHANGED: ...  break; // etat cible atteint
 5    case CREATED: // etat intermediaire atteint
 6      statusURI = respPUT.getOptions().getLocationPath();
 7      reqOBS.setURI(statusURI); // ressource stockant le resultat
 8      reqOBS.sendOBS(new AsyncHandler(respOBS) {
 9                          if(respOBS.getResponseCode() is CONTENT) {
                                  ...
                            }
10                    }); break;
11    case CONTINUE: ...  break; // etat cible deja atteint
12    case BAD REQUEST: ... break; // etat impossible d'etre atteint
13    case BAD OPTION: // token expire, re-acquisition du token
14      acquireToken(targetState, resourceURI, tokenWaiting,
                                  failureWaiting); break;
15    case SERVER ERROR: // changement d'etat ne pouvant etre effectue
16      wait(failureTime); // attente avant une autre tentative
17      retryPUT = reqPUT.sendPUT(); ... break;
18  } }
```

FIGURE 7.11 – Exemple type de pseudo-code pour effectuer un changement d'état

Les lignes 1 et 2 correspondent au paramétrage et à l'envoi de la requête PUT de changement d'état. Celle-ci contient, le token précédemment obtenu ainsi que l'état cible souhaité par l'application. Une fois le code de réponse reçu (ligne 3), l'application doit traiter les différents cas possibles. Si la requête de changement d'état a été exécutée avec succès (lignes 4 et 5), l'application doit effectuer les traitements qu'elle

juge nécessaires. Lorsqu'un état intermédiaire est atteint, nous recommandons aux développeurs d'applications d'utiliser une requête OBSERVE pour être notifié du résultat comme nous le présentons aux lignes 6 à 8. Les développeurs sont libres d'effectuer les actions nécessaires suivant le résultat obtenu (ligne 9) qui est contenu dans la notification envoyée.

Les développeurs doivent traiter le cas où, entre l'acquisition du token et l'envoi de la requête de changement d'état, l'état courant de l'entité a changé. Si l'état courant est celui souhaité (ligne 11) ou que l'état cible n'est plus commandable (ligne 12), il est de la responsabilité du développeur d'application d'effectuer les actions appropriées. A noter que l'état courant a pu changer, non pas à cause d'une autre application puisque notre mécanisme de contrôle de concurrence l'en empêche, mais à cause d'une valeur de capteur qui a été reçue entre-temps (exemple : panne du feu tricolore).

Quand le token a expiré (ligne 13), le développeur d'application peut tenter d'acquérir de nouveau le token afin que sa requête de changement d'état puisse potentiellement avoir lieu et être exécutée. De façon similaire, si une panne a amené l'entité à ne pas pouvoir effectuer le changement d'état, le développeur d'application peut programmer une nouvelle tentative d'envoi après avoir attendu un certain délai qu'il a lui-même défini.

7.3.1.3 Note sur la mise à jour de variable

Le modèle de programmation relatif à la mise à jour d'une variable serait en tout point similaire au changement d'état. Seuls des détails relatifs au contenu de la requête, c'est-à-dire la valeur de la variable et l'URI de la ressource auxquels sont envoyés cette requête, devraient être changés.

En effet, l'acquisition du token se fait en fonction de la valeur courante de la variable, afin de ne pas mettre à jour la variable inutilement. La requête PUT de mise à jour de la variable se fait de la même manière que pour une requête de changement d'état. Plus précisément, les différents codes de réponse doivent être gérés de la même façon pour une requête de changement d'état que pour une requête de mise à jour d'une variable. Exception faite du code relatif à un état intermédiaire atteint qui n'est jamais retourné lors du contrôle d'une variable et qui n'a donc pas besoin d'être traité pour ce type de requête.

7.3.2 Coordination de plusieurs entités

Nous décrivons dans la figure 7.12 un extrait de code caractéristique d'une fonction permettant de coordonner plusieurs entités. Nous avons inféré ce pseudo-code à partir des différentes exécutions de l'application de fluidification du trafic routier. Les paramètres de cette fonction sont : les couples $< URI, etat\_cible >$ des entités à coordonner ("entities"), l'URI de la ressource de coordination ("coordinatorURI") ainsi que le délai d'attente avant de coordonner de nouveau les entités ("retryWaiting").

```
1  function coordination(entities, coordinatorURI, retryWaiting) {
2    foreach(entities) {
3       reqPOST.addBody(addParticipant(entities.getURI(),
                        entities.getTargetState()));
4    }
5    respPOST = reqPOST.setURI(coordinatorURI).POST();
6    switch(respPOST.getResponseCode()) {
7      case CHANGED: ... break;
8      case CONTINUE: ... break;
9      case BAD REQUEST:
10       participantStatus = respPOST.getBody().getParticipantStatus();
11       if(participantStatus is locked) {
12         wait(retryWaiting);
13         coordination(entities, coordinatorURI, retryWaiting);
14       } else if(participantStatus is invalid) { ... }
15       ... break;
16     case INTERNAL ERROR:
17       wait(retryWaiting);
18       coordination(entities, coordinatorURI, retryWaiting); break;
19     // cas lie au mauvais parametrage de la requete cliente (4.xx)
20 } }
```

FIGURE 7.12 – Exemple type de pseudo-code pour coordonner des entités représentant deux feux tricolores et un carrefour

Les lignes 1 et 2 sont relatives à l'ajout de participant au sein de la requête POST afin de coordonner ces entités. Les lignes 5 et 6 correspondent à l'envoi de la requête de coordination à l'URI de la ressource de coordination et à la réception d'une réponse dont le code est extrait.

Lorsque la coordination a abouti (lignes 7 et 8), l'application peut effectuer les traitements jugés nécessaires. Si le processus de coordination n'a pas pu avoir lieu à cause d'un mauvais paramétrage de la requête cliente (ligne 19 commentée), l'application devra paramétrer de nouveau la requête (URI du coordinateur, contenu de la requête, etc.) avant de la ré-envoyer.

Lorsque la coordination échoue à cause d'un problème d'enrôlement d'une entité (ligne 9), l'application doit extraire la raison pour laquelle l'une des entités n'a pas pu être enrôlée (ligne 10). S'il s'agit d'une entité qui est déjà verrouillée (ligne 11), alors la même requête de coordination peut être relancée (ligne 13) après avoir attendu un délai (ligne 12) qui a été spécifié par l'application dans les paramètres de la fonction. En revanche, s'il s'agit d'une entité participante ne pouvant pas atteindre l'état souhaité (ligne 14), elle devra renoncer à effectuer ce processus de coordination et effectuer les traitements qu'elle juge nécessaire.

Lorsque le processus de coordination a échoué durant la phase d'exécution (ligne

16), l'application peut attendre un délai qu'elle a spécifié (ligne 17) et tenter de recommencer le processus de coordination (ligne 18). Il est à noter qu'en fonction des exigences de l'application, celle-ci pourrait contrôler seulement l'une des entités qui a mené à l'échec de la coordination et ne pas recommencer le processus de coordination.

7.4 Recommandations pour le déploiement dans une infrastructure distribuée

Dans cette section, nous préconisons des recommandations relatives au déploiement dans une infrastructure distribuée du code des modèles, des mécanismes de contrôle ainsi que des applications. Comme nous l'avons identifié en sous-section 4.1.4, cette infrastructure possède trois étages ; seul l'étage correspondant au matériel edge et aux serveurs hébergés dans une infrastructure cloud nous intéresse ici.

7.4.1 Déploiement des entités

Le code relatif aux entités correspond aux modèles d'entités et aux ressources REST des entités (R_A, R_A/cs, R_A/tok, R_A/erl, $R_A/derl$, R_A/v_1, R_A/s_1, etc.). Nous recommandons de déployer ce code au sein du matériel formant l'infrastructure edge dans laquelle un serveur CoAP est déployé au sein de chacun de ces matériels. Ainsi, ceux-ci exposent un nombre limité de ressources aux applications pour leur permettre de superviser et contrôler les éléments physiques présents dans leurs zones géographiques respectives.

7.4.1.1 Contrainte de latence

Ce choix est en concordance avec notre objectif visant à favoriser l'émergence des applications de contrôle temps-réel dans la ville intelligente. Déployer le code des entités au plus proche des capteurs et actionneurs permet de satisfaire cet objectif puisque les données des capteurs et les commandes envoyées aux actionneurs transitent uniquement par des réseaux dont un certain niveau de qualité de service est garanti (cf. sous-section 4.3.1).

En revanche, nous ne préconisons pas de déployer le code relatif aux entités au sein de serveurs hébergés dans une infrastructure cloud computing. Cela provoquerait une latence supplémentaire, due au réseau séparant les serveurs du matériel edge, ne permettant plus dans notre contexte de satisfaire les contraintes temps-réel de certains types d'applications.

