# Modèle Multi-Agent de gouvernance de réseau Machine-to-Machine pour la gestion intelligente de place de parking

Mustapha Bilal, Camille Persson, Fano Ramparany, Gauthier Picard, Olivier Boissier

▶ **To cite this version:**

Mustapha Bilal, Camille Persson, Fano Ramparany, Gauthier Picard, Olivier Boissier. Modèle Multi-Agent de gouvernance de réseau Machine-to-Machine pour la gestion intelligente de place de parking. 2011. hal-00919640

**HAL Id: hal-00919640**
**https://hal.science/hal-00919640**

Submitted on 20 Dec 2013

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Modèle Multi-Agent de gouvernance de réseau Machine-to-Machine pour la gestion intelligente de place de parking

Mustapha Bilal[1], Camille Persson[12], Fano Ramparany[1], Guathier Picard[2] and Olivier Boisier[2]
1 : Orange Labs Network and Carrier, TECH/MATIS Grenoble, France {firstname.lastname}@orange.com
2 : Fayol Institute, ISCOD, ENSM-SE, Saint-Etienne, France {lastname}@emse.fr

## 1.2 Contexte et Problématique

### 1.2.1 Problématique de la gestion de parking

Les villes modernes ont fortement besoin de systèmes avancés pour l'assistance au stationnement et de systèmes de transport intelligents donnant aux conducteurs des informations sur le parc de stationnement. Les systèmes d'information de stationnement ignorent généralement plusieurs facteurs et critères du parking. En effet ils ne fournissent pas automatiquement la place de parking optimale qui correspond à la demande des conducteurs.

Les métropoles du monde entier attirent les populations pour y vivre, travailler et visiter, en offrant des styles de vie différents, des possibilités d'emploi, de meilleures ressources et plus de services publics. Comme la modernisation de la ville progresse, le nombre de véhicules augment en conséquence. Au lieu de prendre les transports publics, les personnes voyagent dans leurs véhicules personnels pour plus de commodité et de confort. En raison de l'absence d'une politique bien planifiée pour les installations de stationnement, la demande de places de stationnement est généralement beaucoup plus grande que l'offre. En outre, les zones du centre-ville sont progressivement saturées avec des immeubles de bureaux commerciaux, mais sans avoir des places de parking suffisantes. Les conducteurs passent généralement leurs temps à circuler dans les blocs autour de leur destination pour chercher une place libre ou pour attendre qu'une place se libère. Ces phénomènes ne touchent pas seulement à l'efficacité des activités économiques, mais aussi augmentent l'impact social physique et émotionnel pour les voyageurs urbains.

Trouver une place de stationnement libre est un problème courant commun dans la plupart des grandes villes qui se produit surtout dans les endroits populaires comme les centres commerciaux, les stades et les points d'attraction touristique. Cette situation est devenue plus grave, surtout, pendant les heures de pointe, les saisons de vacances, carnavals et festivals. Ce problème se pose car, la plupart du temps les personnes viennent avec leurs propres moyens de transport individuels, causant, une concurrence importante pour occuper les places de stationnement vacantes. La disponibilité limitée des places de

stationnement vacantes aboutit souvent à la congestion du traffic, ainsi qu'à la frustration du conducteur incapable de trouver une place de parking disponible. En fait, c'est l'une des principales causes de la congestion du traffic. De plus l'émission importante de $CO_2$ est aggravée par la circulation pour trouver une place libre.

Nous allons traiter les problèmes mentionnés ci-dessus en proposant un système de guidage et de réservation de parking. Nous utilisons pour cela la plateforme SensCity en y intégrant l'approche système multi-agents est décrite plus loin dans la section 4.2 pour aider à résoudre la congestion, diminuer l'émission de $CO_2$, optimiser le temps pris pour trouver une place libre et avoir un système qui corresponds totalement aux demandes de chaque conducteur.

En effet, c'est un moyen de réduire la circulation inutile et les rejets de $CO_2$. Une partie de la circulation urbaine est dûe uniquement à la recherche de places disponibles. Sans atteindre les 30% souvent cités qui s'appliquent uniquement à certaines grandes villes comme New York, la part de la circulation urbaine engendrée par les véhicules en recherche de stationnement se situerait entre 5 et 10% [1].

Cette même enquête a estimé le temps moyen perdu par usager à 3.3 minutes dans le quartier Vaucanson de Grenoble, 11.8 minutes à Lyon (quartier de la Presquîle), 10 minutes dans la zone de Paris quartier du Commerce et 17.7 minutes dans la zone de Paris quartier Saint-Germain. Dans des cas extrêmes, l'automobiliste peut même renoncer à son déplacement, 64% des résidents interrogés dans les 4 zones étudiées déclarent qu'il leur est déjà arrivé de renoncer à un déplacement après avoir cherché sans succès une place de stationnement. Les disparités entre les zones sont fortes, ce chifre est de 48% à Grenoble, de 67% à Lyon et de 100% dans les deux zones parisiennes. Par ailleurs, la recherche d'une place de stationnement induit les mêmes nuisances que la circulation automobile : bruit, insécurité, pollution atmosphérique, pollution par émission de gaz à effet de serre et congestion.

Toutes les personnes qui ont un jour passé des heures à chercher une place de parking, pour finalement se retrouver très loin de leur destination, ont sûrement rêvé d'un appareil capable de leur indiquer les places disponibles. Ce souhait sera bientôt réalisé grâce à Orange.

### 1.2.2 Projet SensCity : une architecture Machine-to-Machine pour le développement durable

Le projet SensCity a l'ambition de se mettre au service du développement durable en proposant aux collectivités territoriales des dispositifs tels la télèrelève ou la télésurveillance comme moyen de réduire leur consommation de ressources (distribution de l'eau, éclairage public) et d'assumer leur responsabilité environnementale.

Machine-to-Machine (M2M) se réfère à des technologies qui permettent à la fois aux systèmes sans fil et filaire de communiquer avec d'autres appareils de la même capacité. M2M utilise un appareil (comme un capteur ou d'un compteur) pour capturer un événement (comme la température, l'inventaire de niveau, etc), qui est relayé par un réseau (sans fil, filaire ou hybride) à une application (logiciel), qui traduit l'événement capturé en informations significatives

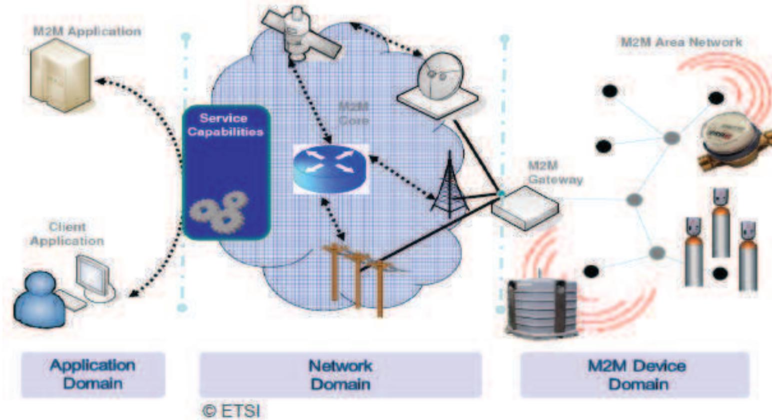
Toutefois, les communications modernes de M2M ont élargi au-delà d'une connexion un à un et transformé en un système de réseaux qui transmet des données à des appareils personnels. L'expansion des réseaux sans fil à travers le monde en a fait beaucoup plus facile pour les communications M2M à prendre place et a diminué la quantité d'énergie et le temps nécessaire pour l'information à communiquer entre les machines.

Cependant, le comité technique M2M de l'ETSI travaille sur les standards pour ces infrastructures. La figure 1.2 montre une telle architecture divisée en trois domaines : *device*, *réseau* et *application*. Dans cette architecture les passerelles ( *gateways*) et la plateforme centrale font le lien entre deux domaines.

Orange Labs développe un réseau de capteurs destinés à être installés sous les places de parking, qui permettra aux automobilistes de connaître les places disponibles. Ce projet s'inscrit dans un programme plus vaste destiné à favoriser le développement durable des villes grâce aux technologies M2M (ou Machine-to-Machine). Ce programme se fonde sur l'association des technologies de l'information et de la communication, avec des objets intelligents et communicants, dans le but de donner à ces derniers les moyens d'interagir sans intervention humaine avec le système d'information d'une organisation ou d'une entreprise [3].

Le projet dans sa globalité s'intitule *SensCity*. L'objet de ce projet est l'analyse des technologies permettant de dégager la solution d'infrastructure de réseau de capteurs urbains et la création d'une in-

FIGURE 1.2 – Une architecture Machine-to-Machine telle que définie par l'ETSI [ref ETSI][2].




frastructure de réseau opérable. Cette infrastructure répondra aux besoins de sociétés tierces spécialisées qui souhaitent faire communiquer des équipements de types capteurs ou machines pour des services dans les agglomérations et pour les citadins.

Elle permet le développement de services basés sur les données remontées par ces équipements ou la capables de les commander à distance pour dégager de la valeur auprès des collectivités locales. L'infrastructure ainsi déployée doit être respectueuse de l'environnement urbain et doit s'inscrire dans une démarche forte de développement durable.

L'un des services identifiés a pour but de pallier au problème de recherche de places de parking en s'inscrivant dans cette optique de développement durable.

Le projet est basé sur des Capteurs et services pour la ville durable. Projet de pôle de compétitivité visant la création d'un écosystème des services M2M à l'échelle de la ville par le moyen de réseaux opérables adaptés aux capteurs et équipements urbains.

La platforme M2M est un middleware qui fournit à des applications un accès aux *devices* de manière transparente. Elle découvre automatiquement les *devices* et leurs capacités afin de réduire les temps d'installation et d'administration de *devices* et expose leurs capacités en tant que Web Services. Ces services permettent à des fournisseurs de services M2M –clients de la plateforme– de construire des applications M2M pour leurs clients finaux, simplement et sans soucis de déploiement.

La plateforme SensCity se décompose en deux couches distinctes (voir Figure 4.1) : l'*Application Platform (USP)* et *Urban Collect & Command Platform (UCCP)*. La première couche interagit avec les applications clientes pour leur fournir un accès transparent avec les *devices*. La seconde gère les communications avec ces mêmes *devices* via des passerelles GSM.

Une telle architecture permet le déploiement d'un grand nombre de capteurs et d'effecteurs dans le monde physique. Elle semble donc appropriée pour une application telle que la gestion de parc de stationnement qui doit surveiller et agir directement dans l'environnement urbain.

# Chapitre 2

# État de l'art

La problématique de gestion de parking soulève plusieurs problèmes . Nous étudions ici les différents travaux liés à cette problématique.

## 2.1 Surveillance de places de parking

Le problème de gestion de parking est devenue un centre d'intérêt pour plusieurs entités de la société qui s'occupent des citoyens et même de la nature. Plusieurs travaux ont abordé ce point pour résoudre d'une manière ou d'autre le problème de trouver une place de parking. Les chercheurs dans ce domaine ont utilisé différentes technologies afin de trouver un moyen efficace qui s'inscrivent aux attentes de tout le monde et qui va constituer une solution qui touche tous les cotés du problème traité.

*Approches non-agent*

À cet effet, une approche possible consiste en un système de survaillance de parking en utilisant les caméras avec la technique de traitement de l'image [4] . Les caméras de surveillance et de sécurité pré-existantes (CCTV) sont utilisées comme capteurs pour identifier les places de stationnement vacantes et l'image capturée est traitée par le microcontrôleur RabbitCore et les données traitées sont transmises via ZigBee à un ordinateur central pour stocker et mettre à jour le statut d'occupation des places disponibles dans la base de données.

Outre cela, les CCTV permettent de détecter l'arrivée d'une voiture à son espace de stationnement attribué précédemment et changer le statut de cet emplacement de stationnement du "Réservé" à "Occupés'. Dans le cas qu'un véhicule garé d'ailleurs autre que son espace assigné, le statut de l'espace vacant prise par le conducteur sera également mis à jour par "Occupé" comme détecté par le CCTV. Un micro-contrôleur RabbitCore de traitement d'image est connecté à CCTV pour la détection des palces vacantes et les données traitées sont transférée via le réseau de capteurs sans fil Zigbee (WSN) à l'ordinateur central pour actualiser les informations. Le calcul est effectué sur l'appareil micro-contrôleur dû au fait que le traitement de l'image nécessite une puissance de calcul intensive pour exécuter les algorithmes de traitement d'image, cela évite d'envoyer de gros volumes d'images brutes ce qui la rend plus efficace.

Le système SPARK [5] offre une approche différente dans laquelle les voitures détectent les places libres en circulant et les communiquant par un réseau VANET (voiture à voiture). Le système SPARK base à avoir une "Trusted Authority", (TA), "On Board Unit" (OBU) qui se trouvent dans tous les véhicules,"RoadSide Units" (RSU) pour surveiller et gérer le stationnement en utilisant la technologie de communication VANET. Les OBU se communiquent ensemble et avec les RSU pour assurer des informations importantes sur le traffic et sur les places de stationnement où les RSU contiennent des informations sur les véhicules et les places qui sont libres dans leurs portés. L'espace de stationnement est une ressource spatio-temporel enregistré par les RSU dans une zone de parking. Chaque enregistrement d'une place de parking contient l'attribut POS (position) où chaque place de parking peut tirer sa position (xi, yi) sur le plan euclidien unique déterminé par les différents RSU qui gèrent une zone de parking spécifique, l'attribut RES (Réservation) qui représente le statut de réservation d'une place de parking, si la place est réservée, RES = 1, sinon, RES = 0, l'attribut Occupation (OCC) qui désigne le statut d'occupation d'une place de parking, si la place est occupée, l'OCC = 1, sinon,l'OCC = 0, l'attribut PID (Pseduo-ID) : Si la place est occupée par une OBU, ce champ enregistre le pseduo-ID de

l'OBU, l'attribut TID (ID Ticket) : Si la place est occupée par une OBU, ce champs enregistre ticket ID de l'OBU, l'attribut TKey (Ticket Key) : Si la place est occupée par une OBU, ce champ enregistre le Ticket Key de l'OBU, l'atribut ST (Start Time) qui enregistre le temps de l'OBU au moment de stationnement dans la place de parking, l'attribut LUT (Last Update Time) qui enregistre l'horodateur à laquelle l'OBU envoie les derniers messages.

Dans une zone de stationnement, puisque tous les données et informations des places de parking sont stocké dans le RSU. Ce dernier peut gérer facilement l'ensemble du parc de stationnement en utilisant ces données.

Un autre système appelé SPARK [6] propose une approche basée dur un réseau de capteur sans fil (WSN). Le sous-sytème WSN traite principalement la surveillance de statut des places de parking. Ce sous-système détecte le statut de place de parking avec une technique hybride de détection et ils transmet l'information sur le statut par une fréquence radio (RF). Il reçoit également des commandes du sous-système de gestion du stationnement pour effectuer de diverses procédures. Le sous-système WSN est composé en interne de quatre modules importantes qui comprends un module de détection, unmodule de routage, un module de diffusion et un module de statut. Un autre sous-système qui entre en jeux qui est le sous-système Sink, il recueille le rapport sur la situation de stationnement de sous-système de WSN et l'envoie au sous-système de gestion du stationnement. Il agit comme une passerelle entre le réseau de capteurs sans fil et les réseaux externes. Ce sous-système transmet également les informations concernant le changement de statut de stationnement reçue de sous-système de gestion de stationnement au sous-système de guidage à travers le WiFi / Bluetooth / RF. Le sous-système WSN récupère le statut (occupé / vacant) de la place de stationnement et transmet des messages à travers l'unité de communication RF au sous-système de la Sink et change le statut d'affichage. Le sous-système Sink traite les messages reçus du sous-système de WSN et envoie les données traitées vers le sous-système de gestion par Série / USB / Ethernet. Cette information est stockée dans le module de base de données du sous-système de gestion. Donc des nœuds capteurs sont utilisés et qui s'agit du plus bas niveau du système et sont fournies par des noeuds de capteurs autonomes. Ces petits appareils alimentés par piles sont placées dans les zones d'intérêt. Ces nœuds détectent la présence de voitures dans les places en utilisant les capteurs de lumière qui sont attachés à chaque capteur et envoient l'événement au sous-système SINK à travers la communication RF. Les LED attachées avec des nœuds fournissent également des informations d'état concernant les places. Le nœud Sink est de capteurs individuels qui communiquent et coordonnent l'un avec l'autre. Les nœuds de capteurs prends généralement la forme d'un réseau multi-sauts en transmettant les messages des autres, ce qui élargit considérablement les options de connectivité. Finalement, les données de chaque noeud du capteur sont propagées vers le nœud passerelle / Sink.

Une autre approche est utilisée dans [7] pour surveiller les places libres dans la ville ou dans une zone de parking. L'idée se base que chaque véhicule est déployé avec un transmetteur sans fil de courte portée et d'un processeur simple. La portée de transmission est de environ de 1m et il peut être l'un des dispositifs actuels de courte portée, tels que les dispositifs ZigBee, les appareils Bluetooth et les périphériques infrarouges. L'infrastructure se compose d'un transmetteur sans fil, des ceintures de parking, périphériques à infrarouge (IFD) & un ordinateur de contrôle. L'émetteur-récepteur sans fil peut être partie d'un réseau local sans fil (WLAN). Il est utilisé pour transmettre l'information concernant le stationnement (par exemple, la capacité de places de stationnement vides et les informations de réservation. Les ceintures de stationnement et de l'IFD, coopèrent ensemble pour le check-in d'un véhicule. L'IFD à la place de stationnement envoie un signal lumineux. Une lumière jaune signifie endroit vide. Une lumière bleue signifie endroit rempli. Une lumière rouge signifie que les véhicules qui sont garés illégalement seront chargés d'une amende. Pour éviter la fausse détection de véhicules, ils ont utilisé deux IFD et les ceintures de vérification si les véhicules se comportent correctement.

le centre informatique entraîne fréquemment des informations qui sont mises à jour au station de base. La station de base transmet ensuite ces informations au station publisher. les véhicules qui passent par la ceinture de station publisher reçoit une copie de ces informations de parking. Cette copie inclut les places du parking, la capacité des lieux vides, et l'agencement des espaces vides (layout of empty spaces).