7.4.1.2 Contrainte de fiabilité

En terme de fiabilité, déployer notre code au sein des serveurs mutualisés dans l'infrastructure cloud semble un choix idéal puisque ceux-ci peuvent comporter un

service de réplication utilisant un algorithme de consensus standard (Paxos [93], Zab [142], Raft [127])) pour garantir la tolérance aux pannes. Cependant, il nous faut prendre en considération ce qu'implique les autres pannes : coupure des réseaux de capteurs et actionneurs, défaillance du matériel edge ou encore pannes des réseaux séparant le matériel edge des serveurs.

En cas de coupure d'un ou plusieurs réseaux de capteurs et d'actionneurs, les conséquences sont les mêmes quelque soit l'emplacement du code. Les ressources REST peuvent répondre à des requêtes GET et/ou PUT. Cependant l'état courant n'est plus à jour puisqu'aucune donnée des capteurs est reçue et aucune commande aux actionneurs ne peut être envoyée. Ainsi, quelque soit l'emplacement du code, les applications de supervision ont une vision inconsistante des éléments physiques et les applications de contrôle ne peuvent voir leurs requêtes aboutir.

Lorsque le code est déployé au sein des serveurs, la coupure du réseau séparant le matériel "edge" des serveurs entraînera les mêmes conséquences que celles décrites dans le paragraphe précédent. Au contraire, si ce code est déployé au sein de l'infrastructure edge, alors il n'y aura aucune conséquence pour les entités et les applications qui l'utilisent.

En cas de défaillance de l'un des matériels "edge", peu importe où le code des entités est déployé, l'impact est le même pour les applications. Si le code est déployé au sein des serveurs, les applications pourront interroger les ressources REST via des requêtes GET mais la réponse retournée sera inconsistante puisque les données des capteurs ne peuvent plus être acheminées. De même, les requêtes PUT n'aboutiront pas puisque le matériel edge ne pourra plus relayer les commandes envoyées aux actionneurs.

Ainsi, bien que les serveurs aient un meilleur niveau de tolérance des pannes que le matériel formant le edge, les différents cas de pannes nous amènent à conclure qu'il est préférable de déployer le code des entités au sein du matériel edge.

7.4.2 Déploiement du mécanisme de coordination

Le déploiement du mécanisme de coordination correspond au déploiement du code relatif à la ressource de coordination. Compte-tenu de notre implémentation, où les URIs des entités participantes sont construites à partir des requêtes des applications, le coordinateur peut être déployé au sein des serveurs comme au sein du matériel edge.

Nous recommandons de ne pas déployer la ressource de coordination au sein de l'infrastructure cloud, bien que cette dernière solution possède des atouts en terme d'élasticité et de scalabilité. En cas de panne du réseau séparant le matériel edge des serveurs, les communications entre les coordinateurs et les entités participantes sont impossibles et donnent lieu à l'échec des processus de coordination. De même, comme évoqué dans la sous-section précédente, les délais de communication se trouvent impactés compte-tenu des latences réseaux entre les serveurs et le matériel edge. Puisque la coordination est utilisée par des applications de contrôle pouvant avoir des contraintes temps-réel, nous ne recommandons pas le déploiement de ce mécanisme

au sein des serveurs.

A l'opposé le déploiement du code du mécanisme de coordination au sein du matériel "edge" évite ces délais de communication supplémentaires. De ce fait, nous recommandons de déployer la ressource de coordination au sein de chaque matériel "edge" où chaque coordinateur coordonne les entités hébergées au sein du même matériel. Notre recommandation s'appuie sur le fait qu'il fait sens uniquement de coordonner des entités proches géographiquement, tels que les feux FT_1, FT_2 et le carrefour C. Les entités gérées par le matériel edge étant du ressort du découpage géographique effectué.

Enfin, en terme de fiabilité, la défaillance du mécanisme de coordination implique que le matériel edge est lui même défaillant. Il devient alors impossible de communiquer avec les capteurs et actionneurs sous-jacents, quelque soit l'endroit où la ressource est déployée. Ainsi, la défaillance d'un coordinateur ne peut être résolue par le protocole de coordination et une intervention d'un opérateur humain est requise.

Il est à noter que dans le cas de déploiement du mécanisme de coordination sur le matériel "edge", les applications doivent connaître tous les URIs des ressources de coordination qui sont déployés.

7.4.3 Déploiement des applications

7.4.3.1 Délais impactant les applications

Les applications peuvent être déployées au sein des serveurs ou du matériel edge. En fonction de ce choix, celles-ci sont sujettes à divers délais que nous illustrons dans la figure 7.13. Dans cette figure, nous prenons le cas d'une application de contrôle, hébergée au sein d'un serveur, qui contrôle une entité.

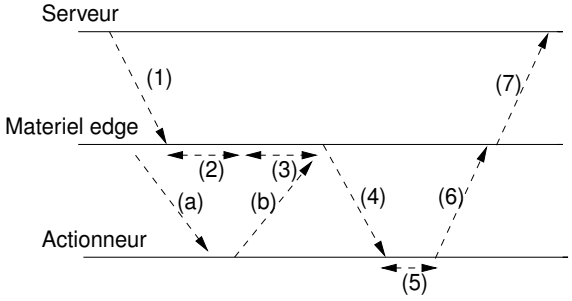


FIGURE 7.13 – Délais lorsqu'une requête de contrôle est envoyée depuis un serveur vers une entité qui l'exécute avec succès

Les lignes pointillées (1) et (7) sont relatives aux délais de communication liés au protocole applicatif (ici, CoAP) utilisé entre les serveurs et le matériel edge. Comme décrite en sous-section 4.3.1.2, cette latence est très variable et peut être de l'ordre de quelques millisecondes à quelques secondes. Ces délais de communications sont propres à un déploiement sur des serveurs, les délais que nous présentons par la suite

concernent à la fois des applications hébergées au sein de serveurs comme au sein de matériels "edge".

Ceux-ci peuvent être acceptables par certains types d'application de contrôle simple ou de supervision. Pour ce type d'application, qui correspond dans notre exemple à l'application d'éclairage ou de disponibilité des places de parking, nous recommandons un déploiement au sein des serveurs. De plus, ce choix de déploiement permet de ne pas surcharger inutilement le matériel edge.

La ligne pointillée (2) correspond au délai lié à la mise en attente de requêtes (GET et/ou PUT) REST reçues. Ce délai est propre à chaque ressource du serveur CoAP, lorsque un nombre important de requêtes est en cours de traitement par une ressource, les requêtes reçues à destination de celles-ci sont mises en attente. Nous considérons que ce délai est négligeable, seules les requêtes PUT peuvent engendrer un traitement conséquent et celles-ci ne peuvent être envoyées que par une seule application à la fois, compte tenu de notre mécanisme de contrôle.

La ligne pointillée (3) est le délai lié à la mise en attente de la commande avant d'être envoyée à l'actionneur. Une fois la commande précédente terminée comme illustrée par la ligne pointillée (b), en réponse à la commande en ligne pointillée (a), la commande mise en attente peut être envoyée. Ce délai est négligeable puisque notre mécanisme de contrôle de concurrence n'autorise le contrôle d'une entité pour une seule application à la fois.

Enfin, les lignes pointillées (4) et (6) sont liées au délai de communication entre entité et actionneur. Cette latence doit être maîtrisée pour que des applications temps-réel puissent exister (cf. sous-section 4.3.1.1). Afin d'être exhaustif dans notre description des délais, la ligne pointillée (5) représente le délai lorsqu'un actionneur reçoit une requête et envoie la réponse. Celui-ci est très faible (inférieur à quelques millisecondes) puisqu'il correspond simplement au traitement de la requête.

Ainsi, nous recommandons de déployer les applications qui ont des contraintes temps-réel au sein du matériel "edge" puisque les temps de réponse peuvent être garantis, à défaut d'un déploiement au sein des serveurs. A noter que les applications qui ont des contraintes temps-réel et qui sont déployées au sein du matériel "edge" ne doivent pas être consommatrices d'un trop grand nombre de ressources matérielles afin que le matériel "edge" ne soit pas surchargé.

7.4.3.2 Fonctionnement autonome des applications

Pour les applications qui ont besoin de pouvoir fonctionner en autonomie, nous recommandons de les déployer au sein du matériel "edge". Le déploiement au sein de serveurs hébergés dans une infrastructure cloud possède de nombreux avantages pour les applications (passage à l'échelle, ressources matérielles nombreuses, etc) mais il ne permet pas un mode fonctionnement autonome des applications.

En effet, les applications déployées au sein des serveurs ne peuvent plus contrôler ni superviser les entités lorsqu'il y a une coupure du réseau séparant les serveurs du matériel edge. Au contraire, les applications déployées au sein du matériel edge peuvent toujours contrôler/superviser les entités présentes au sein du même matériel.