Autres solutions pour les places de parking sont données par des entreprises comme [8] qui utilise des détecteurs à ultrasons à LED pour détecter la présence du véhicule en utilisant la technologie des ultrasons (à l'intérieur) et des capteurs sans fil à l'extérieurs qui détecte la présence du véhicule en utilisant un magnétomètre (monté en surface ou incorporé dans le sol).La société Lyberta [9] qui offre une solution dans ce domaine en utilisant des capteurs pour détecter toujours les places disponibles.

La société OptiPark [10] qui utilise un réseau des capteurs sans fil pour détecter si des places de stationnement individuelles sont vacantes ou non. Cette société a déployé différents systèmes dans des différentes ville comme Amsterdam, Baden, Bologna, Bruges et Madrid. La société StreetLine [11] très connue dans le domaine des solutions de parking dans le monde entier, présente une solution efficace pour résoudre ce problème au moyen des capteurs qui sont embarqués au niveau des rues et chaque capteur contient en réalité un tableau des différentes composantes de détection et de la logique pour gérer la collecte de données à la place de parking individuelle.

*Approches agent*

L'approche multi-agent est introduit dans [12] où un système de réseau de coordination entre les conducteurs et les parkings est pris en charge, permettant à un ensemble d'informations de voiture du système multi-agents d'être créé dans le véhicule. Ce système peut communiquer et coordonner avec les agents de parkings de tous les endroits. Donc il y a plusieurs agents utilisés dont l'un est le "Car Park Agent" qui se trouve dans chaque place de parking pour transmettre les informations reliées à la place aux autres agents inscrits dans le système golbale, L'agent de parking contrôle le comportement de la place de parking à laquelle il appartient et cet agent possède une meilleure intelligence pour apprendre à ajuster avec plus de souplesse les informations qu'il récupère. Une autre approche multi-agent est utilisée dans [13] mais pas au niveau des surveillance des places de parking, à ce niveau là il y a des capteurs qui s'occupent transportes les informations sur les places de parking au "infostation". Dans [14] un agent dans la zone de parking surveille en donnant des informations sur la disponibilité d'une place et sur sa légalité. [15] a parlé d'avoir deux agents dans la zone de parking, l'une qui va attribuer la place la plus proche et l'autre la place qui corresponds au profile de l'utilisateur.

Après avoir vu dans cette partie, comment la surveillance des places de parking a été traité par différentes approches, la partie suivante recenser le moyen de guider le conducteur vers les places libres surveillées.

## 2.2 Guidage vers la place de parking

Plusieurs méthodes sont présentées pour aider le conducteur à se diriger vers la place de parking sélectionné. Ces méthodes différent d'un système à un autre mais toujours en arrivant pour le même but qui consiste à trouver une place rapidement et efficacement

*Approches non-agent*

Dans [4], le système utilise l'algorithme A-Star (A *) pour attribuer automatiquement un espace pour les conducteurs pour leur donner le plus court chemin en fonction du point d'entrée du bâtiment. Le conducteur est guidé vers la place spécifié grâce aux panneaux électrique et une carte imprimée sur le ticket de stationnement. Le même principe est utilisé dans [8] où comme l'automobiliste est guidé vers les différents lots avec des panneaux électriques qui sont placés à différents emplacements pour afficher le nombre exact de places libres dans un lot. Dans [5] avec avec le RSU (Road Side Unit) un véhicule peut facilement trouver un espace de stationnement vacant dans un grand parking et un système de navigation propre à chaque conducteur est utilisé car d'après les auteurs le système GPS ne produit pas un système de navigation à temps réel donc ils ont faire appele aux RSU pour récupérer les informations au temps réel pour guider le conducteur. La société Lyberta [9] utilise l'affichage d'une carte géographique sur le smart-phone avec les stationnements libres et le statut de chacun : résidentiel ou de courte durée,libre ou occupé qui est d'une manière ou d'autre proche au solution donnée par la société OptiPark [10] qui fait afficher sur le smart-phone une carte géographique pour guider le conducteur à sa place de parking. Dans [6] un sous-système de guidage automatique est utilisé où les nœuds directeurs divisent leurs zones qu'ils gérent en plusieurs sections selon "the turn offs of the parking layout". Ce sous- système permet aux véhicules de trouve une place de parking dans le moindre temps et il se compose de deux modules. L'application de guidage qui traite les informations par le sous-système de gestion, au moment il ya un changement dans le statut, les transmet au SINK. Les données traitées sont ensuite transmises par le SINK à l'application de guidage, qui est ensuite représentées sur l'écran de guidage du parking. L'écran de guidage du parking : Ce module rassemble les informations de l'application de guidage et les affiche pour les utilisateurs. Il montre la disponibilité des terrains de stationnement dans les trois directions (gauche / droite / avant). Un autre sous-système est présenté dans cet article qui est le Sous- système d'affichage d'entrée. Comme son nom l'indique, ce sous-système est placé à l'entrée du parking. Il montre l'état des places de parking pour les arrivants avant d'entrer dans l'aire de stationnement. Ce sous-système est divisé

en deux modules, l'application d'affichage d'entrée : Chaque fois qu'il ya un changement dans le statut, le module d'affichage de stationnement d'entrée sur le sous-système de gestion traite l'information et la transmet au sous-système d'affichage de stationnement d'entrée. L'application d'affichage qui fonctionne sur ce sous-système reçoit et traite les données. Les données traitées sont ensuite transmises à l'affichage d'entrée de parking pour l'affichage du statut. l'autre module est l'affichage de l'entrée du parking : Ce module affiche les informations sur l'état de stationnement pour les utilisateurs qui sont reçues de l'application d'affichage d'entrée. Il montre l'état complet de parking entier (le nombre total des places occupées / vacantes). La société StreetLine [11] offre deux méthodes pour guider le conducteur, tout à d'abord quand les données sont envoyées par les capteurs à leurs serveurs,ces derniers relaient l'information à leurs applications web et mobiles pour guider ensuite le conducteur à sa place de parking et l'autre méthode est l'utilisation des panneaux électriques qui sont mis surtout aux intersections des rues. Dans [7] une simple carte de guidage est donnée au conducteur après sa réservation et elle est affiché soit sur son téléphone portable, soit sur le système de guidage de son véhicule soit sur son laptop.

*Approches agent*

Dans [13] Une fois qu'une place est choisie, l'agent Assistant Personnel examine les détails du place et affiche des directions précises. Une explication audio accompagnant la description texte était proposée à ce service, car elle offrirait la moindre distraction, permettant à l'utilisateur de se concentrer sur la conduite. La représentation graphique sur le téléphone portable (ou même sur l'oridnateurs protable) dépends de la capacité de ce téléphone, si la représentation graphique n'est pas possible il y aura des informations sous forme d'un texte qui va être afficher sur ce téléphone pour guider le conducteur. Dans [15], un agent (VCL) va simplement utiliser les données de GPS qui est embarqué dans la voiture pour guider le conducteur à sa place de parking choisie.

Dans [16],ils proposent une structure de système moderne de distribution de stationnement intelligente de guidage ( distributed intelligent parking guidnance system-DIPGS) en utilisant l'approche de navigation multi-phase en explorant la technologie multi-agents. Le système fournit une plateforme de negotitation pour l'itinéraire et facilite de la navigation des places de parking du position actuelle au palce de parking prévue. Chaque centre de service des informations de stationnement (parking information service center-PISC) maintient la carte du trafic et le trafic de données de district local. Une carte du trafic contient des routes locales, et ses nœuds externes qui peuvent être divisés en deux types : le noeud d'entré et de noeud de sortie, la carte virtuelle de trafic de l'ensemble du système est composée par tous les nœuds externes de la carte de la carte de trafic locale. La carte du trafic locale et toute carte du trafic virtuelle actualisent les données de trafic et de statut du trafic périodiquement par le centre de gestion du trafic (traffic management center-TMC).Chaque PISC est le seul responsable de la navigation de la voiture conduite dans son propre district, et donc, le processus de navigation de la voiture commence à partir du noeud d'entrée dans le district des locaux pour le noeud de sortie adjacentes au district suivant. Cependant, quand il ya beaucoup des sortie acquiesces adjacentes au district suivante, la voiture aura besoin de sélectionner un noeud de sortie appropriée en tant que destination dans le district local. Le noeud de sortie voulu est déterminé provisoirement. Lorsque la voiture vient d'entrer dans le quartier, d'une part, elle a besoin d'estimer l'état du trafic sur les routes à chaque noeud de sortie dans les districts locales, et les routes connectées au nœud de sortie dans le district voisin. Après une comparaison exhaustive, la voiture sera de déterminer un point de sortie optimale. un algorithme de négociation d'itinéraires est également proposé pour la l'agent voiture et Agent-SIG pour négocier les routes acceptables pour les deux parties sur la base de calcul de l'utilité itinéraire.

Dans [12], le système de la négociation de stationnement et de guidage (the parking negotiation and guidance system-PNGS) souligne le lien entre les conducteurs et les parkings afin que le système d'information embarqués puissent négocier avec les parkings et dériver des chemins optimaux à partir de l'emplacement actuel de conducteur vers la place de parking ainsi que du place parking à la destination finale. Donc dans cet article le guidage contient deux rôles qui sont impliqués : l'agent conducteur et l'agent de guidage. L'agent conducteur interagit avec l'agent de guidage pour trouver l'itinéraire optimal sur la carte. L'agent conducteur demande tout d'abord une proposition de l'agent de guidage, puis les demandes de renseignements sur l'état après avoir reçu une réponse. L'agent de guidage réponds à l'agent conducteur en fournissant les informations de route, afin que le conducteur puisse efficacement arrive à la place de parking correspondante. Donc L'agent de guidage contrôle de l'information géographique constitué d'un ensemble des zones et cet même agent modélise alors le comportement dynamique d'une zone spécifique à laquelle est associée en fournissant secteur des services spécifiques. L'agent de guidage utilise le système d'informations géographiques qui trouve l'emplacement et les données de destination,

puis obtient la distance la plus courte de la position de départ de véhicule au place de parking puis au destination finale en utilisant l'algorithme de Dijkstra.

On a vue dans les deux parties précédentes comment les places de parking sont surveillées et comment le conducteur est guidé à sa place, dans la partie qui va suivre ( 2.3) on présente des différentes manières possibles pour réserver une place de parking.

## 2.3 Réservation d'une place de parking

Le service de parking intelligente est payant mais chaque solution essaye en même temps d'optimiser l'utilisation de ce service afin de réduire le coût pour l'utilisateur. Plusieurs systèmes de paiement ont été mis en œuvre en utilisant différentes technologies et techniques.

*Approches non-agent*

Dans [7] les conducteurs peuvent consulter et réserver des places de parking à la volée. Un service de stationnement sans-arrêt peut être fournis aux conducteurs. Donc il y un Module d'application sur le téléphone,Ipad ou laptop des conducteurs qui gère le système de parking tout entier. La fonction principale de l'application comprends la gestion des comptes (trésorerie, crédit / débit) et la gestion des opérations, la tolérance aux pannes et de gestion de la maintenance. Chaque ordre de réservation est constitué d'un ELP-ID (Electronic license plate) de véhicule, du numéro de la place réservée, un horodatage, et du temps d'expiration de transaction. Si le véhicule réserve une place, mais ne se présente pas avant l'heure d'expiration de transaction, l'emplacement réservé est libéré et le véhicule doit réserver une autre place de stationnement.

Si un véhicule ne réserve pas une place et arrive directement au parking, le stand d'entrée de stationnement va choisir aléatoirement un endroit vide pour le véhicule. Si plusieurs réservations entrent en conflit, de nouvelles places de stationnement sont sélectionnées au hasard parmi les espaces vides pour résoudre le conflit. Si la place conflictuelle est la dernière place, la place sera attribuée au véhicule avec l'horodatage ancien. Les autres véhicules seront informés que la place libre n'est plus disponible.

Si un conducteur décide d'annuler une réservation de parking, le processus d'annulation est similaire au processus de réservation. Un ordre d'annulation est tombée au ceinture et ensuite transmis à la station de base où l'ordre d'annulation est traité. Si le chauffeur ne libère pas la place avant le délai d'expiration, il est facturé en partie.

Lorsque le véhicule avec une réservation valide arrive au poste d'entrée de stationnement, l'émetteur-récepteur du véhicule peut communiquer avec l'émetteur-récepteur du stand afin de lui transmettre le numéro de réservation. L'ordre de réservation est transmis au centre de contrôle où la transaction est validée, une carte de guidage pour la place de stationnement est calculée. Le centre de contrôle transmet ensuite la confirmation et l'orientation dans la carte pour les kiosques où ces informations sont imprimées et ont fournies au conducteur.

Dans [6] le système permet la réservation d'une place de parking à distance. Un message de réservation est transmis au module qui s'occupe de réservation de stationnement. Il récupère les données de la base de données de capteurs et, en se basant sur la disponibilité des places de stationnement, il fera parvenir un accusé de réception au client. L'application de réservation de parking fonctionne sur des appareils permettant aux utilisateurs de donner leurs coordonnées de stationnement (comme le temps de stationnement) afin de réserver un parking. Cette information est ensuite traitée par l'application cliente et envoyée pour réserver la place de parking.

Dans le système prototype présenté par cet article, l'utilisateur saisit ses coordonnées de stationnement sur l'interface de réservation cliente fonctionnant sur son téléphone portable. Ce message est transmis au serveur de gestion par SMS. Le serveur de gestion trouve le numéro de la place de parking libre et envoie un message de confirmation à l'utilisateur avec le numéro de stationnement et l'heure de début de stationnement. Le serveur de gestion envoie également des informations d'état au sous-système WSN qui indique par la LED bleue le stationnement réservé et au GUI de stationnement pour mettre à jour les places de parking.

La technologie RFID est utilisée par [17] pour identifier les véhicules qui entrent dans une zone de parking. La réservation d'une place est automatique lors de l'entrée d'un véhicule dans cette zone de parking. A la fin de chaque mois, le total des frais de chaque conducteur membre de cette zone de parking est calculé. Ensuite, ces frais sont débités de son compte bancaire et transférés aux comptes des gestionnaires de parking. Les sociétés OptiPark [10] et Lyberta [9] offrent un service de pré-réservation

de place sur smartphones. Dans [5] la place à réserver par le conducteur dépend de la proximité et de la nature de la place avec des préférences de conducteur. Dans [4] la réservation se fait à distance par le conducteur et la confirmation de sa réservation est transmise ensuite à un serveur central qui, à son tour, transfère ces informations au kiosque correspondant, dès que le conducteur arrive à sa place le temps de facturation commencer

*Approches agent*

Dans [14] le choix d'une place de parking se fait par des préfrences sur le prix, la proximité et les préférences. Cela est le même cas dans [16] tandis que dans [15] on ajoute, sur la proximité, le prix et le profile du conducteur le type de place où le conducteur a droit de se garer c'est à dire il y a des places qui sont réservées déjà pour certaines personnes comme les directeurs, visiteurs, étudiants, handicapés..etc

Dans [13] la réservation est faite sur les téléphones portables, sur les laptops et les IPads où l'agent, assistant personnel, transmet également une demande de réservation de place de parking au Centre InfoStation. Une fois la place a été réservée (et que l'occupation de place est confirmée par le réseau de capteurs), aucun autre conducteur ne peut la prendre. Dans [12] le réseau de coordination entre le conducteur et les parkings signifie qu'un conducteur peut saisir l'emplacement actuel et sa destination via l'interface utilisateur lorsque le conducteur souhaite se garer. L'agent peut compléter une négociation avec d'autres agents du système . Les agents travaillent automatiquement, sans se reposer sur la coordination, permettant aux conducteurs de chercher une place de parking libre rapidement, et d'avoir le prix de stationnement négocié avec le parc automobile, et l'appariement des spécifications de la voiture du conducteur.

Deux systèmes d'information de stationnement sont distingués en fonction de leur comportement dans la prise de réservations. Les systèmes d'engagement qui fournissent une garantie de réservation sur demande. Et, des systèmes non-engageant de réservation qui ne donnent qu'une indication approximative de la possibilité de trouver une place de parking, et garantir la disponibilité de cette place. Cette limitation en matière de sécurité est reflétée par le prix : l'engagement des systèmes d'information de stationnement ont généralement un prix plus élevé.

## 2.4 Synthèse

L'utilisation de systèmes multi-agents (SMA) est une solution naturelle et efficace pour faire face à des situations complexes dans des environnements distribués. En particulier, la situation étudiée dans mon stage se concentre sur l'intégration de ce système dans l'infrastructure de projet SensCity et sur le développement d'une application de gestion du stationnement à l'aide d'agents pour gérer la distribution de connaissances et de parvenir à un consensus sur l'affectation d'une zone de stationnement pour un véhicule. Donc cette technologie de SMA résout le problème de l'attribution des places de parking d'une façon automatique en exploitant les connaissances acquises par les agents.

Cette approche va répondre parfaitement aux problématiques d'un système multi-agents pour bien montrer l'ensemble des agents qui vont agir entre eux dans un environnement donné comment ils interagissent avec l'environnement en même temps. Dans cette approche la capacité des agents à réaliser des actions pour atteindre des objectifs est une question clé, ainsi que la focalisation sur la méthode d'interaction entre les agents en faisant la différence entre la notion de collaboration et coopération. L'adaptation est l'une des caractéristiques qui peut se situer au niveau individuel ou collectif afin de permettre au système d'évoluer.

# Chapitre 3

# Proposition de gestion intelligente de parking

Après avoir vu les différentes approches et techniques abordées par des différents chercheurs et même sociétés qui s'occupent de trouver une solution efficace pour le problème de parking dans les villes, on présente notre approche qui nous amene à avoir une parking intelligente en intégrant le système multi-agents dans le projet SensCity car un système multi-agents (SMA) est un système composé de plusieurs agents intélligents qui sont en interaction. et ce genre de système est utilisé pour résoudre les problèmes qui sont difficiles ou impossibles pour un agent individuel ou un système monolithique à résoudre. L'intelligence peut inclure certains de recherche méthodologique, fonctionnelle, procédurale ou algorithmique, l'approche de trouver et de traiter. Donc les technologies multi-agents apportent adaptation, flexibilité et proactivité dans un cadre décentralisé.