7.5 Conclusion

Nous avons présenté dans ce chapitre l'implémentation de notre travail qui est basé sur notre exemple conducteur. Notre implémentation est composée de modèles d'entités, d'un simulateur de la ville intelligente, des mécanismes de contrôle ainsi que d'applications de démonstration. L'objectif de notre implémentation était d'ordre qualitatif. Nous avons montré comment notre travail facilite l'intégration de nouveaux capteurs et actionneurs ainsi que le développement d'applications pour la ville intelligente.

Nous avons décrit en détail notre preuve de concept, qu'il s'agisse de choix technologiques ou d'aspects techniques à propos des éléments implémentés (automates, simulateur, applications, mécanismes de contrôle). Nous avons donné des détails à propos de la bibliothèque de modèles d'entités qui est utilisée par les différents acteurs de la ville pour créer, déployer et utiliser des modèles d'entités. L'utilisation des modèles d'entités s'effectuant par l'intermédiaire des ressources, nous avons décrit en détail les interactions qu'ont les applications avec ces dernières.

Puis, nous avons présenté le modèle de programmation que les applications doivent adopter pour contrôler et superviser la ville intelligente. Les applications doivent tenir compte des mécanismes de contrôle de concurrence et de coordination pour interagir avec les différentes entités. En effet, ces mécanismes ont une influence sur le modèle de programmation des applications.

Enfin, nous avons proposé des recommandations relatives au déploiement de code des entités, des mécanismes et des applications au sein d'une infrastructure distribuée. Nous préconisons de déployer le code des entités au sein du matériel edge pour satisfaire les exigences de contrôle temps-réel de certaines applications. Nous avons également identifié que de déployer du code au sein de serveurs n'était pas forcément judicieux lorsque la raison invoquée était la fiabilité. Pour les mêmes raisons que nous venons d'évoquer, nous recommandons de déployer le code de nos mécanismes au sein du matériel edge. Pour finir, le déploiement du code des applications doit tenir compte des contraintes de celles-ci. Nous préconisons de ne pas déployer des applications au sein du matériel edge lorsque celles-ci n'ont pas de contraintes temps-réel afin d'éviter une surcharge du matériel edge. Au contraire, lorsque les applications ont des contraintes temps-réel, nous recommandons de les déployer au sein de ce matériel.

Chapitre 8

Conclusion et Perspectives

Sommaire

8.1 Conclusion

L'objectif de cette thèse était de contribuer au développement d'une plateforme horizontale pour la ville intelligente et à son adoption par les différents acteurs opérant et utilisant des équipements de celle-ci.

Nous avons commencé par identifier le contexte de notre travail nous permettant de déterminer que la sécurité était un enjeu majeur dans une plateforme horizontale. Nous n'avons pas adressé cette thématique qui pourrait faire l'objet d'un autre travail de thèse. Cependant, nous avons établi qu'une plateforme partagée pour la ville intelligente profitait d'un environnement maîtrisée. Une plateforme horizontale n'est ni totalement ouverte, au sens où n'importe quelle application peut l'utiliser, ni totalement fermée puisque de nouveaux acteurs sont potentiellement amenés à s'intégrer à une ville. De ce fait, nous sommes partis du principe que les applications utilisant la plateforme devaient être connues, identifiées et tenues de respecter un contrat légal avec l'entité responsable de la ville ou de la métropole.

Puis, nous avons mis en lumière les problèmes peu traitées dans les plateformes actuelles et constaté que les applications de contrôle n'étaient peu ou pas adressées par ces plateformes.

Pour cela, nous avons développé deux mécanismes de contrôle qui favorisent le partage et la réutilisation des actionneurs afin de permettre à des applications de contrôle d'utiliser une plateforme horizontale. Le premier est un mécanisme de contrôle de concurrence que nous avons conçu après avoir déterminé que des conflits apparaissaient dès lors que les actionneurs étaient partagés entre les applications des acteurs de la ville intelligente. Le second est un mécanisme de coordination conçu pour permettre de contrôler simultanément plusieurs actionneurs et ainsi favoriser la réutilisation de ceux-ci. Nous avons développé des programmes Promela et utilisé l'outil SPIN afin de valider les principes des mécanismes que nous avons proposés.

Nous avons déterminé qu'il fallait tenir compte des contraintes "temps-réel" que pouvaient exiger certaines applications de contrôle. Afin d'y parvenir, nous avons identifié plusieurs critères. Les latences des réseaux d'accès aux actionneurs doivent être bornés et garantis. Une infrastructure edge doit être privilégiée pour maîtriser les délais des communications entre les applications et les capteurs et actionneurs. Il fallait également tenir compte des contraintes "temps-réel" des applications dans la conception des solutions solutions proposées.

Dans ce travail de thèse, nous avons également contribué à promouvoir l'utilisation d'une plateforme horizontale par les différents acteurs de la ville.

Nous avons choisi de représenter la ville intelligente par des entités dont les modèles, issus du domaine des systèmes réactifs, ont fait leurs preuves. Les entités sont une consolidation des données récoltées et sont génériques puisqu'ils représentent les éléments physiques que toute ville possède. Afin de faciliter la supervision et le contrôle de la ville intelligente, nous avons standardisé les interactions avec ces modèles en respectant les principes architecturaux REST et en choisissant une architecture orientée ressource. Dans cette optique, nous avons présenté un modèle de programmation à adopter par les développeurs d'applications pour interagir avec les différentes ressources. Afin de rendre aisée l'intégration de nouveaux acteurs dans la plateforme, nous avons automatisé la génération des ressources REST à partir des entités.

Dans le but d'évaluer nos propositions, nous avons développé une preuve de concept de notre travail sur la base d'un exemple caractéristique de la ville intelligente. Cette preuve de concept nous a permis de montrer les étapes à respecter par les acteurs de la ville afin qu'ils utilisent la plateforme, qu'il s'agisse de développement d'applications, d'utilisation des modèles ou de déploiement de code dans l'infrastructure. De plus, la conception de cette preuve de concept constitue une première étape pour l'intégration de notre travail dans la plateforme FIWARE.

Le travail actuel pourrait faire l'objet d'un test utilisabilité afin d'être expérimenté sur des équipements réels. Les participants seraient des développeurs d'applications pour la ville intelligente qui doivent, sur la base de scénarios, superviser et contrôler des entités. Le but de ce test serait de mesurer le temps de prise en main et d'apprentissage de ces développeurs afin d'évaluer qualitativement notre travail. Cette expérimentation devrait être reproduite avec une mise en situation des participants dans le rôle d'un opérateur devant partager ses données des capteurs et les accès à ses actionneurs. Il s'agirait ici d'évaluer la partie de notre travail consacrée à la sélection, au paramétrage et au déploiement d'entités.

L'étape suivante de l'expérimentation consisterait à évaluer le fonctionnement de notre plateforme logiciel dans un environnement matériel réel de type expérimental : les capteurs et actionneurs sont réels, ils utilisent diverses technologies réseaux (4G, LoRa, Zigbee, etc.) et sont placés dans un environnement physique. L'objectif serait d'évaluer l'infrastructure que nous avons choisie dans notre travail, tel que le matériel edge, au travers de métriques quantitatives issues de divers tests (montée en charge, performance, robustesse, etc.).

8.2 Perspectives

8.2.1 Perspectives techniques

8.2.1.1 Généricité des modèles

L'un des objectifs de notre travail étant lié à l'ouverture de la plateforme, nous nous sommes intéressés à l'utilisation d'ontologies pour donner une base formelle et sémantique à cette interopérabilité. En effet, certains de nos choix techniques et de modélisation (cf. section 7.2.1) ont été prévus pour permettre l'intégration d'ontologies à notre travail. Pour rappel, les ontologies permettent de décrire la nomenclature ainsi que la signification de concepts appartenant à un domaine particulier (cf. section 2.3.1).

De ce fait, l'une des perspectives de notre travail serait de relier à des ontologies décrivant la ville intelligente, les entités et leurs états. Cependant, il s'agit d'un domaine de recherche à part entière (cf. section 2.3.2) qui va au-delà de notre travail.

A notre connaissance, une ontologie complète pour la ville intelligente n'existe pas. En revanche, de multiples ontologies ont été définies, relatives aux différents domaines de la ville (énergie, transport, etc.) telle que l'ontologie Common Information Model [141] dans le domaine de l'électricité. Nous avions également identifié l'existence de l'ontologie cityGML [91] qui ne décrit que des aspects géographiques de la ville. Ces ontologies devraient être utilisées conjointement et réconciliées ensemble là ou elles s'intersectent pour que nous puissions intégrer des méta-données sémantiques à notre travail.

Ainsi, nous pourrions tirer parti de ces méta-données sémantiques et informer les applications des conséquences d'un échec d'un processus de coordination (cf. section 6.3.1). Les applications pourraient également sélectionner les ressources qu'elles souhaitent superviser et contrôler en utilisant sur la base des méta-données qu'elles possèdent. Par exemple, une application pourrait choisir de superviser uniquement les entités qui sont des routes et qui sont dans une zone géographique spécifique. A un niveau de granularité plus fin, ces méta-données pourraient être exploitées afin que les applications prennent connaissance de la signification des états qu'elles obtiennent ou changent (cf. section 5.5.2.3). Elles n'auraient ainsi plus besoin de connaître à l'avance ce que signifient les différents états.

Il est à noter que d'un point de vue technique, ces méta-données sémantiques pourraient être fournies aux applications dans les représentations des ressources. Nous avons identifié que cela impliquerait d'utiliser un format supportant l'utilisation de ces méta-données tel que le format JSON-LD [162] ou Hydra [94]. Le format Hydra serait un choix judicieux puisqu'il est basé sur JSON-LD et comporte nativement une ontologie décrivant la sémantique des ressources REST (exemple : verbes supportés par une ressource). Cela nous permettrait, entre autre, d'indiquer aux applications les verbes supportés par la ressource utilisée pour effectuer des changements d'état (PUT et GET lorsqu'un état est commandable et seulement GET si l'état n'est pas commandable).

8.2.1.2 Intégration dans FIWARE

L'une des nos perspectives est d'intégrer notre travail dans la plateforme FIWARE.

Comme nous l'avons évoqué (cf. section 4.1.2.2), la plateforme FIWARE inclut du matériel edge qui peut être utilisé dans notre travail pour satisfaire les contraintes temps-réel des applications. L'enabler cible de FIWARE de l'intégration de notre travail se nomme "IoT Data Edge Consolidation (IDEC)" [139]. Nous avons déjà décrit en [138] comment cette intégration pourrait être effectuée dans FIWARE. Nous avons décrit une architecture fonctionnelle comportant nos travaux qui est compatible avec les briques logicielles présentes dans l'enabler cible, telles que le Complex Event Processing de la plateforme.

Une perspective plus lointaine serait d'inclure nos travaux dans une plateforme qui possède une infrastructure de "fog computing" [28]. Le fog computing peut être décrit comme le regroupement d'une infrastructure edge et cloud. Son objectif est d'offrir les mêmes avantages que le cloud computing(puissance de calcul, élasticité, etc.) et l'edge computing (faible latence, géo-distribution, etc.) en fournissant une infrastructure mêlant de manière homogène des serveurs et du matériel plus contraints tels que des passerelles, des routeurs, des devices, etc. Cependant, aucune plateforme opérationnelle ne possède à notre connaissance une infrastructure de fog computing au sens plein du terme. Il s'agit d'un type d'infrastructure encore à l'état de prototype.

8.2.1.3 Améliorations des solutions proposées

Bibliothèque de modèles d'entités La constitution de la bibliothèque de modèles d'entités cf. section 7.2.2.2) pourrait comporter des modèles génériques, que nous appelons des facettes, pouvant être composées ensemble. Ces facettes décriraient des comportements génériques que possèdent des éléments physiques.

L'objectif de ces facettes seraient d'être composées ensemble afin de faciliter la création de nouveaux modèles d'entités. Nous utiliserions les mécanismes de composition des automates pour automatiquement effectuer la composition des facettes choisies en un modèle.

Ces facettes seraient pré-définies et devraient être suffisamment génériques pour être réutilisées dans la création de plusieurs modèles différents. Ainsi, l'un des travaux à effectuer serait de répertorier les différents comportements communs à des éléments physiques.

L'une des facettes pourrait correspondre, par exemple, à décrire le comportement allumer ou éteint d'un élément physique. Cette facette serait un automate comportant les états ON et OFF. Lors de l'intégration d'une nouvelle entité possédant ce comportement, cette facette serait réutilisée. Elle pourrait être composée avec une autre facette, décrivant des modes de fonctionnement communs à des éléments physiques, afin d'obtenir un modèle d'entité.

Mécanisme de coordination Le mécanisme de coordination que nous avons proposé (cf. section 6.4) pourrait supporter d'autres opérations que des séquences de changement d'états successives.

Des mini-programmes pourraient être décrits par les applications et transmis au mécanisme de coordination afin que celui-ci les exécute atomiquement. Par exemple, une application pourrait indiquer de changer conditionnellement l'état d'une entité coordonnée, uniquement lorsque les autres entités coordonnées ont préalablement changé d'états avec succès. Pour parvenir à un tel mécanisme, nous devrions définir un DSL (Domain Specific Language) décrivant toutes les instructions qui ont du sens d'être utilisées par les applications et exécutées par le mécanisme de coordination. Pour la conception de ce DSL, nous pourrions nous inspirer de l'adaptation de BPEL pour REST qui a proposée [129].

Nous pourrions également généraliser la validation du mécanisme de notre mécanisme de coordination. Un cas d'usage généralisable ("petit monde") devrait être définie pour décrire exhaustivement toutes interactions pouvant exister entre les ressources, les processus de coordination et les applications. Des propriétés LTL devraient être également définies afin de prouver formellement notre algorithme. L'une de ces propriétés décrirait, par exemple, qu'une ressource ne peut jamais être enrôlée lorsque son token a été générée et est valide.

8.2.2 Perspectives générales

8.2.2.1 Place de la sécurité dans la ville intelligente

Comme nous l'avons évoqué au cours de cette thèse, la sécurité est une problématique importante qui doit être adressée dans la conception d'une plateforme horizontale.

Dans notre travail nous avons généré du code pour éviter des erreurs de programmation des modèles d'entités choisis dans la bibliothèque. Dans la conception de nos modèles, nous avons également cherché à protéger les équipements de mauvaises utilisations. Nous avons fait en sorte que les applications ne puissent pas effectuer malencontreusement d'acte malveillant, par exemple en monopolisant le token d'une entité contrôlable.

D'un point de technique, des choix pourraient également permettre de garantir une meilleure sécurité dans la ville intelligente. Il serait possible de sécuriser les différentes communications, qu'ils s'agissent des communications de la plateforme et des applications ou entre la plateforme et les capteurs/actionneurs. Il serait également possible d'identifier et d'authentifier les applications et de leur donner des niveaux de permission pour restreindre de plus ou moins leurs actions de contrôle et de supervision. Enfin, le matériel formant l'infrastructure cloud et edge pourrait être lui-même sécurisé pour protéger les données en cas de cyber-attaque.

Cependant, nous considérons que ces moyens techniques ne permettent pas de garantir un risque zéro d'acte malveillant, volontaire ou non, dans la ville intelligente.

A titre d'exemple, les feux d'un carrefour ne peuvent être tous verts en même

temps puisque l'entité du carrefour ne le permet pas. Cependant, il serait possible pour une personne mal-intentionné de se brancher physiquement sur les feux tricolores d'un carrefour afin de les positionner tous au vert. Similairement, du vandalisme pourrait rendre inopérants les capteurs et actionneurs présents dans la ville. Par conséquent, nous pensons que la sécurité dans la ville intelligente ne relève pas seulement d'une problématique technique et informatique. En effet, la sécurité implique un certain niveau de résistance à des actes malveillants qui n'est jamais totale.

8.2.2.2 Développement de la ville intelligente

Depuis le début de notre travail, le domaine de la ville intelligente et de l'internet des objets a largement évolué.

De nombreuses avancées ont été effectuées dans le domaine de l'IoT, nous en citons quelques unes. Des technologies réseaux telles que LoRa et Sigfox sont apparues et ont été déployées et expérimentées dans diverses villes. Par ailleurs, l'organisme de standardisation oneM2M créé en 2012 a publié une deuxième version des spécifications techniques relatives à l'architecture de référence pour le M2M et l'IoT qu'il propose.

De manière similaire à l'IoT, de nombreuses initiatives ont pris place autour des villes intelligentes dans le monde. A Paris, un appel à projet a été lancé autour de la ville intelligente [160]. En Inde, il est prévu à l'horizon 2029, le développement d'une centaine de villes intelligentes [159]. Dans la même optique, des villes entières pourvues de capteurs et actionneurs se construisent : Songdo en Corée du Sud, Masdar aux Emirats Arabes, etc.

L'évolution de l'IoT permet ainsi l'essor des villes intelligentes. Cependant, les nombreuses initiatives autour de la ville intelligente n'utilisent pas, à notre connaissance, de plateforme horizontale et fédératrice. Bien que nous ayons énuméré les bénéfices d'une telle plateforme, il reste des problématiques ne relevant pas du domaine scientifique et qui freinent son adoption.

Une problématique d'ordre économique se pose lorsqu'une plateforme horizontale est partagée. Un modèle économique devrait être défini pour que la plateforme soit économiquement viable auprès des différents acteurs de la ville. Dans l'idéal, la définition d'un modèle économique devrait avoir pour ligne directrice les bénéfices (environnementaux, sociétaux, énergétiques) qu'engendrent une telle plateforme plutôt que ses bénéfices financiers.