La modélisation multi-agents de notre système est de distrubuer les connaissances qui sont attribuées aux agents afin d'avoir une méthode d'interaction entre eux et de définir une organisation. Dans notre système qui est considéré comme un système distribué, l'un de challenge est de choisir/déterminer l'agent qui va se communiquer avec le bon agent sachant que la présence des agents mobiles dans un réseau forme une particularité que nous devions le prendre en considération. Notre système saisit les caractéristiques d'un système multi-agents coopératif, c'est à dire le système reflète la résolution rationnelle du problème en termes de principes pour optimiser l'utilité globale du système et chaque agent observe qu'une partie de l'état global du système. les agents doivent être capables de prendre des décisions individuelles sur la base des informations disponibles récupérer soit par eux mêmes soit par la communication et l'échange avec d'autres agents en formant une coordination supplémentaire locale et non locale pour gérer les contraintes d'interdépendance de leurs activités.

Le système entier peut être divisé en sous-systèmes, le sous système *Driver Guidance* qui contient les agents *DriverGuideAg* et *ParkSelectorAg*. Ce dernier sous-système se communique avec un autre sous-système *Parking Management* qui contient les agents *AccountAg* et *ParkingAreaAg* qui à son tour se communique avec le sous-système *USP* de l'architecture de SensCity. La couche *USP* se communique au Gatway afin d'être en communication avec les capteurs. La communication entre *Driver Guidance* et *Parking Management* sera décrite dans la section 3.1 puis entre *Parking Management* et la couche *USP* dans la section 4.

Cette proposition se base sur des différentes entités qui forment notre système. Des capteurs qui sont répartis dans toute la ville pour récupérer toutes informations sur les places de parking, un groupe des places qui forme une zone de parking et un système embarqué dans les smartphones qui assure l'interaction entre l'utilisateur et les autres entités du système.

## 3.1 Modèle de gouvernance par une organisation (MOISE)

Le langage de modélisation d'organisation Moise est utilisé pour définir une organisation au sein d'un système multi-agents suivant trois dimensions indépendantes :(i) la spécification structurelle (SS) et (ii) la spécification fonctionnelle (FS) reliées ensemble par la spécification normative (NS). Ainsi MOISE est particulièrement adapté à la définition de modèles de gouvernance flexibles.

*La spécification structurelle* est un ensemble de groupes composés de rôles. En fonction de cette spécification, les agents s'engage dans un rôle en fonction de leurs compétences et créent des groupes suivant la spécification organisationnelle. Quand un agent s'engage dans un rôle, il s'engage à remplir lui-même les normes concernant les plans sociaux pour atteindre les buts du système.

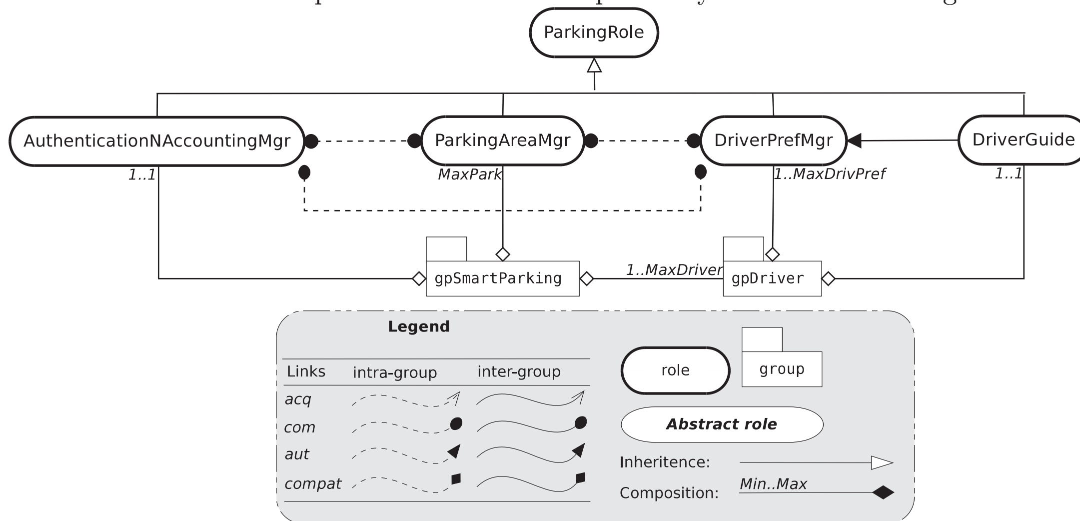
*La spécification fonctionnelle* définit un ensemble de buts pour l'organisation et un ensemble de missions à remplir pour atteindre ces buts organisés en schémas sociaux. Un schéma social est un plan pour atteindre un but, décomposé en un arbre de sous-buts, décrits en utilisant une opérateur de plan et une liste de sous-buts. Deux types de buts peuvent être définis : les buts ponctuels ou achievement (A), devant être atteints une seule fois, et les buts de maintenance (M) devant être constamment satisfaits. Chaque but a un temps maximum pour être accompli ou time-to-fulfill (TTF). Pour chaque schéma, un ensemble de missions regroupent les buts du schéma en des ensembles cohérents pouvant être alloués aux agents. Comme les rôles, les missions ont une cardinalité qui définit le nombre minimum/maximum d'agents pouvant y souscrire.

Enfin, *la spécification normative* assigne les missions aux rôles. Les normes définissent comment les missions peuvent ou doivent être achevées et par quels rôles. Une norme n spécifie une relation déontique d entre un rôle r et une mission m avant un temps T T F quand lorsque la condition c devient vraie.

### 3.1.1 La spécification structurelle

Les rôles sont utilisés afin de rendre les agents capables d'effectuer leurs fonctions dans le système. Ces rôles permettent de contraindre les comportements individuels des agents. Chaque rôle spécifie le comportement autorisé d'un agent dans l'organisation au travers de l'ensemble des activités qu'il peut exercer. L'organisation de système de parking est structurée en deux groupes (voir Figure **??**) : (i) gpSmartParking et (ii) gpDriver.

Figure 3.1 – Spécification structurelle pour le système Smart Parking.




Le groupe gpSmartParking est composé de rôles pour regrouper tous les places de zones de parking Chaque rôle a une définition et une cardinalité dans l'organisation. Voici la liste des rôles d'une infrastructure de système de parking qui sont inclus dans gpSmartParking :

- *parkingAreaMgr* : qui prend la responsabilité de gérer une place de parking dans une zone. Cette place a des propriétés (occupée, libre..etc) et elle est géolocalisée dans la zone auxquelle elle appartient. Il a un lien de communication avec le rôle *driverPrefMgr* et avec le rôle *authenticationNAccountingMgr*.
- *authenticationNAccountingMgr* : Il est responsable de l'authentification des conducteurs et des zones de parking. Il est aussi responsable de la facturation de l'utilisateur du système (le service de notre système de parking). Il a un lien de communication avec le rôle *driver PrefMgr* et le rôle *parkingAreaMgr*.

Le groupe gpDriver est composé des rôles qui aident à trouver une place de parking et à guider le conducteur vers la place sélectionnée . Voici la liste des rôles qui sont inclus dans ce groupe :

- *driverGuide* : Il est responsable de guider l'utilisateur vers la place de parking qu'il a sélectionné. Pour arriver à cette place, *driverGuide* fait appel au rôle *driverPrefMgr* donc un lien d'autorité sur ce dernier est établi.

- *driverPrefMgr* : Il est responsable de trouver une place de parking qui corresponds le mieux au profile du conducteur. Pour trouver cette place, ce rôle se communique avec *driverGuide* donc il a un lien de communication avec ce rôle.

### 3.1.2 Spécification fonctionnelle

Cette partie définit la spécification fonctionnelle qui gouverne les buts et missions du système de parking, à travers la figure 3.2. Ce schéma social a pour but racine (*driverParked*) qui est satisfait par la séquence (i) Authentifier le conducteur (*driverAuthenticated*), (ii) la facturation de l'utilisation du système (*driverBilled*) puis (iii) guider le conducteur vers la place de parking sélectionnée (*driverGuided*).

Le premier sous-but consiste à être authentifié dès que le conducteur commence à utiliser le système. Il est accompli par la mission *mAuth*.

Le sous-but *driverBilled*, qui est un sous-but de maintenance, consiste à commencer la facturation dès que l'utilisateur commence à utiliser le système et quand le conducteur stationne dans sa place selectionnée. Il est accompli par la mission *mBill*.

Le sous-but *driverGuided* consiste à guider le conducteur vers la place qui corresponds le mieux à ses attentes. *driverGuided* est un but ponctuel satisfait par la séquence : (i) Trouver une place (*placeFound*) puis (ii) Aller à la place trouvée (*goneToPlace*).

*placeFound* est aussi un but ponctuel satisfait par la séquence : (i) Trouver les places qui correspondent aux critères de l'utilisateur (*bidForPlacesResponsed*) puis (ii) Chercher une meilleure place (*bestPlaceSelected*) qui sert à améliorer la sélection de la place.

Les buts de ce schéma social sont regroupés dans les missions suivantes :

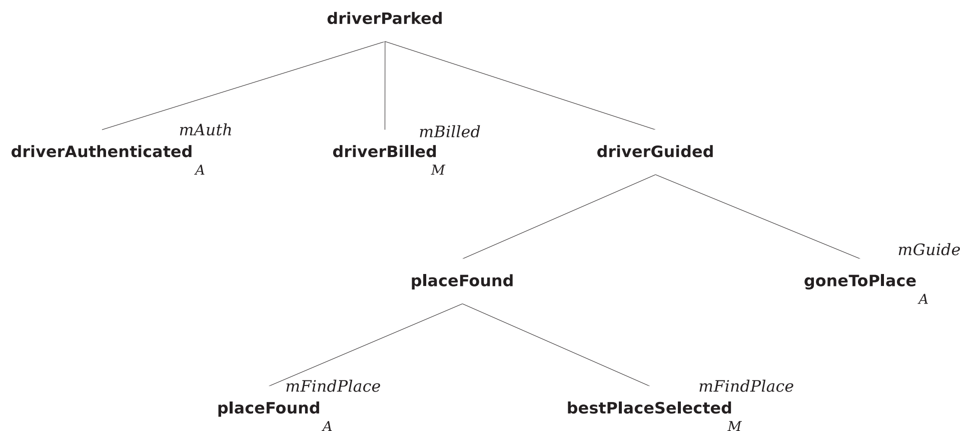
- mAuth [2..$N_PA$] pour authentifier l'utilisateur au niveau du système. Il y a une demande et une réponse ce qui nécessite au moins deux agents qui s'occupent de ces fonctions puis c'est le nombre des zones de parking qui détermine éventuellement le nombre des agents associés à cette mission ;

- mBill [1..1] pour commencer à facturer l'utilisateur lors de son utilisation du service et au moment il occupe la place qu'il a sélectionné. C'est un agent par chaque conducteur pour produire la facturation correspondante au conducteur ;

- mGuide [1..1] pour guider l'utilisateur vers sa place de parking sélectionnée. Un agent est associé à cette mission vu que le système guide un seul conducteur vers une place de parking ;

- mFindPlace [1..1] pour trouver la meilleur place possible qui correspond aux contraintes de l'utilisateur. Après l'amélioration des places cherchées, un agent s'occupe de sélectionner la meilleure place

Figure 3.2 – Spécifications fonctionnelles pour le système Smart Parking.



### 3.1.3 Spécification normative

Cette partie définit les normes pour contraindre les actions que les agents peuvent effectuer dans le système de smart parking. La table 3.1 résume certaines de ces normes relatives aux schémas présentés précédemment : les normes n01 à n05 définissant comment les données doivent être échangée.

TABLE 3.1 – Spécifications normative pour le système smart parking

| ID | Condition | Rôle | Relation | Mission | TTF |
|---|---|---|---|---|---|
| $n_{00}$ | fulfilled(obligation(_,$n_{04}$,_,_)) | *driverPrefMgr* | obligation | *mAuth* | ttfmAuth |
| $n_{01}$ | _ | *authenticationNAccountingMgr* | obligation | *mAuth* | ttfmAuth |
| $n_{02}$ | fulfilled(obligation(_,$n_{00}$,_,_)) | *driverPrefMgr* | permission | *mFindPlace* | ttfmFindPlace |
| $n_{03}$ | fulfilled(obligation(_,$n_{00}$,_,_)) | *parkingAreaMgr* | obligation | *mFindPlace* | ttfmFindPlace |
| $n_{04}$ | fulfilled(obligation(_,$n_{00}$,_,_)) | *driverGuide* | obligation | *mGuide* | ttfmGuide |
| $n_{05}$ | fulfilled(obligation(_,$n_{00}$,_,_)) | *authenticationNAccountingMgr* | obligation | *mBill* | ttfmBill |

Les agents jouant le rôle *driverGuide* sont supposés guider l'utilisateur, en effectuant la mission *mGuide*. Comme précisé ci-dessus, cette mission consiste à demander à l'utilisateur ($n_{00}$ ) de lui communiquer sa destination afin de démarrer le système . Dans tous les cas, les agents *DriverGuideAg* doivent s'engager dans la mission *mGuide*. Cependant, la valeur ttfmGuide peut varier avec le trajet à faire par le conducteur.

Les autres mission $n_{03}$, $n_{04}$ et $n_{05}$ démarrent à la demande de $n_{00}$ ce qui est logique car après que l'utilisateur communique sa destination à l'agent qui joue le rôle *driverGuide* l'agent qui joue le rôle *driverPrefMgr* s'occupe de gérer le système entier.

La partie développement se trouve en annexe C.0.1. Cette partie est toujours en cours et il y aura des changements à faire.

## 3.2 Modélisation Multi-Agents du problème

Notre système de smart parking est basé sur un système Multi-Agent. Chaque agent correspond à une fonctionnalité du système et adopte un rôle correspondant dans l'organisation définie précédemment. Les agents de notre système sont : *DriverGuideAg*, *DriverGuideAg*, *ParkSelectorAg*, *ParkingAreaAg* et *AccountingAg*.

### 3.2.1 Agent *DriverGuideAg*

Il aide le conducteur à arriver à la place la plus proche de sa destination en lui indiquant l'itinéraire. Son rôle *driverGuide* est de spécifier son comportement dans l'organisation pour arriver à son but *gone-ToPlace* qui est de trouver une place de Parking avec des contraintes donnés. Cet agent s'inscrit à la mission *mGuide*.

Pour arriver à son but, l'agent *DriverGuideAg* communique avec l'agent *ParkSelectorAg* qui joue le rôle *driverPrefMgr* afin de lui montrer sur l'écran de smart-phone (ou sur l'écran du système embarqué dans le véhicule) une carte géographique avec les directions vers la place trouvée. L'agent *DriverGuideAg* utilise l'artefact *SetDestArt* afin que ce dernier met à jour toujours l'itinéraire sur la carte avec chaque amélioration qui sera faite au niveau de la meilleure place trouvée.

### 3.2.2 Agent *ParkSelectorAg*

C'est l'agent qui sélectionne une des places de parking proposées par les agents jouant le rôle *parkingAreaMgr* en fonction du profile de l'utilisateur. Il adopte le rôle *driverPrefMgr* dans un groupe gpDriver de l'organisation. En conséquence, il s'inscrit dans la réalisation des missions (i) *mAuth* et (ii) *mFindPlace*.

La mission *mAuth* consiste à identifier l'utilisateur (but *driverAuthenticated*) pour pouvoir utiliser le système et être facturé en conséquence. Pour cela, il envoie les identifiants de l'utilisateur à l'agent jouant le rôle *authenticationNAccountingMgr* qui lui fourni en retour un identifiant de session à utiliser pour les requêtes au près des *parkingAreaMgr*.

Son rôle dans la mission *mFindPlace* est de sélectionner la place de parking qui correspond le mieux au profile de l'utilisateur. Il doit donc satisfaire les buts *bidForPlacesResponsed* et *bestPlaceSelected*.

Il contribue à la satisfaction du but *bidForPlacesResponsed* en contactant des agents jouant le rôle *parkingAreaMgr* pour qu'ils lui fournissent les caractéristiques de leurs places de parking. Il sélectionne ces destinataires qui se trouvent dans un rayon $\Delta$ de la destination et qui regroupe tous les agents *ParkingAreaAg* qui se trouvent dans ce rayon. Les différentes zones groupées dans la rayon forment une plus grande zone $Z_{dest}$ (equation 3.1).Si la place n'est pas trouvée, la grande zone $Z_{dest}$ augmente d'une valeur $\Delta$ multipliée par $\delta$ (equation 3.2).

$$Z_{dest} = \{\forall z \in Z_{all} : |destination + z.location| \leq \Delta\} \tag{3.1}$$

$$\Delta \leftarrow \Delta \times \delta \qquad where\,\delta > 1 \tag{3.2}$$

Les critères forment une fonction d'évaluation pour une propriété. Les différentes propriétés sont définies dans la table 3.2 qui regroupe des exemples de critères de préférence sur les propriétés d'une zone de place de parking. Chaque utilisateur peut avoir plusieurs profils prédéfinis (travail, vacance, weekend..etc). Chaque profile forme un ensemble de critères sur les propriétés des zones de parking et un critère est une fonction d'évaluation d'une propriété qui renvoie *1* si le critère est satisfait et *0* sinon.

La distance est une propriété que l'utilisateur peut en mettre un critère pour préciser sa préférence pour la distance qui sépare sa destination de la place visée. De même pour le prix horaire d'une place de parking, le type d'une place, la durée maximale autorisée pour une place, la probabilité de disponibilité et le temps maximal de réservation avant l'arrivée.

L'agent *ParkSelectorAg* envoie une requête

$$DestinationAndArriving = \{Destination, ArrivalTime\}$$

aux agents *ParkingAreaAg* qui se trouvent dans une zone $Z_{dest}$ en communiquant l'heure de l'arrivée au place avec la destination. Les agent *ParkingAreaAg* trouvent tous les places libres (avec tous leurs propriétés) dans la zone $Z_{dest}$ et les renvoie par la requête

$$Places = \{ParkingPlacesAvailable\}$$

à l'agent *ParkSelectorAg* qui va s'occuper de trouver le match avec ses préférences sur les différentes propriétés des places trouvées. Donc l'agent *ParkSelectorAg* effectue une fonction d'évaluation pour choisir une place de parking qui répond le mieux à ces critères et il répond par la requête

$$GetPath = \{CoordinatesOfSelectedPlace\}$$

à l'agent *DriverGuideAg* en lui précisant la localisation géographique de la meilleure place sélectionnée.