L'initiative de l'adoption d'une plateforme horizontale relève également d'un choix politique qui est guidé par la municipalité. En effet, la ligne directrice du développement d'une ville à l'autre n'est pas la même selon les affinités politiques de ses responsables.

Des questions de responsabilité se posent également lorsqu'une plateforme horizontale doit être déployée et maintenue. Par exemple, à qui incombe la responsabilité lorsque des actionneurs sont défaillants, pouvant avoir des conséquences sur les citoyens (par exemple, la défaillance d'un feu tricolore) ? Cela pourrait être po-

tentiellement la municipalité, l'acteur qui détient ces équipements, les acteurs qui ont ré-utilisé ces équipements, etc. De même, qui est responsable du matériel, edge par exemple, hébergeant la plateforme ? Les responsabilités de chaque acteur et de la municipalité vis-à-vis de la plateforme et des capteurs et actionneurs qui sont partagés devraient ainsi être préalablement définies.

Des problèmes sociétaux relatifs à la vie privée des citoyens et à la confidentialité peuvent également se poser. Bien que ces sujets aient été couverts en partie dans littérature [106, 35], il demeure des questions éthiques. Compte-tenu des révélations ces dernières années à propos du bafouement de la vie privée, une telle plateforme pourrait être vue comme un "Big Brother" par les citoyens. L'une des réponses serait d'impliquer les citoyens dans l'utilisation d'une plateforme horizontale afin que celle-ci ne soit plus vue comme une présence omnisciente. Cette implication pourrait se faire, par exemple, par l'utilisation du "participatory sensing" [33].

De manière générale, nous pensons qu'une plateforme horizontale est viable d'un point de vue scientifique et technologique. Cependant, une multitude de questions d'ordre économique, politique, juridique et éthique restent à résoudre selon nous afin que des plateformes horizontales puissent être déployées et effectivement utilisées.

Bibliographie

[1] Fiware european project. http://www.fiware.org. Accessed : 2016-11-01. (Cité en pages 40 et 56.)

[2] 3GPP. 3rd generation partnership project (3gpp) : Long term evolution - advanced (lte-a) presentation. http://www.3gpp.org/technologies/keywords-acronyms/97-lte-advanced. Accessed : 2016-03-28. (Cité en page 10.)

[3] 3GPP. 3rd generation partnership project (3gpp) : Long term evolution (lte) presentation. http://www.3gpp.org/technologies/keywords-acronyms/98-lte. Accessed : 2016-03-28. (Cité en page 10.)

[4] Mohamed Abomhara and Geir M Køien. Security and privacy in the internet of things : Current status and open issues. In *Privacy and Security in Mobile Systems (PRISMS), 2014 International Conference on*, pages 1–8. IEEE, 2014. (Cité en page 51.)

[5] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks : a survey. *Computer networks*, 38(4) :393–422, 2002. (Cité en page 10.)

[6] 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. (Cité en page 26.)

[7] M. B. Alaya, S. Medjiah, T. Monteil, and K. Drira. Toward semantic interoperability in onem2m architecture. *IEEE Communications Magazine*, 53(12) :35–41, Dec 2015. (Cité en page 44.)

[8] M. Ben Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira. Om2m : Extensible etsi-compliant m2m service platform with self-configuration capability. *Procedia Computer Science*, 32 :1079 – 1086, 2014. (Cité en page 43.)

[9] Enocean Alliance. http://www.enocean-alliance.org/, 2016. Accessed : 2017-03-29. (Cité en page 10.)

[10] LoRa Alliance. Lora alliance wide area networks for iot. https://www.lora-alliance.org/, 2016. Accessed : 2016-11-01. (Cité en pages 10 et 56.)

[11] OneM2M Alliance. Onem2m : Standards for m2m and the internet of things. http://www.onem2m.org/technical/published-documents, 2016. Accessed : 2016-11-01. (Cité en pages 7 et 43.)

[12] ZigBee Alliance et al. Zigbee specification, 2006. (Cité en page 10.)

[13] Almanac european project. http://www.almanac-project.eu/. Accessed : 2016-11-17. (Cité en page 40.)

[14] Arduino : an open-source electronics platform based on flexible, easy-to-use hardware and software. https://www.arduino.cc/. Accessed : 2016-11-15. (Cité en page 28.)

[15] Arm mbed iot device platform. https://www.mbed.com/. Accessed : 2016-11-15. (Cité en page 28.)

[16] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4) :50–58, 2010. (Cité en page 8.)

[17] Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, et al. Gene ontology : tool for the unification of biology. *Nature genetics*, 25(1) :25–29, 2000. (Cité en page 20.)

[18] Muhammad Atif. Analysis and verification of two-phase commit & three-phase commit protocols. In *Emerging Technologies, 2009. ICET 2009*, pages 326–331. IEEE, 2009. (Cité en page 123.)

[19] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things : A survey. *Computer networks*, 54(15) :2787–2805, 2010. (Cité en page 26.)

[20] Carolina Tripp Barba, Miguel Angel Mateos, Pablo Reganas Soto, Ahmad Mohamad Mezher, and M Aguilar Igartua. Smart city for vanets using warning messages, traffic statistics and intelligent traffic lights. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pages 902–907. IEEE, 2012. (Cité en page 24.)

[21] Sean Bechhofer. Owl : Web ontology language. In *Encyclopedia of Database Systems*, pages 2008–2009. Springer, 2009. (Cité en page 22.)

[22] Albert Benveniste and Gérard Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9) :1270–1282, 1991. (Cité en page 12.)

[23] Tim Berners-Lee and Dan Connolly. Notation3 (n3) : A readable rdf syntax. W3c team submission, W3C, 2011. http ://www.w3.org/TeamSubmission/n3/. (Cité en page 22.)

[24] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5) :28–37, 2001. (Cité en page 21.)

[25] Gérard Berry and Georges Gonthier. The esterel synchronous programming language : Design, semantics, implementation. *Science of computer programming*, 19(2) :87–152, 1992. (Cité en page 13.)

[26] Daniel Bimschas, Henning Hasemann, Manfred Hauswirth, Marcel Karnstedt, Oliver Kleine, Alexander Kröller, Myriam Leggieri, Richard Mietz, Alexandre Passant, Dennis Pfisterer, et al. Semantic-service provisioning for the internet of things. *Electronic Communications of the EASST*, 37, 2011. (Cité en page 38.)

[27] D. Bonino, M. T. D. Alizo, A. Alapetite, T. Gilbert, M. Axling, H. Udsen, J. A. C. Soto, and M. Spirito. Almanac : Internet of things for smart cities. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 309–316, Aug 2015. (Cité en page 40.)

[28] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012. (Cité en page 158.)

[29] Giacomo Brambilla, Marco Picone, Simone Cirani, Michele Amoretti, and Francesco Zanichelli. A simulation platform for large-scale internet of things scenarios in urban environments. In *Proceedings of the First International Conference on IoT in Urban Space*, pages 50–55. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014. (Cité en page 46.)

[30] T. Bray. The javascript object notation (json) data interchange format. RFC 7159, RFC Editor, March 2014. http://www.rfc-editor.org/rfc/rfc7159.txt. (Cité en page 127.)

[31] Tim Bray, Eve Maler, François Yergeau, Jean Paoli, and Michael Sperberg-McQueen. Extensible markup language (xml) 1.0 (third edition). W3c recommendation, W3C, feb 2004. http ://www.w3.org/TR/2004/REC-xml-20040204. (Cité en page 22.)

[32] Julien Bruneau, Wilfried Jouve, and Charles Consel. Diasim : A parameterized simulator for pervasive computing applications. In *Mobile and Ubiquitous Systems : Networking & Services, MobiQuitous, 2009. MobiQuitous' 09. 6th Annual International*, pages 1–10. IEEE, 2009. (Cité en page 46.)

[33] Jeffrey A Burke, Deborah Estrin, Mark Hansen, Andrew Parker, Nithya Ramanathan, Sasank Reddy, and Mani B Srivastava. Participatory sensing. *Center for Embedded Network Sensing*, 2006. (Cité en page 161.)

[34] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler. Assertions and protocols for the oasis security assertion markup language (saml) v2.0. *Oasis Standard*, Mar 2005. (Cité en page 41.)

[35] Quyet H Cao, Giyyarpuram Madhusudan, Reza Farahbakhsh, and Noel Crespi. Usage control for data handling in smart cities. In *Global Communications Conference (GLOBECOM), 2015 IEEE*, pages 1–6. IEEE, 2015. (Cité en page 161.)

[36] Gavin Carothers and Eric Prud'hommeaux. Rdf 1.1 turtle. W3c recommendation, W3C, 2014. http ://www.w3.org/TR/2014/REC-turtle-20140225/. (Cité en page 22.)

[37] Carriots : a platform as a service designed for internet of things and machine to machine projects. https://www.carriots.com. Accessed : 2016-11-01. (Cité en page 56.)

[38] Hafedh Chourabi, Taewoo Nam, Shawn Walker, José Ramón Gil-Garcia, Sehl Mellouli, Karine Nahon, Theresa A Pardo, and Hans Jochen Scholl. Understanding smart cities : An integrative framework. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 2289–2297. IEEE, 2012. (Cité en page 1.)

[39] M. Collina, G. E. Corazza, and A. Vanelli-Coralli. Introducing the qest broker : Scaling the iot by bridging mqtt and rest. In *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, pages 36–41, Sept 2012. (Cité en page 7.)

[40] Michael Compton, Payam Barnaghi, Luis Bermudez, RaúL GarcíA-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, et al. The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics : Science, Services and Agents on the World Wide Web*, 17 :25–32, 2012. (Cité en pages 22, 36 et 39.)

[41] Hyper consortium. Hypercat consortium. `http://www.hypercat.io/`, 2016. Accessed : 2016-11-01. (Cité en page 35.)

[42] Open Geospatial Consortium. Sensor web enablement standard. `http://www.opengeospatial.org/standards`, 2011. Accessed : 2016-11-11. (Cité en page 38.)

[43] Apache CXF. Open source services framework to build and develop rest, soap, corba services. `http://cxf.apache.org/`. Accessed : 2016-11-01. (Cité en page 132.)

[44] Datavenue, a internet of things platform from orange company. `https://datavenue.orange.com/fr`. Accessed : 2017-01-29. (Cité en page 31.)

[45] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462. IEEE, 2004. (Cité en page 37.)

[46] Eclipse. Eclipse californium framework : a java coap implementation to build servers and applications. `http://www.eclipse.org/californium/`. Accessed : 2016-11-01. (Cité en page 127.)

[47] Kapali P. Eswaran, Jim N Gray, Raymond A. Lorie, and Irving L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11) :624–633, 1976. (Cité en page 77.)

[48] Maria Fazio, Maurizio Paone, Antonio Puliafito, and Massimo Villari. Heterogeneous sensors become homogeneous things in smart cities. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 775–780. IEEE, 2012. (Cité en page 37.)

[49] Maria Fazio, Antonio Puliafito, and Massimo Villari. Iot4s : A new architecture to exploit sensing capabilities in smart cities. *International Journal of Web and Grid Services*, 10(2-3) :114–138, 2014. (Cité en page 37.)

[50] R. Fielding and J. Reschke. Hypertext transfer protocol (http/1.1) : Semantics and content. RFC 7231, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7231.txt. (Cité en page 6.)

[51] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. (Cité en pages 6 et 7.)

[52] Luca Filipponi, Andrea Vitaletti, Giada Landi, Vincenzo Memeo, Giorgio Laura, and Paolo Pucci. Smart city : An event driven architecture for monitoring public spaces with heterogeneous sensors. In *Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on*, pages 281–286. IEEE, 2010. (Cité en page 25.)

[53] Fitbit smart wristband. https://www.fitbit.com. Accessed : 2016-11-01. (Cité en page 30.)

[54] Friend of a friend (foaf) ontology. http://xmlns.com/foaf/spec/. Accessed : 2017-03-21. (Cité en page 21.)

[55] Frederico T Fonseca, Max J Egenhofer, Peggy Agouris, and Gilberto Câmara. Using ontologies for integrated geographic information systems. *Transactions in GIS*, 6(3) :231–257, 2002. (Cité en page 20.)

[56] Eclipse foundation. Eclipse : open source community of tools, projects and collaborative working groups. http://www.eclipse.org. Accessed : 2016-11-01. (Cité en pages 43 et 127.)

[57] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. (Cité en page 132.)

[58] Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening ontologies with dolce. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 166–181. Springer, 2002. (Cité en page 39.)

[59] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing : Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 45(5) :37–42, 2015. (Cité en page 8.)

[60] Hector Garcia-Molina and Kenneth Salem. Sagas. *SIGMOD Rec.*, 16(3) :249–259, December 1987. (Cité en pages 103 et 110.)

[61] David Garlan and Mary Shaw. An introduction to software architecture. *Advances in software engineering and knowledge engineering*, 1(3.4), 1993. (Cité en page 6.)

[62] Yanfeng Geng and Christos G Cassandras. New "smart parking" system based on resource allocation and reservations. *IEEE Transactions on Intelligent Transportation Systems*, 14(3) :1129–1139, 2013. (Cité en page 24.)

[63] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009. (Cité en page 51.)

[64] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy : An emerging low-power wireless technology. *Sensors*, 12(9) :11734–11753, 2012. (Cité en page 10.)

[65] Google. Gson : a java serialization/deserialization library that can convert java objects into json and back. `https://github.com/google/gson`. Accessed : 2016-11-01. (Cité en page 128.)

[66] Jim Gray et al. The transaction concept : Virtues and limitations. In *VLDB*, volume 81, pages 144–154, 1981. (Cité en page 110.)

[67] Jim Gray and Andreas Reuter. *Transaction Processing : Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1992. (Cité en pages 103 et 110.)

[68] Thomas R Gruber et al. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2) :199–220, 1993. (Cité en page 20.)

[69] 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. (Cité en page 26.)

[70] Dominique Guinard, Iulia Ion, and Simon Mayer. *In Search of an Internet of Things Service Architecture : REST or WS-* ? A Developers' Perspective*, pages 326–337. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. (Cité en page 8.)

[71] Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the web of things. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010. (Cité en page 8.)

[72] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, 15(4) :287–317, December 1983. (Cité en page 110.)

[73] Andrei Hagiu and Julian Wright. Multi-sided platforms. *International Journal of Industrial Organization*, 43 :162–174, 2015. (Cité en page 26.)

[74] Nicholas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9) :1305–1320, 1991. (Cité en page 13.)

[75] Nicolas Halbwachs. *Synchronous programming of reactive systems*, volume 215. Springer Science & Business Media, 2013. (Cité en page 11.)

[76] D Hardt. The oauth 2.0 authorization framework. RFC 6749, RFC Editor, October 2012. `http://www.rfc-editor.org/rfc/rfc6749.txt`. (Cité en pages 29 et 30.)

[77] David Harel and Amir Pnueli. On the development of reactive systems. In *Logics and models of concurrent systems*, pages 477–498. Springer, 1985. (Cité en page 11.)

[78] K. Hartke. Observing resources in the constrained application protocol (coap). RFC 7641, RFC Editor, September 2015. (Cité en pages 10 et 127.)

[79] Gerard Holzmann. *Spin Model Checker, the : Primer and Reference Manual*. Addison-Wesley Professional, first edition, 2003. (Cité en pages 15 et 118.)

[80] Gerard J. Holzmann. The model checker spin. *IEEE Transactions on software engineering*, 23(5) :279–295, 1997. (Cité en page 16.)

[81] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. Swrl : A semantic web rule language combining owl and ruleml. Technical report, W3C, 2004. http ://www.w3.org/TR/2004/REC-xml-20040204. (Cité en page 44.)

[82] Yunfei Hou, Yunjie Zhao, Aditya Wagh, Longfei Zhang, Chunming Qiao, Kevin F Hulme, Changxu Wu, Adel W Sadek, and Xuejie Liu. Simulation-based testing and evaluation tools for transportation cyber–physical systems. *IEEE Transactions on Vehicular Technology*, 65(3) :1098–1108, 2016. (Cité en page 46.)

[83] icore european project. http://www.iot-icore.eu/home, 2013. Accessed : 2016-11-07. (Cité en pages 42 et 56.)

[84] MYI Idris, YY Leng, EM Tamil, NM NOOR, and Z Razak. Car park system : a review of smart parking system and its technology. *Inf. Technol. J*, 8(2) :101–113, 2009. (Cité en page 24.)

[85] Internet of things - architecture (iot-a). european project. http://www.iot-a.eu/public/public-documents. Accessed : 2016-11-01. (Cité en page 56.)

[86] Jersey. Restful web services in java. https://jersey.java.net/. Accessed : 2016-11-01. (Cité en page 132.)

[87] Zhanlin Ji, Ivan Ganchev, Máirtín O'Droma, Li Zhao, and Xueji Zhang. A cloud-based car parking middleware for iot-based smart cities : design and implementation. *Sensors*, 14(12) :22372–22393, 2014. (Cité en page 24.)

[88] Stamatis Karnouskos and Thiago Nass De Holanda. Simulation of a smart grid city with software agents. In *Computer Modeling and Simulation, 2009. EMS'09. Third UKSim European Symposium on*, pages 424–429. IEEE, 2009. (Cité en page 46.)