TABLE 3.2 – Exemples de critères de préférence sur les propriétés d'une zone de places de parking

| Propriétés | Type | Format | Description |
|---|---|---|---|
| $Distance$ | Int | $< D_{max}$ | – Distance en mètres qui sépare une place de parking de la destination finale |
| $Price$ | Float | $< P_{max}$ | – Prix horaire maximal d'une place de parking |
| $PlaceType$ | $\{O, H, E, R, M\}$ | $= P_{type}$ | – Type d'une place de parking qui peut être ordinaire (O), pour handicapés (H), avec chargeur pour véhicules électriques (E), réservable (R) ou pour motos (M) |
| $ParkingDuration$ | Time | $< T_{max}$ | – Durée maximale de stationnement autorisée exprimée en minutes (m), heures (h) ou jours (d) |
| $Availability$ | [0..1] | $> P_{avail}$ | – Probabilité qu'une place de la zone soit encore disponible à l'arrivée du conducteur. Dans le cas d'une place réservable, c'est la probabilité de disponibilité à partir de l'ouverture de réservation $TimeToReserve$ |
| $TimeToReserve$ | Float | $< TR_{max}$ | – Temps maximal de réservation d'une place avant l'arrivée du conducteur. Elle est annulée automatiquement à la fin de ce temps. |

Les critères sont pondérés par un niveau d'importance, dont les valeurs sont ordonnées de manière décroissante comme suit : *Mandatory (M)*, *Important (I)* et *Optional (O)*. Par exemple, si un critère sur la propriété *distance* a une importance {M}, cette propriété sera à satisfaire avant les autres d'importance moindre. Plusieurs critères peuvent être de même importance. Notre système sélectionnera la place qui

satisfera le plus de critère de plus grande importance, et utilise les niveaux suivant d'importance pour départager d'éventuels ex-æquo.

L'équation 3.3 $eval(P)$ évalue une place $P$ par rapport à la liste des critères $criteria$ qui correspond aux préférences de l'utilisateur. Elle retourne une valeure numérique égale à la somme des critères satisfaits pondérée par leur importance. Où $cr_i(P)$ vaut 1 si le critère $cr_i$ est satsifait par la place $P$ et $N_{cr}^{cr_i.imp}$ est le nombre de critères $N_{cr}$ à la puissance $cr.imp$, c'est-à-dire l'importance du critère.

$$eval(P) = \sum_{cr_i \in criteria} cr_i(P) * N_{cr}^{cr_i.imp} \quad (3.3)$$

La procédure 1 détermine une place qui correspond le mieux aux préférences de l'utilisateur. Une liste des places est envoyée à cette procédure qui va considéré en premier temps que la première place dans la liste est la meilleure place trouvée ensuite il s'agit de passer dans toute la liste des places en évaluant à chaque fois une place de la liste avec la meilleure qui est déjà trouvée et cela se fait par l'équation 3.3 $eval(P)$. Si une meilleure place $P$ est trouvée, cette place remplacera la place $BestPlace$.

**Procedure 1** La procédure *findPlace* qui détermine la place qui correspond le mieux aux préférences de l'utilisateur.

**Input:** $Places = \{P_0..P_{max}\}$ : list of possible places
**Output:** $BestPlace$ : the chosen place which matches the best the user preferences

```
1: Imp ← {M, I, O}                # Importance sequence : Mandatory, Important, Optional
2: BestPlace ← P0
3: for (P in Places\P0) do
4:    if eval(P) > eval(BestPlace) then
5:       BestPlace ← P
6:    end if
7: end for
8: return BestPlace
```

Le calcul de satisfaction d'un critère pouvant être couteux, on propose une petite optimisation. On peut compare par ordre d'importance en faisant appel à *compare* (voir procédure 2) affecté à BestPlace. On introduit à cet effet l'équation $eval_{Imp}$ 3.4 qui évalue la place $P$ pour une importance donnée.

$$eval_{Imp}(P) = \sum_{\substack{cr_i \\ cr_i.imp=Imp}} cr_i(P) \quad (3.4)$$

Il suffit de remplacer les lignes (4), (5) et (6) de la procédure 1 par la ligne suivante :

$BestPlace \leftarrow compare(BestPlace, P, Imp)$, Appel au procédure 2 compare(P1, P2,Imp)

La procédure compare(P1, P2,Imp) 2 nous permet de comparer deux places par l'ordre d'importance en lui passant une liste des importances ordonnées et qui va retourner la meilleure place possible pour y aller avec une satisfaction aux importances données.

### 3.2.3 Agents *ParkingAreaAg*

Il aide le conducteur à arriver à la place la plus proche de sa destination en lui offrant les places qui correspondent le mieux à ses contraintes. Il vise aussi à avoir un maximum nombre de réservation dans la zone qu'il gère. Son rôle *parkingAreaMgr* prend la responsabilité de gérer les places de parking dans une zone avec tous les propriétés qu'il possèdent et de chercher la meilleur place pour l'agent *ParkSelectorAg* et cela pour accomplir ses buts qui sont : *bidForPlacesResponsed* qui présente un nombre des places adéquats à l'agent *ParkSelectorAg*, *bestPlaceSelected* pour continuer à améliorer la place trouvée et *driverBilled* pour commencer à la facturation au moment de l'arrivée à la place réservée. Cet agent s'inscrit aux deux missions *mBill* et *mFindPlace*.

Pour arriver à ses buts, l'agent *ParkingAreaAg* communique avec l'agent *ParkSelectorAg* qui joue le rôle *driverPrefMgr* afin de lui proposer la meilleure place possible et il communique avec l'agent*AccountingAg* qui joue le rôle *authenticationNAccountingMgr* pour commencer la facturation.

**Procedure 2** La procédure *compare* qui compare 2 places en fonction des critères utilisateurs.

**Input:** $p_1$ : A place in the list of places
**Input:** $p_2$ : The best place found til the moment
**Input:** $Imp$ : order of importance to compare
**Output:** $ChosenPlace$ : the place which matches best the user's preferences, $p_1$ if equals

1: **if** $Imp_0 = \emptyset$ **then**
2: return $p_1$
3: **end if**
4: **if** $(\text{eval}_{Imp_0}(p_1) > \text{eval}_{Imp_0}(p_2))$ **then**
5: return $p_1$
6: **else if** $(\text{eval}_{Imp_0}(p_1) = \text{eval}_{Imp_0}(p_2))$ **then**
7: return $compare(p_1, p_2, Imp \backslash Imp_0)$
8: **else**
9: return $p_2$
10: **end if**

L'agent *ParkingAreaAg* utilise l'artefact *ReservationArt* afin que ce dernier attribue la place de parking réservée et confirmée par l'agent ParkSelectorAg.

### 3.2.4 Agents *AccountingAg*

Il aide d'une part à authentifier l'agent *ParkSelectorAg* pour lui ouvrir une session sur le serveur et d'autre part pour lui facturer tout au longue de l'utilisation du service. Ainsi, Il aborde le rôle *authenticationNAccountingMgr* et cela pour accomplir ses buts : *driverBilled* pour la facturation et *driverAuthenticated* pour l'authentification. Cet agent s'inscrit aux deux missions *mBill* et *mAuth*.

Pour arriver à ses buts, l'agent *AccountingAg* communique avec l'agent *ParkSelectorAg* qui joue le rôle *driverPrefMgr* afin de lui facturer et il communique avec l'agent*ParkingAreaAg* qui joue le rôle *parkingAreaMgr* pour commencer la facturation de l'agent *ParkSelectorAg* sur la place qu'il a pris.

L'agent *AccountingAg* utilise l'artefact *BillingArt* afin que ce dernier authentifie et facture l'agent *ParkSelectorAg*.

## 3.3 Vu globale du système

Les entités mentionnées dans la section 3 sont représentés par des agents qui gèrent la relation entre eux. À Chaque capteur il est attribué un agent *ParkingSensorAg* qui sera décrit en détail dans la section 4. Un certain nombre des agents *ParkingSensorAg* qui se trouvent dans une zone géographique forme une zone de parking gérée elle même par l'agent *ParkingAreaAg* qui communique à son tour avec d'autres agents du système.

l'agent *ParkSelectorAg* (3.2.2) se trouve du côté conducteur et qui s'occupe de ses préférences (critères) pour choisir la meilleure place de parking. L'agent *AccountingAg* (3.2.4) assure l'authentification au niveau conducteur ainsi que la facturation de l'utilisation du service. Un autre agent *DriverGuideAg* (3.2.1), comme son nom l'indique, il est utilisé pour guider le conducteur vers sa place de parking choisie.

L'agent *ParkingSensorAg* (4.4.1) nous aide à récupérer les informations pour une place de parking avec tous les caractéristiques de cette place. Pour la simplicité, on considère chaque place de parking est équipé d'un capteur qui est géré par un agent avec des propriétés qui le sont attribuées.

Chaque agent *ParkingSensorAg* communique à l'agent *ParkingAreaAg* sa localisation géographique avec des coordonnés (x,y) et sa disponibilité (libre or occupé) puis l'agent *ParkingAreaAg* ajoute des connaissances supplémentaires pour chaque place de parking dans sa base de données comme le type de place ( place pour les handicapés, place pour les véhicules électriques, pour les motos..etc), le type de réservation que le conducteur peut faire (place pour une courte durée ou pour une longue durée)sa durée de réservation (voir 3.2. Donc une place groupe un ensemble des caractéristiques et des informations qui facilitent d'une manière ou d'autre à l'agent *ParkingAreaAg* d'avoir un management complet pour sa zone de parking et ensuite de passer ces informations à l'agent *ParkingAreaAg* qui devrait en servir selon la requête envoyée par l'agent *ParkSelectorAg* pour trouver une place de parking.

Une fois l'agent *ParkingAreaAg* récupère tout de ses agents *ParkingSensorAg*, il garde ces informations dans sa base de données pour chaque place de parking.À ce niveau on voit bien l'échange des connaissances entre les agents. L'une des connaissance ajoutée à cet agent est le prix attribué à chaque place de parking.

L'avantage d'avoir un agent qui gère une zone de parking est pour simplifier la communication avec l'agent *ParkSelectorAg* qui sera mener à communiquer seulement avec l'agent *ParkingAreaAg* afin d'avoir une connaissance globale sur tous les places dans cette zone au lieu de communiquer avec des milliers des agents *ParkingSensorAg* ce qui va être une perte de temps, sur-chargement de réseau et une complexité ajoutée au système.

Comme détaillé ci-dessus, chaque agent a un rôle à remplir. Au niveau conducteur, l'agent *DriverGuideAg* est celui qui lance le service, il demande à l'utilisateur de préciser sa destination. Chaque conducteur enregistre son profile dans le système embarqué dans sa voiture ou dans son smart-phone et il commence à chercher une place de parking qui corresponds le mieux à son profile et ses attentes.

Ce profile peut contenir des contraintes sur le prix, distance vers la place de parking, durée de stationnement, type de véhicule et des informations personnelles (Nom, prénom, sexe, numéro de téléphone, adresse..etc) pour rendre la facturation plus facile à faire. L'utilisation de ce service pour la première fois requiert des étapes à faire pour assurer la sécurité du système (comme un login et mot de passe) et à chaque fois l'utilisateur aura accès à ce service l'agent *AccountingAg* commence à interagir avec l'agent *ParkSelectorAg* d'une part pour des raisons de sécurité (authentifier) et d'autre part pour commencer la facturation au moment il gare.

l'agent *ParkSelectorAg* commence à récupérer les informations des agents *ParkingAreaAg* qui se trouvent dans une zone géographique près de sa destination et dans un deuxième temps cet agent commence à tirer les places qui correspondent à ses contraintes. Après avoir choisi la place il faut être guider pour y arriver, à ce moment où il intervient l'agent *DriverGuideAg* pour guider le conducteur vers sa place de parking choisie et de cette place jusqu'à sa destination finale, cet agent peut se débarrasser de son rôle comme un agent quand le système de guidage qui est embarqué va être utilisé indépendamment de notre service.

Pour que chaque agent satisfait son but il faut qu'il utilise ses connaissances locales avec des connaissances qui vont être acquises par la communications établies avec un autre agent. Prenant l'agent *ParkingAreaAg* qui groupe les différentes propriétés d'une place de parking. Cet agent mets à jour ses connaissance selon l'état de l'agent *ParkingSensorAg* , c'est à dire au moment l'agent *ParkSelectorAg* arrive à sa place de parking, l'agent *ParkingSensorAg* interagit avec l'agent *ParkingAreaAg* qui à son tour va changer la propriété de cette place (disponibilité et durée de la réservation..etc) et dans ce cas cette place n'affichera pas pour les autres agent *ParkSelectorAg*. D'autre part, l'agent *ParkingAreaAg* a un but d'être compétitif et d'avoir tous les places réservées durant toute la journée. Il est engagé à donner les informations nécessaires aux autres agents comme le transmis de la localisation géographique de la place de parking à l'agent *ParkSelectorAg*. L'agent *ParkSelectorAg* a comme but de trouver une place de parking qui satisfait le conducteur en mettant en avant les contraintes que ce dernier les mets au moyen de son propre profile. Cet agent utilise ses connaissances locales (profile).

# Chapitre 4

# Intégration avec la platforme M2M SensCity

Notre application est une application cliente de la plateforme SensCity. Cette plateforme offre un accès transparent aux capteurs et actionneurs déployés dans la ville. Comme montré dans la section 4.2, un système multi-agent est utilisé pour gérer l'infrastructure de SensCity, afin d'en assurer l'adaptabilité et la flexibilité. Les agents de notre système (*ParkingAreaAg*) interagissent donc avec les agents de la plateforme pour obtenir des informations sur l'état des places de parking et activer les réservations des utilisateurs. L'infrastructure et les ressources de la plateforme SensCity étant partagées, l'utilisation des devices par les applications doit faire l'objet d'un contrat, représenté sous forme d'organisation $\mathcal{M}OISE$.

Dans un premier temps, nous décrivons le contrat passé entre notre système et les agents devices en section 4.4 ensuite l'intégration d'un SMA dans la plateforme SensCity à l'aide d'artefacts en section 4.3.

## 4.1 L'infrastructure de la plateforme SensCity

Le projet SensCity [18] repose également sur la mutualisation la plus poussée des moyens, en partant d'un protocole de communication standardisé, en travaillant sur l'optimisation des passerelles, middleware standardisés de médiation et de courtage pour arriver au déploiement rapide de nouveaux services et en standardisant au maximum les IHM pour simplifier l'utilisation par les collectivités...

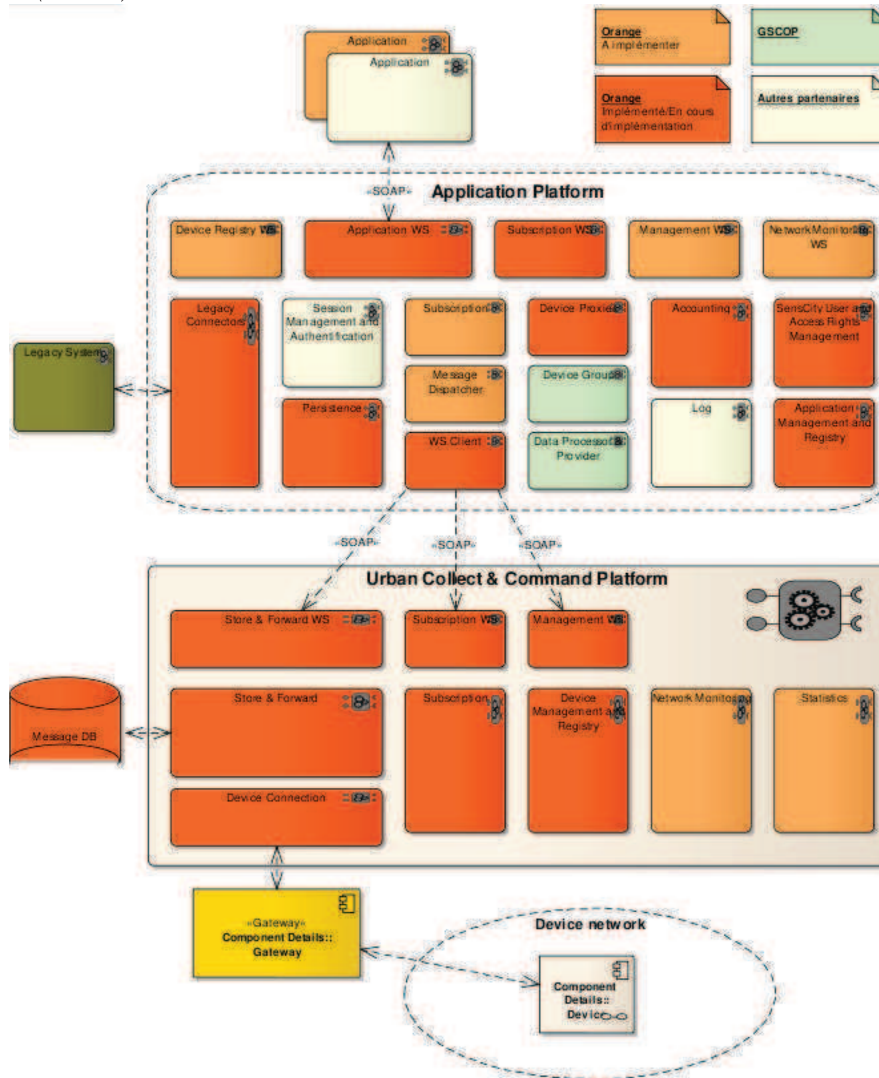
Le projet SensCity repose sur le passage d'une vision verticale à une vision horizontale, du réseau vers les services durables. De nombreux challenges restent à relever tel que :

- Standardisation de l'ensemble des couches
- Système de qualité de bout en bout pour les applications et les services
- Passage à l'échelle (millions d'objets ; dans une ville de 100 000 habitants avec une politique volontariste, il faut 2 ans pour déployer des compteurs d'eau communicants)
- Déploiement, maintenance et évolution de l'infrastructure
- Identification et adressage de billion d'objets
- Gérer la complexité d'interconnexion des bases de données
- Respecter la vie privée des individus

### 4.1.1 Application Platform (USP)

L'USP offre une couche métier aux applications utilisant les services de la plateforme SensCity. Cette plateforme permet aux applications de s'enregistrer et de s'abonner à différents types d'événements en provenance d'équipements. Elle garantit la sécurité de l'accès aux données grâce en particulier à une gestion de droits d'accès. La plateforme pourra dans certains cas décoder partiellement ou entièrement les trames des messages pour fournir aux applications des données de plus haut niveau. L'USP offre un service permettant de gérer la persistence des données durant une période configurable. La plateforme applicative de SensCity offre des interfaces aux applications et aux systèmes externes pour recevoir les données en provenance des équipements ou pour leur en envoyer. Les principaux composants sont :

FIGURE 4.1 – Architécture générale de SensCity : *Application Platform (USP)* et *Urban Collect & Command Platform (UCCP)*.