[89] Aamod Khandekar, Naga Bhushan, Ji Tingfang, and Vieri Vanghi. Lte-advanced : Heterogeneous networks. In *Wireless Conference (EW), 2010 European*, pages 978–982. IEEE, 2010. (Cité en page 10.)

[90] Ganesh S Khekare and Apeksha V Sakhare. A smart city framework for intelligent traffic system using vanet. In *Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), 2013 International Multi-Conference on*, pages 302–305. IEEE, 2013. (Cité en page 24.)

[91] Thomas H. Kolbe, Gerhard Groger, and Lutz Plumer. Citygml - interoperable access to 3d city models. In *Proceedings of the first International Symposium*

on Geo-Information for Disaster Management, pages 21–23. Springer Verlag, 2005. (Cité en pages 60 et 157.)

[92] M. Kovatsch. Copper coap user-agent for firefox : generic browser for the internet of things based on the constrained application protocol (coap). `http://people.inf.ethz.ch/mkovatsc/copper.php`. Accessed : 2016-11-01. (Cité en page 127.)

[93] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4) :18–25, 2001. (Cité en page 150.)

[94] Markus Lanthaler and Christian Gütl. Hydra : A vocabulary for hypermedia-driven web apis. In *Proceedings of the 6th Workshop on Linked Data on the Web (LDOW2013) at the 22nd International World Wide Web Conference (WWW2013)*. CEUR-WS, 2013. (Cité en pages 128 et 157.)

[95] Paul Le Guernic, Albert Benveniste, Patricia Bournai, and Thierry Gautier. Signal–a data flow-oriented language for signal processing. *IEEE transactions on acoustics, speech, and signal processing*, 34(2) :362–374, 1986. (Cité en page 13.)

[96] Rodger Lea and Michael Blackstock. City hub : A cloud-based iot platform for smart cities. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 799–804. IEEE, 2014. (Cité en pages 35 et 58.)

[97] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, et al. Tinyos : An operating system for sensor networks. In *Ambient intelligence*, pages 115–148. Springer, 2005. (Cité en page 37.)

[98] Rhys Lewis. Dereferencing http uris. *Draft Tag Finding (May 31, 2007 (retrieved July 25, 2007), http ://www. w3. org/2001/tag/doc/httpRange-14/2007-05-31/HttpRange-14. html*, 2007. (Cité en page 132.)

[99] Fengjun Li, Bo Luo, and Peng Liu. Secure information aggregation for smart grids using homomorphic encryption. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pages 327–332. IEEE, 2010. (Cité en page 51.)

[100] Wenbin Li and Gilles Privat. Cross-fertilizing data through web of things apis with JSON-LD. In *Proceedings of the 4th Workshop on Services and Applications over Linked APIs and Data co-located with the 13th Extended Semantic Web Conference (ESWC 2016), Crete, Greece, May 29, 2016.*, 2016. (Cité en page 60.)

[101] Meshlium gateways : products from libellium company. `www.libelium.com/products/meshlium/`. Accessed : 2016-11-01. (Cité en page 61.)

[102] Lontalk wired protocol specification. `http://www.echelon.com/technology/lonworks/protocol/`. Accessed : 2017-03-24. (Cité en page 10.)

[103] Alexander Maedche and Steffen Staab. Ontology learning for the semantic web. *IEEE Intelligent systems*, 16(2) :72–79, 2001. (Cité en page 22.)

[104] F. Maraninchi and Y. Rémond. Argos : an automaton-based synchronous language. *Computer Languages*, 27(27) :61–92, 2001. (Cité en page 13.)

[105] Florence Maraninchi. Operational and compositional semantics of synchronous automaton compositions. In *International Conference on Concurrency Theory*, pages 550–564. Springer, 1992. (Cité en page 14.)

[106] Antoni Martinez-Balleste, Pablo A Pérez-Martínez, and Agusti Solanas. The pursuit of citizens' privacy : a privacy-aware smart city is possible. *IEEE Communications Magazine*, 51(6) :136–141, 2013. (Cité en page 161.)

[107] Carlo Maria Medaglia and Alexandru Serbanati. An overview of privacy and security issues in the internet of things. In *The Internet of Things*, pages 389–395. Springer, 2010. (Cité en page 51.)

[108] Maciej Mendalka, Michal Gadaj, Lukasz Kulas, and Krzystof Nyka. Wsn for intelligent street lighting system. In *Information technology (ICIT), 2010 2nd international conference on*, pages 99–100. IEEE, 2010. (Cité en page 25.)

[109] Giovanni Merlino, Dario Bruneo, Salvatore Distefano, Francesco Longo, Antonio Puliafito, and Adnan Al-Anbuky. A smart city lighting case study on an openstack-powered infrastructure. *Sensors*, 15(7) :16314–16335, 2015. (Cité en page 58.)

[110] Julien Mineraud, Oleksiy Mazhelis, Xiang Su, and Sasu Tarkoma. A gap analysis of internet-of-things platforms. *CoRR*, abs/1502.01181, 2015. (Cité en page 8.)

[111] 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. (Cité en page 26.)

[112] Modbus wired protocol specifications. `http://www.modbus.org/specs.php`. Accessed : 2017-03-24. (Cité en page 10.)

[113] Mahshid Helali Moghadam and Nasser Mozayani. A street lighting control system based on holonic structures and traffic system. In *Computer Research and Development (ICCRD), 2011 3rd International Conference on*, volume 1, pages 92–96. IEEE, 2011. (Cité en page 25.)

[114] Tim Moses et al. Extensible access control markup language (xacml) version 2.0. *Oasis Standard*, 200502, 2005. (Cité en page 41.)

[115] Reinhard Müllner and Andreas Riener. An energy efficient pedestrian aware smart street lighting system. *International Journal of Pervasive Computing and Communications*, 7(2) :147–161, 2011. (Cité en page 25.)

[116] Taewoo Nam and Theresa A Pardo. Conceptualizing smart city with dimensions of technology, people, and institutions. In *Proceedings of the 12th Annual International Digital Government Research Conference : Digital Government Innovation in Challenging Times*, pages 282–291. ACM, 2011. (Cité en page 1.)

[117] nCoAP. Java implementation of the coap protocol using netty framework. `https://github.com/okleine/nCoAP`. Accessed : 2016-11-01. (Cité en page 132.)

[118] Netatmo smart home : smart thermostat, weather stations and security camera. https://www.netatmo.com. Accessed : 2016-11-01. (Cité en pages 30 et 55.)

[119] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 2–9. ACM, 2001. (Cité en page 22.)

[120] OASIS. Web services business process execution language version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007. Accessed : 2016-11-01. (Cité en page 114.)

[121] OASIS. Message queuing telemetry transport version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html, 2014. Accessed : 2016-03-25. (Cité en page 10.)

[122] Open mobile alliance - device management. http://www.openmobilealliance.org/wp/overviews/dm_overview.html. Accessed : 2017-02-21. (Cité en pages 44 et 51.)

[123] Open mobile alliance - next generation services interface. http://www.openmobilealliance.org/release/NGSI/. Accessed : 2017-02-26. (Cité en page 40.)

[124] oneM2M. Onem2m specification : Coap protocol binding. http://www.onem2m.org/images/files/deliverables/TS-0008-CoAP_Protocol_Binding-V1_3_2.pdf, 2016. Accessed : 2016-03-27. (Cité en page 7.)

[125] oneM2M. onem2m specification : Http protocol binding. http://www.onem2m.org/images/files/deliverables/Release2/TS-0009-HTTP_Protocol_Binding-V2_6_1.pdf, 2016. Accessed : 2016-03-27. (Cité en page 7.)

[126] oneM2M. Onem2m specification : Mqtt protocol binding. http://www.onem2m.org/images/files/deliverables/Release2/TS-0010-MQTTProtocolBinding-V2_4_1.pdf, 2016. Accessed : 2016-03-27. (Cité en page 7.)

[127] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, 2014. (Cité en page 150.)

[128] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 641–648, Nov 2006. (Cité en pages 38 et 46.)

[129] Cesare Pautasso. Bpel for rest. In *International Conference on Business Process Management*, pages 278–293. Springer, 2008. (Cité en page 159.)

[130] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big"' web services : Making the right architectural decision. In *Proceedings*

of the 17th International Conference on World Wide Web, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM. (Cité en page 8.)

[131] Charith Perera, Arkady B. Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Sensing as a service model for smart cities supported by internet of things. *CoRR*, abs/1307.8198, 2013. (Cité en page 36.)

[132] Dewayne E Perry and Alexander L Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4) :40–52, 1992. (Cité en page 6.)

[133] Riccardo Petrolo, Valeria Loscrí, and Nathalie Mitton. Towards a smart city based on cloud of things. In *Proceedings of the 2014 ACM International Workshop on Wireless and Mobile Technologies for Smart Cities*, WiMobCity '14, pages 61–66, New York, NY, USA, 2014. ACM. (Cité en page 36.)

[134] Riccardo Petrolo, Valeria Loscri, and Nathalie Mitton. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Transactions on Emerging Telecommunications Technologies*, pages 1–11, 2015. (Cité en page 36.)

[135] Dennis Pfisterer, Kay Römer, Daniel Bimschas, Oliver Kleine, Richard Mietz, Cuong Truong, Henning Hasemann, Alexander Kröller, Max Pagel, Manfred Hauswirth, et al. Spitfire : toward a semantic web of things. *IEEE Communications Magazine*, 49(11) :40–48, 2011. (Cité en page 38.)

[136] Philips hue wireless lighting system. http://www.hue.philips.frs. Accessed : 2016-11-01. (Cité en pages 30 et 55.)

[137] Mircea Popa and Costin Cepişcă. Energy consumption saving solutions based on intelligent street lighting control system. *UPB Sci. Bull*, 73 :297–308, 2011. (Cité en page 25.)

[138] G. Privat, L. Lemke, P. Borscia, and M Capdevielle. Edge-of-cloud fast-data consolidation for the internet of things. In *Innovations in Clouds, Internet and Networks Conference , Nineteenth Conference on*. IFIP, 2016. (Cité en page 158.)

[139] FIWARE project. Iot data edge consolidation (idec) generic enabler. https://catalogue.fiware.org/enablers/iot-data-edge-consolidation-ge-cepheus, 2016. Accessed : 2016-04-10. (Cité en page 158.)

[140] Eclipse IoT projects. Eclipse iot : open source iot projects. https://iot.eclipse.org/. Accessed : 2016-04-01. (Cité en page 43.)

[141] Stephen Quirolgico, Pedro Assis, Andrea Westerinen, Michael Baskey, and Ellen Stokes. Toward a formal common information model ontology. In *International Conference on Web Information Systems Engineering*, pages 11–21. Springer, 2004. (Cité en page 157.)

[142] Benjamin Reed and Flavio P Junqueira. A simple totally ordered broadcast protocol. In *proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*, page 2. ACM, 2008. (Cité en page 150.)

[143] RESTeasy. Jboss project to build restful web services and restful java applications. http://resteasy.jboss.org/. Accessed : 2016-11-01. (Cité en page 132.)

[144] Vinny Reynolds, Vinny Cahill, and Aline Senart. Requirements for an ubiquitous computing simulation and emulation environment. In *Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, page 1. ACM, 2006. (Cité en page 46.)

[145] Leonard Richardson and Sam Ruby. *Restful Web Services*. O'Reilly, first edition, 2007. (Cité en pages 7, 70 et 93.)

[146] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software : Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, 4(2) :14, 2009. (Cité en page 51.)

[147] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, et al. Smartsantander : Iot experimentation over a smart city testbed. *Computer Networks*, 61 :217–238, 2014. (Cité en pages 45 et 60.)

[148] Eduardo Felipe Zambom Santana, Daniel Macêdo Bastista, Fabio Kon, and Dejan S Milojicic. Scsimulator : An open source, scalable smart city simulator. In *Tools Session of the 34th Brazilian Symposium on Computer Networks (SBRC). Salvador, Brazil*, 2016. (Cité en page 46.)

[149] Chayan Sarkar, SN Akshay Uttama Nambi, R Venkatesha Prasad, and Abdur Rahim. A scalable distributed architecture towards unifying iot applications. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 508–513. IEEE, 2014. (Cité en page 42.)

[150] Floriano Scioscia and Michele Ruta. Building a semantic web of things : Issues and perspectives in information compression. In *Proceedings of the 2009 IEEE International Conference on Semantic Computing*, ICSC '09, pages 589–594, Washington, DC, USA, 2009. IEEE Computer Society. (Cité en page 38.)

[151] Mary Shaw and David Garlan. *Software architecture : perspectives on an emerging discipline*, volume 1. Prentice Hall Englewood Cliffs, 1996. (Cité en page 6.)

[152] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol. RFC 7252, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7252.txt. (Cité en pages 7 et 127.)

[153] Zach Shelby and Carsten Bormann. *6LoWPAN : The wireless embedded Internet*, volume 43. John Wiley & Sons, 2011. (Cité en page 10.)

[154] Pavel Shvaiko and Jérôme Euzenat. Ontology matching : state of the art and future challenges. *IEEE Transactions on knowledge and data engineering*, 25(1) :158–176, 2013. (Cité en page 20.)

[155] Sabrina Sicari, Alessandra Rizzardi, Alberto Coen-Porisini, Luigi Alfredo Grieco, and Thierry Monteil. Secure om2m service platform. In *Autonomic*

Computing (ICAC), 2015 IEEE International Conference on, pages 313–318. IEEE, 2015. (Cité en page 44.)

[156] Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, and Alberto Coen-Porisini. Security, privacy and trust in internet of things : The road ahead. *Computer Networks*, 76 :146–164, 2015. (Cité en pages 27 et 51.)

[157] Sigfox cellular network operator for the internet of things. https://www.sigfox.com. Accessed : 2016-11-01. (Cité en pages 10 et 56.)

[158] Amir Sinaeepourfard, Jordi Garcia, Xavier Masip-Bruin, Eva Marín-Tordera, Jordi Cirera, Glòria Grau, and Francesc Casaus. Estimating smart city sensors data generation. In *2016 Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pages 1–8. IEEE, 2016. (Cité en pages 45 et 60.)

[159] Smart cities mission, development of 100 smart cities in india. http://www.smartcities.gov.in/, 2016. Accessed : 2016-04-07. (Cité en page 160.)

[160] Smart paris 2024 - smart city. https://smartparis2024.com/smartcity, 2016. Accessed : 2016-04-07. (Cité en page 160.)

[161] Michael Smethurst, Tom Scott, and Styles Rob. Places : a simple lightweight ontology for describing places of geographic interest. http://purl.org/ontology/places. (Cité en page 41.)

[162] Manu Sporny, Gregg Kellogg, Markus Lanthaler, W3C RDF Working Group, et al. Json-ld 1.0 : a json-based serialization for linked data. *W3C Recommendation*, 16, 2014. (Cité en pages 127 et 157.)

[163] Vagan Terziyan, Olena Kaykova, and Dmytro Zhovtobryukh. Ubiroad : Semantic middleware for context-aware smart road environments. In *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*, pages 295–302. IEEE, 2010. (Cité en page 24.)

[164] Thingspeak : open data platform for the internet of things. https://www.thingworx.com. Accessed : 2016-11-01. (Cité en pages 29 et 56.)

[165] Broadband forum : Technical report 069 (cpe wan management protocol). https://www.broadband-forum.org/standards-and-software/technical-specifications/tr-069-files-tools. Accessed : 2017-02-21. (Cité en page 51.)

[166] Bernard Vatant and Marc Wick. Geonames : an ontology to provide elements of description for geographical features. http://www.geonames.org/ontology/documentation.html, 2012. Accessed : 2016-11-17. (Cité en page 40.)

[167] Vital european project. http://vital-iot.eu/project. Accessed : 2016-11-01. (Cité en pages 36 et 56.)

[168] Panagiotis Vlacheas, Raffaele Giaffreda, Vera Stavroulaki, Dimitris Kelaidonis, Vassilis Foteinos, George Poulios, Panagiotis Demestichas, Andrey Somov, Abdur Rahim Biswas, and Klaus Moessner. Enabling smart cities through a

cognitive management framework for the internet of things. *Communications Magazine, IEEE*, 51(6) :102–111, 2013. (Cité en page 42.)

[169] Hongwei Wang and Wenbo He. A reservation-based smart parking system. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 690–695. IEEE, 2011. (Cité en page 24.)

[170] Barcelona to deploy worldsensing's smart parking system fastprk. http://www.worldsensing.com/news-press/barcelona-to-deploy-worldsensings-smart-parking-system-fastprk.html. Accessed : 2017-01-29. (Cité en page 33.)

[171] Worldsensing and sigfox announce the world's largest intelligent parking deployment with micronet, the sigfox network operator for russia. http://www.worldsensing.com/news-press/press-release-worldsensing-and-sigfox-announce-the-worlds-largest-intelligent-parking-deployment-with-micronet-the-sigfox-network-operator-for-russia.html. Accessed : 2017-01-29. (Cité en page 33.)

[172] Xively : Internet of things platform and application solution to connect products and services. https://www.xively.com. Accessed : 2016-11-01. (Cité en pages 28 et 56.)

[173] Andrea Zanella, Nicola Bui, Angelo P Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 2014. (Cité en pages 45 et 60.)