1. *Message Dispatcher :* Envoie les messages reçus de l'UCCP vers les destinations spécifiées par les abonnements.
2. *Subscription :* Gestion des abonnements des applications et des systèmes externes.
3. *Device Proxy :* Représente un équipement réel et exposant les services de cet équipement.
4. *Data Processor & Provider :* Composants génériques pouvant traiter des données et en fournir le résultat.
5. *Legacy Connector :* Framework permettant de développer des connecteurs sous forme de plugins.

### 4.1.2 Urban Collect & Command Platform (UCCP)

L'UCCP est la couche de collecte des messages et d'envoi des commandes du/aux équipements présents dans les différents réseaux déployés en milieu urbain. L'UCCP peut gérer différents réseaux utilisant des protocoles spécifiques. Cette plateforme sert de tampon pour les messages transitant entre les réseaux et les applications.

L'UCCP est typiquement gérée par un opérateur télécom. Cet opérateur offre un certain niveau de service dans l'acheminement des messages.

Cette platforme est utilisée avec tous ces composants afin d'avoir une architecture complète étendure à l'aide des composants qui vont être mises en place lors du déroulement du stage.

Cette couche de la plateforme présente des composants logiciels. La plateforme Urban Collect & Command Platform permet d'effectuer la collecte des messages et de fournir un service d'abonnement à ces messages. Les composants de cette couche sont :

1. *Device Management and Registry :* Prend en charge le provisionning des équipements et fournit les informations relatives aux équipements.
2. *Device :* Un équipement déployé à l'intérieur d'un réseau et communiquant vers la plateforme de collecte. Cet équipement peut soit (non exclusif) :
   (a) envoyer des données
   (b) recevoir des commandes
3. *Device Connection :* Composant assurant la connectivité avec les passerelles.
4. *Device Registry DB :* Base de données utilisée comme registre des équipements. Elle contient l'ensemble des équipements déployés et leurs propriétés :
   (a) applicatives (types de données échangées)
   (b) réseau (adresse, chemin, ...)
   (c) autre (coordonnées géographique, adresse postale, ...)
5. *Device Management and Registry :* Prend en charge le provisionning des équipements et fournit les informations relatives aux équipements. Le composant enregistre pour chaque composant des données :
   (a) applicatives (types de données échangées)
   (b) réseau (adresse, chemin, ...)
   (c) autre (coordonnées géographique, adresse postale, ...)
6. *Device Registry WS :* Interface Web Service permettant d'effectuer des requêtes sur le Device Registry pour récupérer des listes d'équipements en fonction de critères (emplacement, type, ...). L'interface notifie les applications lors de changements dans le réseau (ajout/suppression d'équipement, nouveau type d'équipement, ...).
7. *Management WS :* Interface web service pour la gestion de la plateforme applicative. Cette interface expose les services des composants suivants :
   (a) User and Access Right Management
   (b) Application Management and Registry
8. *Message DB :* La base de données stockant les message remontant des équipements (événements) et les messages descendant aux équipements (commandes). Les messages sont stockés dans cette base à accès rapide pendant une durée spécifiée par un contrat. Au delà de cette durée, les données sont archivées dans une autre base de données.
9. *Network Monitoring :* Composant détectant les dysfonctionnements et générant des événements le cas échéant. Il détecte des comportements anormaux du réseau et des ses éléments par rapport à des comportements paramètres. Ce composant dispose d'une interface de paramétrage des comportements attendus.
10. *Network Monitoring WS :* Interface web service exposant les services du composant Network Monitoring.
11. *Statistics :* Ce composant trace l'ensemble des messages envoyés/reçus par/vers les équipements ou la plateforme. Il stocke pour chaque message les informations nécessaires au calcul de statistique sur le réseau et la plateforme (date d'envoi, date de réception, priorité, taille, direction, ...) Il expose les données à des systèmes externes capables d'en effectuer le traitement.
12. *Store & Forward :* Collecte les message envoyées par les équipements et les stocke dans une base de données jusqu'à ce qu'il soient envoyés vers les applications abonnées ou récupérés par l'Application Platform. Ce composant permet aussi d'envoyer des messages à destination des équipements. Les messages dupliqués sont détectés et supprimés par ce composant. Les messages stockés sont archivés dans une autre base régulièrement à des fréquences paramétrables.

13. *Store & Forward WS :* Interface web service exposant les services du composant *Store & Forward*.
14. *Subscription :* Composant de gestion des abonnements de l'Application Platform aux événements envoyés par les équipements. Le composant notifie l'Application Platform lorsqu'un message correspondant à un abonnement est reçu.
15. *Subscription WS :* Interface web service exposant les services du composant Subscription.

## 4.2 Approche SMA pour le M2M appliquée à SensCity

Le paradigme Machine-to-Machine (M2M) implique des appareils (capteurs, effecteurs) interagissant pour fournir des services localisés dans le monde physique. Avec la maturité du M2M, une demande grandissante pour des solutions mutualisées dans lesquelles les applications peuvent partager un ensemble commun d'appareils émerge. Dans ce contexte, le projet Senscity propose une infrastructure pour mettre en œuvre des applications à l'échelle de la ville, ce qui nécessite de fournir des moyens de gouvernance agile pour prendre en compte l'extensibilité du système (ie. scalability). Par ailleurs, les technologies multi-agents apportent adaptation, flexibilité et pro-activité dans un cadre décentralisé. Ainsi, [19] propose une organisation multi-agent pour exprimer la stratégie de gouvernance de systèmes M2M. Grâce au cadre de modélisation organisationnelle MOISE, il illustre comment mettre en œuvre un SMA pour la gestion intelligente du stationnement.

## 4.3 Intégration d'un SMA dans la platforme SensCity à l'aide d'artefacts

La plateforme du projet SensCity repose sur des serveurs dont l'architecture est à base de composants (Java Beans). Chaque composant implémente une des fonctionnalités présentées et est encapsulé par un artefact correspondant. Cet artefact fournit un accès naturel aux composants pour les agents en fournissant des opérations correspondant aux méthodes du composant.

Afin de permettre aux agents de contrôler la plateforme de SensCity, nous utilisons des artefacts CArtAgO pour encapsuler les composants originales. Lorsque la méthode d'un composant est appelé elle est intercepté par l'artefact correspondant qui à son tour donne la main aux agents. La méthode est ensuite délégué au Java Bean d'origine afin que nous puissions réutiliser des traitements spécifiques (cryptage, base de données...).

Des composants ("beans") sont déclarés dans des fichiers de configuration (XML), le principe est de déclarer un artefact à la place du composant, et d'encapsuler les fonctionnalités en renvoyant les traitements au composant original après avoir mis à jour les propriétés observable et/ou déclenchés les signaux et linked operations. Comme les artefacts doivent être créés par le workspace, on utilise des classes "bridges" qui créent correctement l'artefact et le relie à la plateforme.

Le bean *componentBean* est implémenté par la classe *ComponentClass* implémentant l'interface *IComponent*. L'artefact *ComponentArt* et la classe *ComponentBridge* implémentent l'interface *IComponent*. On déclare la classe *ComponentBridge* pour le bean *componentBean*. On ajoute la déclaration du bean *originalComponentBean* avec la valeur de *ComponentClass*. A la création de la classe *ComponentBridge* crée l'artefact *ComponentArt* via le workspace en lui passant en paramètre *componentBean*. A chaque appel du bean *componentBean*, la classe *ComponentBridge* est appelé. La classe *ComponentBridge* redirige les appels vers *ComponentArt*. *ComponentArt* effectue les traitements *MAS* (updateObsProp, trigger event, signal), bloque éventuellement les traitements (@GUARD) pour donner la main aux agents, appel la classe *ComponentClass* pour effectuer le traitement coté "SensCity" et renvoi au préalable le résultat.

Le problème est qu'un objet Java ne peut pas utiliser directement les artefacts. les artefacts doivent être utilisés par les agents ou par d'autres artefacts. pour cette raison nous utilisons ce que nous appelons des «agents de pont» (bridge agents). Ces agents créent des artefacts qui correspondent aux composantes si elles n'existent pas encore, transfèrent l'appel à la méthode correspondante et retournent le résultat de l'opération.

Prenant l'exemple du composant DeviceCNX, ce composant reçoit / envoie des messages de / vers les devices via une passerelle. Sa déclaration est faite en utilisant un fichier XML (devicecnx.xml) comme suit :

```
<bean id="commandSender" class="com.francetelecom.citypulse.devicecnx.CommandSender"/>
<bean id="eventReceiver" class="com.francetelecom.citypulse.devicecnx.EventReceiver"/>
```

Lors du démarrage de la plateforme une classe DeviceCNXFactory lance ces trois beans :

Listing 4.2 – DeviceCNXFactory.java

```
XmlBeanFactory bf = new XmlBeanFactory(new ClassPathResource("/" + CONFIG\_FILE,
    getClass()));
handler = IHandler.class.cast(bf.getBean("eventReceiver"));
commandSender = ICommandSender.class.cast(bf.getBean("commandSender"));
```

L'intégration des artefacts peut être fait facilement en déclarant l'agent de pont au lieu des deux beans, à condition qu'elle implémente les mêmes interfaces :

```
<bean id="commandSender" class="com.francetelecom.citypulse.mas4m2m.artifacts.devicecnx.
    DeviceCNXBridge"/>
<bean id="eventReceiver" class="com.francetelecom.citypulse.mas4m2m.artifacts.devicecnx.
    DeviceCNXBridge" />
<bean id="originalCommandSender" class="com.francetelecom.citypulse.devicecnx.
    CommandSender"/>
<bean id="originalEventReceiver" class="com.francetelecom.citypulse.devicecnx.
    EventReceiver" />
```

Avoir une référence à l'implémentation originale permet à l'agent de faire le "lien" avec l'artefact :

Quand l'agent de pont est appelé , chaque appel à l'une des méthodes de composant est géré par l'agent du pont qui invoque les opérations d'artefacts. Il obtient le résultat de l'opération via un OpFeedbackParameter et le retourne :

Listing 4.4 – DeviceBridge.java

```
public boolean sendCommandToDevice(Integer deviceId, MessagePriority priority,
  Map<PayloadType, byte[]> payloads, String gateway,
  Byte sequenceNumber, String token)
  throws ResourceNotFoundFaultException, TechnicalFaultException,
  AccessRightFaultException, LoginFaultException {
  try {
    OpFeedbackParam<Boolean> result = new OpFeedbackParam<Boolean>();
    OpFeedbackParam<Exception> exception = new OpFeedbackParam<Exception>();
    doAction(this.deviceCNXArt, new Op("sendCommandToDevice", deviceId,
      priority, payloads, gateway, sequenceNumber,
      token, result, exception ));
    if(exception.get() == null)
      return result.get();
    if(exception.get().getClass() == ResourceNotFoundFaultException.class)
      throw (ResourceNotFoundFaultException)exception.get();
    if(exception.get().getClass() == TechnicalFaultException.class)
      throw (TechnicalFaultException)exception.get();
    if(exception.get().getClass() == AccessRightFaultException.class)
      throw (AccessRightFaultException)exception.get();
    if(exception.get().getClass() == LoginFaultException.class)
      throw (LoginFaultException)exception.get();
  } catch (CartagoException e) {
    e.printStackTrace();
  }
}
```

Enfin, l'artefact DeviceCNXArt notifie les vrais agents de leur activité et éventuellement leur donner la main pour contrôler l'opération. Artefacts "presque" implémente les interfaces, sauf les méthodes sont convertis à l'opération :

Listing 4.5 – DeviceCNXArt.java

```
@OPERATION
public void sendCommandToDevice(Integer deviceId, MessagePriority priority,
```

```
3         Map<PayloadType, byte[]> payloads, String gateway,
          Byte sequenceNumber, String token,
5         OpFeedbackParam<Exception> result,
          OpFeedbackParam<Exception> exceptions){
7   /* *** give the hand to the agents *** */
    callID = makeCmdObservable(deviceId, priority, payloads, gateway,
9       sequenceNumber, token);
    signal(this.agentManager, "CALL:sendCommandToDevice",callID);
11  await("proceed",callID); //TODO add a timeOut parameter
    /* *** call the original component *** */
13  try {
      result.set(this.originalCommandSender.sendCommandToDevice(deviceId,
15    priority, payloads, gateway, sequenceNumber, token);
    } catch (Exception e) {
17    exceptions.set(e);
    }
19 }
```

Le développement des artefacts est détaillé en annexe A et celui du workspace est en annexe B. Le développement est toujours en cours donc le code source risque d'être changé.

Le projet SensCity est peu documenté avec des spécifications obsolètes ce qui nécessitait un travail de rétro ingénierie (voir la table 4.1 et la table 4.2 et qui est une suite de la première ). les préfixes dans la table sont comme suit :

p1=com.francetelecom.citypulse.interfaces

p2=com.francetelecom.citypulse

Table 4.1 – Les différents "beans" avec leurs ID et les interfaces de SensCity

| Noms des "beans" | ID | Interfaces |
|---|---|---|
| *uccp/context* | uccpClient, applicationServices, applicationFactory, sessionServices, sessionFactory, messageServices, messageFactory | p1.internal.client.uccp. IUccpClient, p1.internal.services.uccp. IApplicationServices, p1.internal.services.uccp. ISessionServices, p1.internal.services.uccp. IMessageServices |
| *application − messagedispatcher* | messageDispatcher | p2.application.messagedispatcher.IMessageDispatcher |
| *session − manager* | sessionManager | – |
| *device − cnx* | commandSender, eventReceiver, internalMessageHandler, serverKeystorePath, serverKeystorePassword, clientKeystorePath, clientKeystorePassword | p2.devicecnx.ICommandSender, Ihandler(interface doesn't exist in the source code) et p2.devicecnx.IInternalMessageHandler |
| *facadeContext* | deviceFacade, deviceAdminFacade, wdcManager | p2.facade.device.IDeviceFacade p2.facade.device.IDeviceAdminFacade et p2.facade.device.wdc.IWdcManager |
| *clientapplicationcontext* | notificationServices, notificationFactory, uspClientApplicationClient | p1.external.services.INotificationServices et p1.external.client.clientapplication.IUspClientApplicationClient |
| *clientapplication − manager* | clientApplicationManager | p2.clientapplication.IClientApplicationManager |
| *messageSimulator* | device-simulator,sender | p2.simulator.message.device.IDeviceSimulator et p2.simulator.message.IMessageSender |
| *storeAndForwardContext* | storeAndForward | – |
| *applicationNotifierContext* | applicationNotifier | p2.forward.IApplicationNotifier |
| *internal.client.usp/context* | notificationServices, notificationFactory et internalUspClient | p1.internal.services.usp.INotificationServices, p1.internal.services.usp.INotificationServices et p1.internal.client.usp.IInternalUspClient |
| *storeContext* | storeManager | p2.store.IStoreManager |
| *applicationTest* | applicationTest | – |
| *application − configurator* | applicationConfigurator | p2.application.configuration.IApplicationConfigurator |
| *application − dataextractor* | applicationDataExtractor | p2.application.dataextractor.IDataExtractor |
| *application − store* | applicationStore | p2.application.datastore.IApplicationStore |
| *application − session* | sessionManager | p2.application.session.ISessionManager |
| *.internal.client.uccp/context.xml* | uccpClient | p1.internal.client.uccp.IUccpClient |

TABLE 4.2 – Les classes SensCity les artefacts attribués

| Classes SensCity | Artefacts |
|---|---|
| p1.internal.client.uccp.UccpClient, p2.application.ws.services. ApplicationServiceImpl, p2.ws.services.SessionServicesImpl et p2.ws.services. MessageServiceImpl | UccpClientArt |
| p2.application.messagedispatcher.MessageDispatcher | UccpClientArt |
| p2.session.SessionManager | UserAuthArt |
| p2.devicecnx.CommandSender, p2.devicecnx.EventReceiver et p2.devicecnx.InternalMessageHandler | DeviceCNXArt |
| p2.facade.device.DeviceFacade, p2.facade.device.DeviceAdminFacade et p2.facade.device.wdc.WdcManager | DeviceArt, DeviceRegisteryArt et USPCNXArt |
| p2.application.ws.services.NotificationServicesImpl(doesn't exist in the xml) et p1.external.client.clientapplication.UspClientApplicationClient | USPCNXArt |
| p2.clientapplication.CacheClientApplicationManager | ApplicationCNXArt |
| p2.simulator.message.device.WaterMeteringDeviceSimulator et p2.simulator.message.MessageSender | – |
| – | – |
| p2.forward.ApplicationNotifier | ApplicationCNXArt |
| p2.application.ws.services.NotificationServicesImpl(doesn't exist in the xml) et p1.internal.client.usp.InternalUspClient | – |
| p2.store.StoreManager | – |
| p2.uccpclient.apitests.ApplicationTest | – |
| p2.application.configuration.ApplicationConfigurator | ApplicationCNXArt |
| p2.application.dataextractor.DataExtractor | – |
| p2.application.datastore.ApplicationStore | ApplicationCNXArt |
| p2.application.session.SessionManager | USPCNXArt |
| p1.internal.client.uccp.UccpClient | UccpClientArt |

## 4.4 Contrat d'utilisation des devices

L'application développée passe un contrat avec les agents *DeviceAg* qui peuvent jouer 3 rôles différents et les agents *ParkingAreaAg* qui peuvent jouer 2 rôles qui sont joués aussi par l'agent *DeviceAg*

Chaque agent *M2MAreaAg* passe un contrat avec un ensemble d'agents *DeviceAg* qui gèrent les capteurs situés dans sa zone. L'utilisation de contrat est pertinente à base d'une plateforme comme SensCity qui se décompose en plusieurs composants. Afin d'adapter le contrat aux modèles de composants hiérarchiques, nous avons proposé la forme d'organisation MOISE, qui gère le contrat pour faire la liaison entre les composants du système. Donc ce contrat prend la forme d'organisation MOISE.

Idem à ce qu'on a déjà vue dans la section 3.1 MOISE décrit une organisation suivant deux dimensions indépendantes : (i) la spécification structurelle qui définit les rôles et les groupes dont les agents peuvent s'engager et (ii) la spécification fonctionnelle qui représente un schéma social que le système multi-agent doit remplir avec des rôles et des missions qui abordent ces rôles. Les deux dimensions sont liées par la spécification normative qui relient chaque rôle aux missions. Ces différentes spécifications sont décrites dans les sections suivantes.

### 4.4.1 Rôles

La structure des contrats (structure structurelle)passées par chaque agent définit la structure du système multi-agents . Cette spécification structurelle définit les différentes relations qui existent entre les *agents DeviceAg*, *ParkingPlaceAg*, *GWag* et *ParkingAreaAg*.

Ces agents sont définis à travers des notions de *groupes* : (i) gpM2MParking, (ii) gpDevices et (iii) gpAgreement pour représenter le niveau collectif de la spécification structurelle sachant que chaque groupe un contrat.

Pour représenter le niveau individuel de la structure structurelle le groupe rassembles de rôles (i) *ParkingApp*, (i) *ParkingSensor*, *ParkingGW*, *ParkingRepeater* et *ParkingActuator*, et de *liens* entre les rôles.

Au niveau social et afin de de contraindre et structurer les agents en fonction de leurs rôles qui jouent des liens s'établissent entre les différents rôles de chaque groupe. Le comportement de l'agent *ParkingAreaAg* est décrit par le rôle *ParkingApp*.

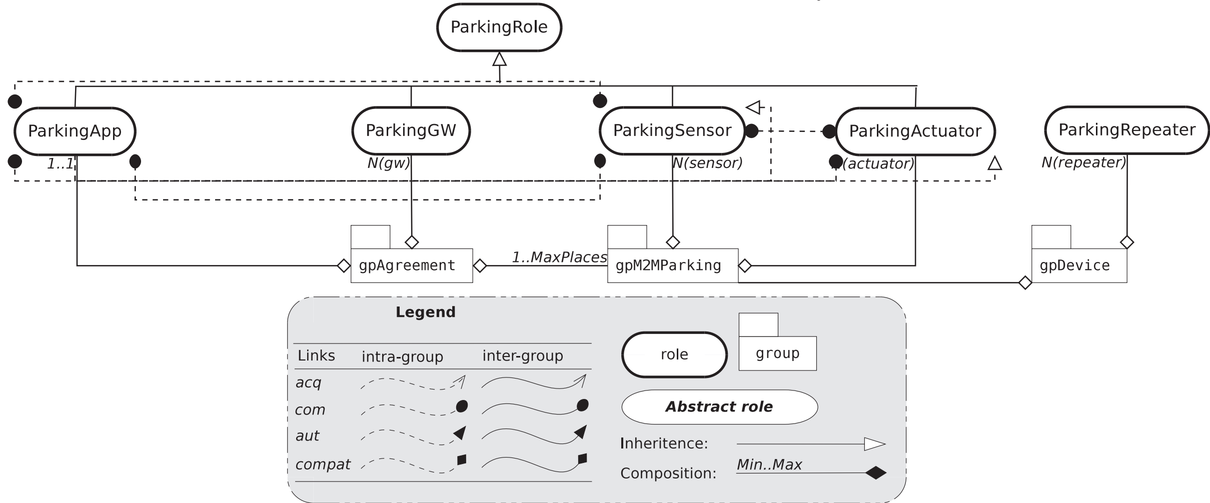
Le rôle *ParkingSensor* et *ParkingActuator* décrivent le comportement attendu de l'agent *ParkingPlaceAg*. Ils sont responsable de transférer les informations sur les places de parking correspondantes.

Un agent *GWAg* joue le rôle *ParkingGW* pour assurer la communication au niveau $GSM$. Le rôle *ParkingApp* est joué par l'agent *ParkingAreaAg*.

La figure 4.2 montre d'une façon explicite la spécification structurelle pour un système M2M.

L'agent *Parking*

FIGURE 4.2 – Spécification structurelle pour un système M2M.

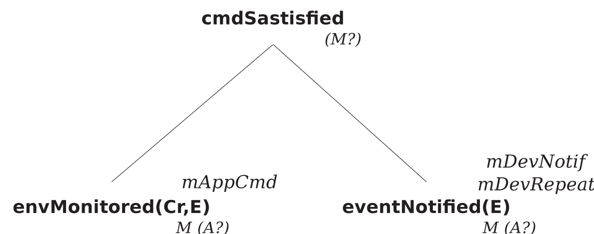



## 4.4.2 Surveillance des places de parking

La surveillance des places de parking est l'une des actions importantes dans notre système. Cette activité est faite par les agents *DeviceAg* qui jouent le rôle *ParkingRepeater* et qui peuvent prendre le forme des agents *ParkingPlaceAg* qui jouent le rôle *ParkingSensor* et *ParkingActuator*. L'agent *ParkingSensorAg* est un capteur qui détecte la présence d'un véhicule en envoyant des informations sur le statut de chaque place de parking. Pour que l'agent *ParkingPlaceAg* arrive à son but, lors de stationnement de l'utilisateur dans la place sélectionnée une détection prend place pour informer la zone à laquelle cet agent appartient que son statut a été changé. Le statut peut prendre une de deux valeurs (libre ou occupé) . Cette détection est communiqué à l'agent *ParkingPlaceAg* par le capteur qu'il gère et qui détecte la présence d'un véhicule au moyen d'un changement de champs magnétique.

Le but *cmdSatisfied* est satisfait par une séquence des buts *envMonitered* et *eventNotified* (voir figure 4.3). Le but *envMonitered* est un but de maintenance qui surveiller l'environnement pour les événements qui correspondent aux critères. Ce but est associé à la mission *mAppCmd*. Le but *eventNotified* est un but de maintenance qui notifie les événement des événements sensés. Ce but est associé aux missions *mDevNotif* et *mDevRepeat*.

FIGURE 4.3 – Schéma social : Execution du contrat par les *capteurs*.

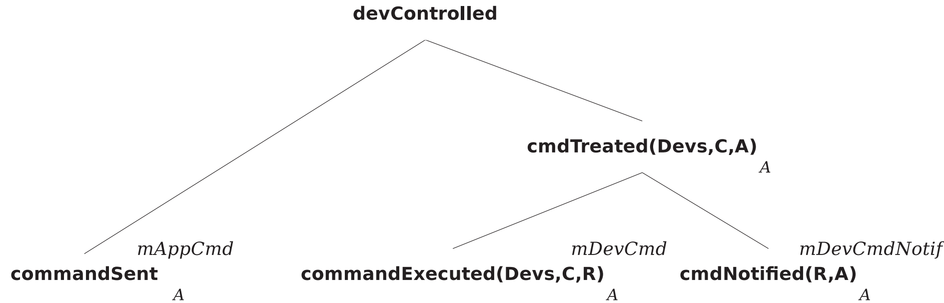



## 4.4.3 Réservation d'une place

Pour réserver une place de parking, plusieurs agents entrent en jeux pour assurer la bonne fonctionnement du système entier. Ces agents ont un but a atteindre et des mission pour en souscrire (voire la

figure 4.4). le but *devControlled* est un but de réalisation (Achievement) qui exécute une commande envoyée par l'application Il est satisfait par une séquence des buts *commandSent* et *cmdTreated*. Le but *commandSent* assure que la commande est envoyée au device, c'est un but de réalisation. Il est associé à la mission *mAppCmd*. Le but *cmdTreated* traite la commande et il est satisfait par une séquence des buts *commandExecuted* et *cmdNotified*. Le but *commandExecuted* est un but de réalisation pour dire que la commande est exécuté par le(s) device(s) et il est associé à la mission *mDevCmd*. le but *cmdNotified* est un but de réalisation pour notifier l'application du résultat de la commande et il est associé à la mission *mDevCmdNotif*

FIGURE 4.4 – Schéma social : Envoi de commande aux *effecteurs* et *capteurs*.




### 4.4.4 Normes représentant le contrat

La spécification normative définit les normes pour contraindre les actions que les agents peuvent faire. La table 4.3 regroupe les normes dans le système avec les conditions, rôles, relations, mission et temps de s'incrire à une mission.

TABLE 4.3 – Spécifications normative pour le système M2M

| ID | Condition | Rôle | Relation | Mission | TTF |
|---|---|---|---|---|---|
| $n_{01}$ | _ | *ParkingApp* | permission | *mAppCmd* | _ |
| $n_{02}$ | command(sensor_request(Cmd),Devs,_) | *ParkingSensor* | obligation | *mAuth* | ttfSensor |
| $n_{03}$ | command(actuator_request(Cmd),Devs,_) | *ParkingActuator* | obligation | *mDevCmd* | ttfReserv |
| $n_{04}$ | command(Cmd,_,Result) | *ParkingSensor* | obligation | *mDevNotif* | ttfResNotif |
| $n_{05}$ | command(Cmd,_,Result | *ParkinActuator* | obligation | *mDevNotif* | ttfResNotif |
| $n_{06}$ | _ | *ParkingSensor* | obligation | *mDevMonit* | ttfmDevMonit |
| $n_{07}$ | _ | *ParkingSensor* | obligation | *mDevRepeat* | ttfmDevRepeat |
| $n_{08}$ | _ | *ParkingSensor* | obligation | *mDevNotif* | ttfmDevNotif |

Le norme $n_{01}$ signifie que *ParkingApp* a la permission d'envoyer la commande aux devices. Le norme $n_{02}$ : les capteurs ont l'obligation de traiter la commande en temps ttfSensor. Le norme $n_{03}$, les parcelles de réservation ont l'obligation de traiter la commande en temps ttfReserv. Les normes $n_{04}$ et $n_{05}$, les capteurs et les parcelles de réservation ont l'obligation de notifier le résultat de la commande.

La partie développement se trouve en annexe C.1. Cette partie est toujours en cours et il y aura des changements à faire.

# Chapitre 5

# Conclusion et perspectives

On présente dans ce chapitre une conclusion du travail effectué et les futurs travaux à faire dans la même problématique.

## 5.1 Bilan du travail réalisé

Le concept de « villes intelligentes » repose sur le déploiement d'un grand nombre de capteurs et d'actionneurs permettant d'automatiser et d'améliorer les services urbains, (gestion de l'éclairage public, le ramassage des ordures) mais aussi de favoriser les interactions entre les services et les usagers (place de parking, transports public, gestion de crise). On appelle ce type système regroupant de tels service des systèmes machine-to-machine (M2M).

De tels systèmes requièrent des propriétés importantes d'autonomie, de réactivité et de proactivité afin de pouvoir s'adapter et d'être évolutifs. Il est important que ces systèmes puissent passer à l'échelle. Les systèmes multi-agents (MAS) permettent de développer des applications distribuées en définissant les différentes entités autonomes du système (agents) et leur interactions aussi bien entre elles qu'avec l'environnement extérieur (organisation). Les agents sont capables de communiquer entre eux afin d'adapter leur organisation en fonction de la situation ou pour répondre à de nouveaux objectifs. Ce type de système est donc particulièrement adapté à notre contexte de système machine-to-machine.

Le sujet de ce stage a consisté à participer à l'intégration d'un système multi-agents au sein des différents composant d'une architecture M2M. Il s'agissait de proposer et de mettre en œuvre l'intégration d'un système multi-agent dans l'infrastructure du projet ANRT SensCity. Le travail restant consistera également à implémenter, tester et à valider l'approche sur la base d'un scénario gestion de place de parking.(en cours jusqu'à la fin de mon stage en 26 Août 2011)

### 5.1.1 Les techniques

La réalisation du travail sera réalisé grâce à l'outil JaCaMo (Jason + CArtAgO + MOISE). Les agents sont développés en Jason, l'environnement en CArtAgO et la structure organisationnelle en Moise.

Le développement de logiciels pour des environnements complexes dynamiques où l'autonomie est nécessaire est notoirement difficile. Une approche plus en plus populaire pour traiter avec une telle tâche est de concevoir et mettre en œuvre un tel logiciel à l'aide de la technologie de système multi-agents en utilisant ces trois outils.

### 5.1.2 Bilan personnel du stage

Les résultats de mon stage portent sur différents axes. L'axe théorique qui a apporté l'intelligence dans une application basée sur des capteurs urbains à l'aide d'un système multi-agent et l'axe pratique qui a définit un modèle de gouvernance Multi-Agent pour une application de parking avec l'implémentation et intégration dans l'infrastructure M2M du projet SensCity. La proposition de gestion intelligente de parking contribue à trouver une place de parking le mieux adapté aux profiles de chaque utilisateur. Un algorithme est présenté qui aide à toujours améliorer la place trouvée en effectuant une fonction d'évaluation pour les places.

## 5.2 Perspectives

Le proposition de gestion intelligente de parking reste toujours à améliorer. Dans ce contexte, notre proposition se base sur une interaction entre l'agent du côté utilisateur et celui du côté zone de parking. Une amélioration peut être apportée par une coopération entre les différentes zones de parking pour attribuer une place de parking qui correspond le mieux possible à l'utilisateur.

Au niveau de la satisfaction de l'utilisateur, le scénario proposé, dans le cas où l'utilisateur ne trouve pas une place de parking dans une zone donnée. La solution de notre système était d'augmenter la zone dans laquelle on cherche la place, mais une autre proposition peut apporter une amélioration au système au niveau de chaque critère. C'est à dire que si l'utilisateur ne trouve pas une place qui correspond totalement à ses critères, on commence par éliminer un critère avant l'autre en commençant par le critère le moins important jusqu'à ce qu'on arrive à trouver une place qui satisfasse le mieux possibles les attentes de l'utilisateur.

L'échange de places entre les conducteurs peut être une solution pour améliorer le système tout entier. Si un conducteur 1 à un moment donné a trouvé une place dans une zone A, étant donné que c'était préférable pour lui qu'il en trouve une dans la zone B et un autre conducteur 2 a trouvé une place dans une zone B tandis qu'il était préférable pour lui qu'il en trouve une dans la zone A. Dans ce cas, un échange des places entre les conducteurs 1 et 2 est possible pour satisfaire les deux ce qui optimiser du système.

Une place choisie par un utilisateur risque d'être occupé par un autre avant qu'il en arrive donc, on peut introduire un système embarqué dans chaque véhicule et utiliser la technologie RFID dans chaque place de parking qui seront plus au moins suffisantes pour s'assurer que le bon conducteur est garé dans la place qu'il a choisie par le biais de notre système.

Le système Muli-agent est intégré dans mon sujet de stage dans l'un des services qui est la gestion intelligente de parking. Les systèmes Multi-agents pourraient être utilisés pour d'autres services urbains tel que l'exploration de la viabilité des systèmes irrigués : dynamique des interactions et modes d'organisation.

# Bibliographie


[1] L. A. Methodologie and R. C. Les, “La recherche d’une place de stationnement : strategies, nuisances associees, enjeux pour la gestion du stationnement en france,” pp. 1–5, 2005.

[2] ETSI, ““ETSI TS 102 690 V<0.10.1>, Machine-to-Machine communications (M2M) ; Functional architecture,”,” pp. 1–201, 2011.

[3] FING, S. Informatique, and Orange, *Livre Blanc,Machine-To-Machine :enjeux et perspectives*, 2006.

[4] M. Y. I. Idris, E. Tamil, Z. Razak, N. Noor, and L. Kin, “Smart Parking System using image processing techniques in wireless sensors network environment,” 2009.

[5] R. Lu, X. Lin, H. Zhu, and X. Shen, *SPARK : A New VANET-Based Smart Parking Scheme for Large Parking Lots*. IEEE, Apr. 2009. [Online]. Available : http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5062057

[6] S. Srikanth, P. Pramod, K. Dileep, S. Tapas, M. U. Patil, and S. C. Babu N, “Design and Implementation of a Prototype Smart PARKing (SPARK) System Using Wireless Sensor Networks,” *2009 International Conference on Advanced Information Networking and Applications Workshops*, pp. 401–406, May 2009. [Online]. Available : http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5136681

[7] G. Yan, S. Olariu, M. C. Weigle, and M. Abuelela, “SmartParking : A Secure and Intelligent Parking System Using NOTICE,” *2008 11th International IEEE Conference on Intelligent Transportation Systems*, pp. 569–574, Oct. 2008. [Online]. Available : http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4732702

[8] I. devices, “Intelligent parking developped by intelligent devices (entreprise : intelligent devices).” [Online]. Available : http://www.intelligentparking.com

[9] Lyberta, “Parking intelligent,” 2009. [Online]. Available : http://www.lyberta.com/

[10] Optipark, “SpotSmart _ Intelligent parking using wireless sensor networks.” [Online]. Available : http://www.optipark.eu/

[11] Streetline, “Parker.” [Online]. Available : http://www.streetlinenetworks.com/

[12] S.-Y. Chou, S.-W. Lin, and C.-C. Li, “Dynamic parking negotiation and guidance using an agent-based platform,” *Expert Systems with Applications*, vol. 35, no. 3, pp. 805–817, Oct. 2008. [Online]. Available : http://linkinghub.elsevier.com/retrieve/pii/S095741740700293X

[13] G. Ivan, M. O’Droma, and M. Damien, “INTELLIGENT CAR PARKING LOCATOR SERVICE,” *Information Technologies and Knowledge*, vol. 2, pp. 166–173, 2008.

[14] D. Teodorovit and M. Dell’Orco, *MULTI AGENT SYSTEMS APPROACH TO PARKING FACILITIES MANAGEMENT*. Springer, 2005, pp. 321–339.

[15] A. B. Juan and A. Muñoz, “Developing an Intelligent Parking Management Application Based on Multi-agent Systems and Semantic Web Technologies,” *Hybrid Artificial Intelligence Systems*, pp. 64–72, 2010.

[16] W. Longfei and C. Hong, “Coorporative Parking Negotiation and Guidance Based on Intelligent Agents,” *Expert Systems with Applications*, pp. 76–79, Jun. 2009. [Online]. Available : http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5231037

[17] Z. Pala and N. Inan, “Smart parking applications using RFID technology,” *International Journal*, pp. 21–23, 2007.

[18] V. Giméno, "Projet SensCity," *Living things*, 2010.

[19] C. Persson, G. Picard, and F. Ramparany, "A Multi-Agent Organization for the Governance of Machine-To-Machine Systems," *Sensors (Peterborough, NH)*, 2011.

# Annexe A

# Artefacts

## A.1 *ApplicationCNXArt*

Listing A.1 – ApplicationCNXArt.java

```java
package com.francetelecom.citypulse.mas4m2m.artifacts;

import java.util.Set;

import com.francetelecom.citypulse.clientapplication.ClientApplication;
import com.francetelecom.citypulse.clientapplication.IClientApplicationManager;
import com.francetelecom.citypulse.forward.IApplicationNotifier;
import com.francetelecom.citypulse.interfaces.internal.exception.business.
    ResourceNotFoundFaultException;
import com.francetelecom.citypulse.interfaces.internal.exception.technical.
    TechnicalFaultException;
import com.francetelecom.citypulse.store.model.Event;

import cartago.Artifact;
import cartago.OPERATION;

public class ApplicationCNXArt extends Artifact implements IClientApplicationManager,
    IApplicationNotifier {
  private IClientApplicationManager originalClientApplicationManager;
  private IApplicationNotifier orginialApplicationNotifier;

  @OPERATION public void init(IClientApplicationManager originalClientApplicationManager
      ,IApplicationNotifier orginialApplicationNotifier){
    this.originalClientApplicationManager=originalClientApplicationManager;
    this.orginialApplicationNotifier = orginialApplicationNotifier;
  }
  @Override
  public Set<String> getAllApplicationNames() throws TechnicalFaultException {
    return this.originalClientApplicationManager.getAllApplicationNames();
  }

  @Override
  public ClientApplication getClientApplication(String instanceLogin)
      throws TechnicalFaultException, ResourceNotFoundFaultException {
    return this.originalClientApplicationManager.getClientApplication(instanceLogin);
  }

  @Override
  public Set<ClientApplication> getConcernedApplications(
      Set<Integer> organizationIds) throws TechnicalFaultException {
    return this.originalClientApplicationManager.getConcernedApplications(
        organizationIds);
  }

  @Override
  public void registerNewClientApplication(ClientApplication newInstance)
```

```
        throws TechnicalFaultException {
43      this.originalClientApplicationManager.registerNewClientApplication(newInstance);
    }
45
    @Override
47  public void removeClientApplication(ClientApplication instanceToRemove)
        throws TechnicalFaultException {
49      this.originalClientApplicationManager.removeClientApplication(instanceToRemove);
    }
51  @Override
    public void addEvent(Integer deviceTypeId, Set<Integer> organizationIds,
53      Event event) {
      this.orginialApplicationNotifier.addEvent(deviceTypeId, organizationIds, event);
55  }
    @Override
57  public boolean isStarted() {
      return this.orginialApplicationNotifier.isStarted();
59  }
    @Override
61  public void startNotification() {
      this.orginialApplicationNotifier.startNotification();
63  }
    @Override
65  public void stopNotification() {
      this.orginialApplicationNotifier.stopNotification();
67  }

69 }

71 }
```

## A.2 *DeviceRegisteryArt*

Listing A.2 – DeviceRegisteryArt.java

```
1  package com.francetelecom.citypulse.mas4m2m.artifacts;

3  import java.util.List;

5  import com.francetelecom.citypulse.application.session.SessionManager;
   import com.francetelecom.citypulse.application.session.SessionManagerFactory;
7  import com.francetelecom.citypulse.facade.FacadeFactory;
   import com.francetelecom.citypulse.facade.device.DeviceAdminFacade;
9  import com.francetelecom.citypulse.facade.device.IDeviceAdminFacade;
   import com.francetelecom.citypulse.facade.device.IDeviceFacade;
11 import com.francetelecom.citypulse.facade.device.DeviceAdminFacadeConstants.
       COM_DIRECTIONS;
   import com.francetelecom.citypulse.facade.device.model.Device;
13 import com.francetelecom.citypulse.facade.device.model.DeviceAddress;
   import com.francetelecom.citypulse.facade.device.model.DeviceAddressInfo;
15 import com.francetelecom.citypulse.facade.device.model.DeviceAddressType;
   import com.francetelecom.citypulse.facade.exception.DeviceUpdateException;
17 import com.francetelecom.citypulse.interfaces.internal.exception.business.
       AccessRightFaultException;
   import com.francetelecom.citypulse.interfaces.internal.exception.business.
       LoginFaultException;
19 import com.francetelecom.citypulse.interfaces.internal.exception.business.
       ResourceNotFoundFaultException;
   import com.francetelecom.citypulse.interfaces.internal.exception.technical.
       TechnicalFaultException;
21 import com.francetelecom.citypulse.interfaces.model.device.DeviceType;
   import com.francetelecom.citypulse.interfaces.model.device.Organisation;
23 import com.francetelecom.citypulse.store.model.Message;

25 import cartago.Artifact;
   import cartago.OPERATION;
27 import cartago.OpFeedbackParam;
```

```
29 /**
    *
31  * @author Mustapha Bilal
    *
33  */
   public class DeviceRegisteryArt extends Artifact implements IDeviceAdminFacade{
35   private String adminToken;
     private IDeviceAdminFacade originalDeviceAdminFacade;
37
     @OPERATION public void init(IDeviceAdminFacade originalDeviceAdminFacade){
39     this.originalDeviceAdminFacade=originalDeviceAdminFacade;
     /**
41    *
      * @param device
43    */
     @OPERATION public void isDeviceAuthorized(DeviceArt device){
45     try {
         originalDeviceAdminFacade.isDeviceAuthorized(getObsProperty("deviceAddress").
             stringValue());
47     } catch (TechnicalFaultException e) {
         e.printStackTrace();
49     }

51   }

53   @Override
     public com.francetelecom.citypulse.facade.device.model.Device activateDevice(
55       com.francetelecom.citypulse.facade.device.model.Device device,
         Organisation organisation) throws TechnicalFaultException {
57     return this.originalDeviceAdminFacade.activateDevice(device, organisation);
     }
59
     @Override
61   public com.francetelecom.citypulse.facade.device.model.Device createDevice(
         com.francetelecom.citypulse.facade.device.model.Device device,
63       DeviceType deviceType, Organisation organisation,
         com.francetelecom.citypulse.facade.device.model.Device gateway)
65       throws ResourceNotFoundFaultException, TechnicalFaultException {
       defineObsProperty("newDevice", device,device.getDeviceType(),device.getId(),device.
           getName(),device.getStatus());
67     signal("newDevice", device.getId());
       return this.originalDeviceAdminFacade.createDevice(device, deviceType, organisation,
           gateway);
69   }

71   @Override
     public DeviceAddressType createDeviceAddressType(String name,
73       COM_DIRECTIONS direction) throws TechnicalFaultException {
       return this.originalDeviceAdminFacade.createDeviceAddressType(name, direction);
75   }

77   @Override
     public DeviceType createDeviceType(String deviceTypeName,
79       Organisation organisation, COM_DIRECTIONS commDirection)
         throws TechnicalFaultException {
81     return this.originalDeviceAdminFacade.createDeviceType(deviceTypeName, organisation,
           commDirection);
     }
83
     @Override
85   public List<DeviceType> getAllDeviceTypeList()
         throws TechnicalFaultException, LoginFaultException {
87     return this.originalDeviceAdminFacade.getAllDeviceTypeList();
     }
89
     @Override
91   public DeviceAddressInfo getDeviceAddressInfo(DeviceAddress address)
         throws TechnicalFaultException, ResourceNotFoundFaultException {
93     return this.originalDeviceAdminFacade.getDeviceAddressInfo(address);
```

```
    }
95
    @Override
97  public DeviceAddressType getDeviceAddressTypeById(Integer addressTypeId)
        throws TechnicalFaultException, ResourceNotFoundFaultException {
99    return this.originalDeviceAdminFacade.getDeviceAddressTypeById(addressTypeId);
    }
101
    @Override
103 public DeviceAddressType getDeviceAddressTypeByName(String addressTypeName)
        throws TechnicalFaultException, ResourceNotFoundFaultException {
105   return this.originalDeviceAdminFacade.getDeviceAddressTypeByName(addressTypeName);
    }
107
    @Override
109 public List<DeviceAddressType> getDeviceAddressTypeListForDeviceType(
        DeviceType deviceType, COM_DIRECTIONS direction)
111     throws ResourceNotFoundFaultException, TechnicalFaultException {
      return this.originalDeviceAdminFacade.getDeviceAddressTypeListForDeviceType(
          deviceType, direction);
113 }

115 @Override
    public com.francetelecom.citypulse.facade.device.model.Device getDeviceByAddress(
117     DeviceAddress address) throws TechnicalFaultException,
        ResourceNotFoundFaultException {
119   return this.originalDeviceAdminFacade.getDeviceByAddress(address);
    }
121
    @Override
123 public com.francetelecom.citypulse.facade.device.model.Device getGatewayForDevice(
        com.francetelecom.citypulse.facade.device.model.Device device)
125     throws TechnicalFaultException, ResourceNotFoundFaultException {
      return this.originalDeviceAdminFacade.getGatewayForDevice(device);
127 }

129 @Override
    public List<com.francetelecom.citypulse.facade.device.model.Device> getGatewayList(
131     Organisation organisation) throws TechnicalFaultException,
        ResourceNotFoundFaultException {
133   return this.originalDeviceAdminFacade.getGatewayList(organisation);
    }
135
    @Override
137 public boolean isDeviceAuthorized(String address)
        throws TechnicalFaultException {
139   return this.originalDeviceAdminFacade.isDeviceAuthorized(address);
    }
141
    @Override
143 public com.francetelecom.citypulse.facade.device.model.Device updateDevice(
        com.francetelecom.citypulse.facade.device.model.Device device)
145     throws TechnicalFaultException, DeviceUpdateException {
      defineObsProperty("deviceID", device.getId());
147   getObsProperty("deviceID").updateValues(device,device.getDeviceType(),device.getName
          (),device.getStatus());
      return this.originalDeviceAdminFacade.updateDevice(device);
149 }

151 }
```

## A.3 *MessagesArt*

Listing A.3 – MessagesArt.java

```
    package com.francetelecom.citypulse.mas4m2m.artifacts;
2
```

```
import java.util.Date;
import java.util.List;
import org.apache.log4j.Logger;

import com.francetelecom.citypulse.interfaces.model.message.MessagePriority;
import com.francetelecom.citypulse.store.model.Command;
import com.francetelecom.citypulse.store.model.Event;
import com.francetelecom.citypulse.store.model.Message;
import com.francetelecom.citypulse.store.model.Payload;
import com.francetelecom.citypulse.store.model.Payload.MessageType;
import cartago.Artifact;
import cartago.LINK;
import cartago.OPERATION;
import cartago.ObsProperty;
import cartago.OpFeedbackParam;

public class MessagesArt extends Artifact {
  protected static Logger logger = Logger.getLogger(MessagesArt.class);
  private List<Message> messages;
  /**
   * The initial operation for the artifact where we have zero message at the begining
   */
  @OPERATION public void init(){
    defineObsProperty("number of messages",0);
  }     //end of nit

  /**
   *
   * observable message strucuture :
   * (id,senderAddress,senderAddressType,...)
   *
   * @param message
   */
  @LINK void addMessages(Message message) {
    defineObsProperty(message.getId(), message.getSenderAddress(),
                  message.getSenderAddressType(), message.getReceiverAddress(),
                  message.getSequenceNumber(),message.getDeviceId(),message.
                      getDeviceTypeId(),
                  message.getAppPayload(),message.getDate(),
                  message.getReceiverAddressType(), message.getPriority().value().
                      toString()
                  , message);
    this.messages.add(message);
    signal("newMessage", message.getId());
    logger.info(message.getId());
  }     // end of addMessage

  /**
   * To remove a given Message
   * @param message
   */

  @OPERATION void removeMessage(Message message) {
    this.messages.remove(message);
    removeObsProperty(message.getId());
    logger.info(message);
  }     //end of removeMessage

  /**
   * Modify a message only by modifying some parameters.<br/>
   * No sense to modify the senderAddress/we can't:
   * <ul>
   *  <li>senderAddress: needed for msg integrity</li>
   *  <li>Date: To specify the sending date</li>
   * </ul>
   * @param messageID
   * @param receiverAddress
   * @param applPayload
   * @param receiverAddressType
   * @param priority
```

```
      */
72  @OPERATION void modifyMessage(int messageID, /*String senderAddress,String
        senderAddressType*/
                    String receiverAddress,/*int sequenceNumber int deviceId,
74                  int deviceTypeId*/ String applPayload,/*Date date*/
                    String receiverAddressType,String priority){
76    Message message = messages.get(messageID);    // get the message having as ID "
          messageID"
      logger.info(message);
78    message.setReceiverAddress(receiverAddress);  // change the receiver address for the
          new receiver address
      message.setReceiverAddress(receiverAddress);
80    message.setAppPayload(applPayload.getBytes());
      //defineObsProperty("receiver", message.getReceiverAddress());
82    //changeReceiver(receiverAddressType,message);
      message.setPriority(MessagePriority.valueOf(priority));
84    message.setReceiverAddressType(getReceiverAddressType(message));
      getObsProperty("messageID").updateValues(messageID,receiverAddress,applPayload,
          receiverAddressType,priority); //update obsProperty
86    logger.info(message);
      // TODO : any action to SensCity ?
88    // (can be "No")
      //defineObsProperty("priority", message.getPriority().value());
90    //changePriority(priority, message);
    }                           // end of modifyMessage
92

94  /**
     * duplicate a message by passing the ID of the message we want to duplicate
96   */
    @OPERATION void duplicateMessage(int messageID, OpFeedbackParam<String> newMsgID){
98    Message m = messages.get(messageID);
      Message newMessage;
100   logger.info(m);
      if(m instanceof Command)
102     newMessage = new Command(new Date(), m.getId(), m.getStatus(), m.getPriority(),
              //set the id of the old message then change it later on (below)
                    m.getSenderAddressType(), m.getSenderAddress(),
104                 m.getReceiverAddressType(), m.getReceiverAddress(),
                    (byte) m.getSequenceNumber(), m.getDeviceId(), m.getDeviceTypeId(),
106                 m.getPayloads());
      else
108     newMessage = new Event(new Date(), m.getId(), m.getStatus(), m.getPriority(),
                    m.getSenderAddressType(), m.getSenderAddress(),
110                 m.getReceiverAddressType(), m.getReceiverAddress(),
                    (byte) m.getSequenceNumber(), m.getDeviceId(), m.getDeviceTypeId(),
112                 m.getPayloads());
      this.messages.add(newMessage);                          // get the message
          having the ID "messageID" and add it to the list
114   int lastID= this.messages.lastIndexOf(messages);                //get the last
          ID of the list => ID of added message
      newMsgID.set(String.valueOf(lastID));                     //set the newMsgID
          by the id of the duplicated message
116   defineObsProperty("lastID",newMessage.getId(), newMessage.getSenderAddress(),
                newMessage.getSenderAddressType(),newMessage.getReceiverAddress(),
118             newMessage.getSequenceNumber(),newMessage.getDeviceId(),newMessage.
                    getDeviceTypeId(),
                newMessage.getAppPayload(),newMessage.getDate(),
120             newMessage.getReceiverAddressType(), newMessage.getPriority().value().
                    toString()
                ,newMessage);
122   logger.info(newMessage);
    }                                         //end of duplicateMessage
124
    @OPERATION void storeMessage(String messageID){
126       // Store in Message DB -> comment ds SensCity ?
    }     //end of storeMessage
128
    /**
130  * get the Type of ReceiverAddress
```

```
     * @param message
132  * @return
     */
134 private int getReceiverAddressType(Message message){
      logger.info(message.getReceiverAddressType());
136   if(message.getReceiverAddressType()==Message.ADR_TYPE_INTERNAL)
        return message.ADR_TYPE_INTERNAL;
138   else
        return 0;      //we have to put zero since other type is not defined in the class
             message
140 }             //end of getReceiverAddressType

142 /**
     * get the Type of SenderAddress
144  */
    private String getSenderAddressType(Message message){
146   logger.info(message.getSenderAddressType());
      if(message.getSenderAddressType() == Message.ADR_TYPE_INTERNAL)
148     return "INTERNAL";
      else
150     return "EXTERNAL";
    }     //end if getSenderAddressType
152
    /**
154  *  to know the type of the message
     * @param message
156  * @return
     */
158
    private String getMessageType(Message message){
160   MessageType type = Payload.MessageType.valueOf(message.getClass());
      logger.info(type);
162   if(type == MessageType.Command)
        return "COMMAND";
164   else if(type == MessageType.Event)
        return "EVENT";
166   else
        return "DEFAULT";
168 }     //end of messageType

170 /**
     * change the receiver address , called by modify method
172  * @param newReceiverAddress
     * @param message
174  */

176 /**
     * get the status of the message,we make it stand alone in case we r going 2 use it
         again many times
178  */
    private String getStatus(Message message){
180   logger.info(message.getStatus());
      try{
182     ObsProperty status = getObsProperty("status");
        if(status == null) {
184       //defineObsProperty("status",this.message.getsStatus());
          defineObsProperty("status");
186       status = getObsProperty("status");
        }
188     switch (message.getStatus()) {
          case Message.STATUS_CANCELLED:
190         status.updateValue("STATUS_CANCELLED");
            return status.toString();
192       case Message.STATUS_CREATED:
            status.updateValue("STATUS_CREATED");
194         return status.toString();
          case Message.STATUS_SENT:
196         status.updateValue("STATUS_SENT");
            return status.toString();
198       case Message.STATUS_RECEIVED:
```

```
          status.updateValue("STATUS_RECEIVED");
200       return status.toString();
        case Message.STATUS_FAILED:
202       status.updateValue("STATUS_FAILED");
          return status.toString();
204     default:
          break;
206     }     //end of switch
      } catch(Exception e) {
208     System.out.println(e);
      }       //end of Try
210   return null;
    }         //end of the updateObsStatus()
212
}//end of the MessageArt
```

## A.4 *UCCPClientArt*

Listing A.4 – UCCPClientArt.java

```
1  package com.francetelecom.citypulse.mas4m2m.artifacts;

3  import java.io.File;
   import java.util.Date;
5  import java.util.List;
   import java.util.Set;
7
   import javax.jws.WebMethod;
9  import javax.jws.WebParam;
   import javax.jws.WebResult;
11 import javax.jws.WebService;
   import javax.jws.soap.SOAPBinding;
13
   import com.francetelecom.citypulse.application.messagedispatcher.IMessageDispatcher;
15 import com.francetelecom.citypulse.interfaces.internal.client.IWsClient;
   import com.francetelecom.citypulse.interfaces.internal.client.uccp.IUccpClient;
17 import com.francetelecom.citypulse.interfaces.internal.exception.business.
       AccessRightFaultException;
   import com.francetelecom.citypulse.interfaces.internal.exception.business.
       LoginFaultException;
19 import com.francetelecom.citypulse.interfaces.internal.exception.business.
       ResourceNotFoundFaultException;
   import com.francetelecom.citypulse.interfaces.internal.exception.technical.
       TechnicalFaultException;
21 import com.francetelecom.citypulse.interfaces.internal.services.uccp.
       IApplicationServices;
   import com.francetelecom.citypulse.interfaces.internal.services.uccp.IMessageServices;
23 import com.francetelecom.citypulse.interfaces.internal.services.uccp.ISessionServices;
   import com.francetelecom.citypulse.interfaces.model.message.Event;
25 import com.francetelecom.citypulse.interfaces.model.message.Notification;
   import com.francetelecom.citypulse.store.IStoreManager;
27 import com.francetelecom.citypulse.store.StoreFactory;
   import com.francetelecom.citypulse.store.model.Message;
29 import com.francetelecom.citypulse.store.model.Payload;

31
   import cartago.Artifact;
33 import cartago.OPERATION;
   @WebService(targetNamespace = "urn:cityheart/uccp", name = "MessageServices")
35 @SOAPBinding(style = SOAPBinding.Style.DOCUMENT)
   public class UCCPClientArt extends Artifact implements IUccpClient,IMessageDispatcher {
37   private IWsClient myIWsClient;
     private Payload payload;
39   private Message myMessage;
     private MessagesArt myMessageArt;
41   private IUccpClient originalUccpClient;
     private IMessageDispatcher originalMessageDispatcher;
```

```
43   @OPERATION public void init(IUccpClient originalUccpClient,IMessageDispatcher
         originalMessageDispatcher){
       this.originalUccpClient = originalUccpClient;
45       this.originalMessageDispatcher = originalMessageDispatcher;
       myIWsClient.addRequestLogging();
47     myIWsClient.addResponseLogging();
     }
49
     @OPERATION public void sendMessage(Message myMessage, String gateway){// i focused
         here on the content nd not by determining every property:will b added later on
51     this.myMessage = myMessage;
       payload.setMessage(myMessage);
53     myMessage.setReceiverAddress(gateway);
       myMessageArt.init();
55   }

57   @OPERATION public void receivedMsg(MessagesArt myMessageArt){
       signal("Received Message from", getObsProperty("sender"));
59     signal("Message received: ",getObsProperty("messageContent"));
     }
61   @WebResult(name = "response", targetNamespace = "urn:cityheart/uccp")
     @WebMethod
63   @OPERATION public List<Event> retrieveEventListFromDevice(
         @WebParam(name = "macId", targetNamespace = "urn:cityheart/uccp")
65       String macId,
         @WebParam(name = "fromDate", targetNamespace = "urn:cityheart/uccp")
67       Date fromDate,
         @WebParam(name = "toDate", targetNamespace = "urn:cityheart/uccp")
69       Date toDate,
         @WebParam(name = "token", targetNamespace = "urn:cityheart/uccp")
71       String token) throws TechnicalFaultException, LoginFaultException,
         AccessRightFaultException, ResourceNotFoundFaultException{
73         return null;
         }
75   @WebResult(name = "response", targetNamespace = "urn:cityheart/uccp")
     @WebMethod
77   public List<Event> retrieveEventListFromDeviceType(
         @WebParam(name = "deviceTypeId", targetNamespace = "urn:cityheart/uccp")
79       Integer deviceTypeId,
         @WebParam(name = "organisationId", targetNamespace = "urn:cityheart/uccp")
81       Integer organisationId,
         @WebParam(name = "fromDate", targetNamespace = "urn:cityheart/uccp")
83       Date fromDate,
         @WebParam(name = "toDate", targetNamespace = "urn:cityheart/uccp")
85       Date toDate,
         @WebParam(name = "token", targetNamespace = "urn:cityheart/uccp")
87       String token) throws TechnicalFaultException, LoginFaultException,
         AccessRightFaultException, ResourceNotFoundFaultException{
89         return null;
         }
91
     @Override
93   public IApplicationServices getApplicationServices() {
       return this.originalUccpClient.getApplicationServices();
95   }

97   @Override
     public IMessageServices getMessageServices() {
99     return this.originalUccpClient.getMessageServices();
     }
101
     @Override
103  public ISessionServices getSessionServices() {
       return this.originalUccpClient.getSessionServices();
105  }

107  @Override
     public void addRequestLogging() {
109    this.originalUccpClient.addRequestLogging();
     }
```

```
111
      @Override
113   public void addResponseLogging() {
        this.originalUccpClient.addResponseLogging();
115   }

117   @Override
      public void enableSsl(File keystore, String keyStorePassword)
119       throws TechnicalFaultException {
        this.originalUccpClient.enableSsl(keystore, keyStorePassword);
121   }

123   @Override
      public void enableSsl(String keystore, String keyStorePassword)
125       throws TechnicalFaultException {
        this.originalUccpClient.enableSsl(keystore, keyStorePassword);
127   }

129   @Override
      public void setAddress(String address) {
131     this.originalUccpClient.setAddress(address);
      }
133
      @Override
135   public void setProxy(String proxyServer, int proxyPort, boolean socks) {
        this.originalUccpClient.setProxy(proxyServer, proxyPort, socks);
137   }

139   @Override
      public void setTimeout(long serviceLength, long connectionDelay) {
141     this.setTimeout(serviceLength, connectionDelay);
      }
143
      @Override
145   public void recieveMessage(Notification notification) {
        this.originalMessageDispatcher.recieveMessage(notification);
147     signal("messageReceived", notification);
      }
149
      @Override
151   public void recieveUrgentEvent(Event event, Integer deviceTypeId,
          Set<Integer> organizationIds) throws TechnicalFaultException {
153     this.originalMessageDispatcher.recieveUrgentEvent(event, deviceTypeId,
            organizationIds);
      signal("urgentEventReceived", event,deviceTypeId);
155   }

157   @Override
      public void start() {
159     this.originalMessageDispatcher.start();
      }
161
      @Override
163   public void stop() {
        this.originalMessageDispatcher.stop();
165   }

167
    }
```

## A.5 *UserAuthArt*

Listing A.5 – UserAuthArt.java

```
1 package com.francetelecom.citypulse.mas4m2m.artifacts;

3
```

```
   import java.util.Map;
 5
   import com.francetelecom.citypulse.application.configuration.IApplicationConfigurator;
 7 import com.francetelecom.citypulse.application.configuration.model.
       ApplicationConfiguration;
   import com.francetelecom.citypulse.application.configuration.model.Subscription;
 9 import com.francetelecom.citypulse.application.session.ISessionManager;
   import com.francetelecom.citypulse.application.session.SessionManager;
11 import com.francetelecom.citypulse.interfaces.internal.exception.business.
       LoginFaultException;
   import com.francetelecom.citypulse.interfaces.internal.exception.business.
       ResourceNotFoundFaultException;
13 import com.francetelecom.citypulse.interfaces.internal.exception.technical.
       TechnicalFaultException;
   import com.francetelecom.citypulse.session.Session;
15

17 import cartago.Artifact;
   import cartago.OPERATION;
19
   /**
21  *
    * @author Mustapha Bilal
23  *
    */
25 public class UserAuthArt extends Artifact
               implements ISessionManager{
27   private TmpSession mySession;
     private Map<String, Session> sessionMap;
29   private Session adminSession;
     private ISessionManager originalSessionManager;
31   private SessionManager SessionManager;
     private Subscription mySubscription;
33   private IApplicationConfigurator myIApplicationConfigurator;
     private ApplicationConfiguration myApplicationConfiguration;
35
     public void init(ISessionManager originalSessionManager) {
37     this.originalSessionManager = originalSessionManager;

39   }

41   @OPERATION public void initOld(TmpSession mySession){
       this.mySession = mySession;
43       defineObsProperty("sessionID", mySession.getSessionId());
         defineObsProperty("applicationId", mySession.getApplicationId());
45       defineObsProperty("applicationKey", mySession.getApplicationKey());
         defineObsProperty("uccpLogin", mySession.getUccpLogin());
47       defineObsProperty("uccpPassword", mySession.getUccpPassword());
         defineObsProperty("token", mySession.getToken());
49       defineObsProperty("mainApplication", mySession.isMainApplication());
         defineObsProperty("lastActionDate", mySession.getLastActionDate());
51       defineObsProperty("lastTokenVerification", mySession.getLastTokenVerification());

53     }
     @OPERATION public boolean login(SessionManager mySessionManager){
55     this.mySessionManager = mySessionManager;
       try {
57       mySessionManager.login(getObsProperty("applicationId").stringValue(),
             getObsProperty("applicationKey").stringValue());
       } catch (LoginFaultException e) {
59       e.printStackTrace();
           } catch (TechnicalFaultException e) {
61       e.printStackTrace();
           }
63       try {
         mySessionManager.checkSession(getObsProperty("sessionId").stringValue());
65     } catch (LoginFaultException e) {
         e.printStackTrace();
67       }
       try {
```

```
69        mySessionManager.getAdminToken();
      } catch (LoginFaultException e) {
71        e.printStackTrace();
        } catch (TechnicalFaultException e) {
73        e.printStackTrace();
          }
75      return true;
    }
77  @OPERATION public void logout(){
      mySessionManager.logout(getObsProperty("sessionId").stringValue());
79  }

81  /**
     *
83   * @param mySubscription
     * @param myIApplicationConfigurator
85   * @param myApplicationConfiguration
     */
87  public void subscribe(Subscription mySubscription,IApplicationConfigurator
        myIApplicationConfigurator,
        ApplicationConfiguration myApplicationConfiguration){
89      this.myApplicationConfiguration=myApplicationConfiguration;
      this.mySubscription=mySubscription;
91      this.myIApplicationConfigurator = myIApplicationConfigurator;
      defineObsProperty("subscriptionId", mySubscription.getSubscriptionId());
93      defineObsProperty("deviceGroup", mySubscription.getDeviceGroup().getDeviceTypeId());
      defineObsProperty("applicationID", mySubscription.getApplication().getApplicationId
          ());
95      defineObsProperty("notificationURL", mySubscription.getApplication().
          getNotificationUrl());
      defineObsProperty("uccpAccess", mySubscription.getApplication().getUccpAccess());
97      defineObsProperty("description", mySubscription.getApplication().getDescription());
      defineObsProperty("lastReadEvents", mySubscription.getLastReadEvents());
99      defineObsProperty("eventFilter", mySubscription.getEventFilter());
        try {
101       myIApplicationConfigurator.registerApplication(myApplicationConfiguration);
      } catch (TechnicalFaultException e) {
103       e.printStackTrace();
      }
105     try {
        myIApplicationConfigurator.addSubscription(myApplicationConfiguration,
            mySubscription.getDeviceGroup(), mySubscription.getEventFilter());
107     } catch (TechnicalFaultException e) {
        e.printStackTrace();
109     }
    }
111 @OPERATION public void updateSubscription(Subscription mySubscription){
      updateSubscription(mySubscription);
113 }
    @OPERATION public void removeSubscription(Subscription mySubscription){
115     try {
        myIApplicationConfigurator.removeSubscription(getObsProperty("applicationId").
            stringValue(),mySubscription.getSubscriptionId());
117     } catch (ResourceNotFoundFaultException e) {
        e.printStackTrace();
119     } catch (TechnicalFaultException e) {
        e.printStackTrace();
121     }
    }
123 @OPERATION public void lockSubscription(Subscription mySubscription){
      lockSubscription(mySubscription);
125 }
    @OPERATION public void unlockSubscription(Subscription mySubscription){
127     try {
        myIApplicationConfigurator.unlockSubscription(mySubscription.getSubscriptionId());
129     } catch (TechnicalFaultException e) {
        e.printStackTrace();
131     }
    }
```

```
133  @OPERATION public void addPayloadTranslator(ApplicationConfiguration
         myApplicationConfiguration,Subscription mySubscription,
         String myTranslatorClassName){
135    try {
         myIApplicationConfigurator.addPayloadTranslator(myApplicationConfiguration,
137          mySubscription.getDeviceGroup().getDeviceTypeId(),myTranslatorClassName);
       } catch (TechnicalFaultException e) {
139      e.printStackTrace();
       }
141  }
     @OPERATION public void removePayloadTranslator(ApplicationConfiguration
         myApplicationConfiguration,Subscription mySubscription,
143      String myTranslatorClassName){
       try {
145      myIApplicationConfigurator.removePayloadTranslator(mySubscription.getApplication()
             .getApplicationId(), mySubscription.getDeviceGroup().getDeviceTypeId());
       } catch (ResourceNotFoundFaultException e) {
147      // TODO Auto-generated catch block
         e.printStackTrace();
149    } catch (TechnicalFaultException e) {
         e.printStackTrace();
151    }
     }
153  @Override
     public void checkSession(String sessionId) throws LoginFaultException {
155    this.originalSessionManager.checkSession(sessionId);
     }
157  @Override
     public String getAdminToken() throws TechnicalFaultException,
159      LoginFaultException {
       return this.originalSessionManager.getAdminToken();
161  }
     @Override
163  public String getLogin(String sessionId) throws LoginFaultException {
       return this.originalSessionManager.getLogin(sessionId);
165  }
     @Override
167  public String getToken(String sessionId) throws TechnicalFaultException,
         LoginFaultException {
169    return originalSessionManager.getToken(sessionId);
     }
171
     @Override
173  public boolean isAdminSession(String sessionId) throws LoginFaultException {
       return this.originalSessionManager.isAdminSession(sessionId);
175  }

177  @Override
     public String login(String applicationId, String applicationKey)
179      throws LoginFaultException, TechnicalFaultException {
       return this.originalSessionManager.login(applicationId, applicationKey);
181  }

183
     @Override
185  @OPERATION public void logout(String sessionId) {
       this.originalSessionManager.logout(sessionId);
187  }
   }
```

## A.6 *USPCNXArt*

Listing A.6 – USPCNXArt.java

```
   package com.francetelecom.citypulse.mas4m2m.artifacts;
2
```

```
 4  import java.util.Set;

 6  import javax.jws.WebService;
    import javax.jws.soap.SOAPBinding;
 8

10  import com.francetelecom.citypulse.facade.device.wdc.IWdcManager;
    import com.francetelecom.citypulse.facade.device.wdc.data.Wdc;
12  import com.francetelecom.citypulse.facade.device.wdc.data.WdcManagedDevice;
    import com.francetelecom.citypulse.interfaces.external.exception.business.
        MisssingRequiredParameter;
14  import com.francetelecom.citypulse.interfaces.external.model.DeviceData;
    import com.francetelecom.citypulse.interfaces.external.services.INotificationServices;
16  import com.francetelecom.citypulse.interfaces.internal.exception.business.
        ResourceNotFoundFaultException;
    import com.francetelecom.citypulse.interfaces.internal.exception.technical.
        TechnicalFaultException;
18

20
    import cartago.Artifact;
22  import cartago.OPERATION;
    @WebService(targetNamespace = "urn:cityheart/usp", name = "NotificationServices")
24  @SOAPBinding(style = SOAPBinding.Style.DOCUMENT)
    public class USPCNXArt extends Artifact implements INotificationServices {
26    private TempNotification myNotification;
      private MessagesArt myMessageArt;
28    private INotificationServices myINotificationService;
      private IWdcManager myIWdcmanager;
30    private DeviceArt myDevice;
      private WdcManagedDevice myWdcManagedDevice;
32    private INotificationServices originalINotificationServices;
      @OPERATION public void init(INotificationServices originalINotificationServices){
34      this.originalINotificationServices = originalINotificationServices;
      }
36
      @OPERATION public void notifyUSP(MessagesArt myMessageArt){
38      this.myMessageArt = myMessageArt;
        myNotification.setDeviceTypeId(getObsProperty("deviceTypeId").intValue());// peut
             tre  a  sera mieux de garder l'oganisation et on le set  1
40      myNotification.toString();
        myINotificationService.ping();
42    }

44    @OPERATION public void deliverMessage(MessagesArt myMessageArt){
        TmpMessage message = new TmpMessage();
46      message.setSequenceNumber((byte) getObsProperty("sequenceNumber").intValue());
        message.setReceiverAddressType(getObsProperty("receiverAddrType").intValue());
48      message.setSenderAddressType(getObsProperty("senderAddrType").intValue());
        message.setReceiverAddress(getObsProperty("receiver").stringValue());
50      message.setSenderAddress(getObsProperty("sender").stringValue());
        message.setAppPayload(getObsProperty("messageContent").stringValue().getBytes());
52    }
      @OPERATION public void deviceUnsubscribe(IWdcManager myIWdcmanager,DeviceArt myDevice)
          {
54      this.myIWdcmanager=myIWdcmanager;
        this.myDevice = myDevice;
56       try {
          if (myIWdcmanager.isDeviceManagedByWdc(getObsProperty("deviceName").stringValue())
              ==null)
58          signal("Managed by wdc", getObsProperty("deviceName"));
         } catch (TechnicalFaultException e) {
60        e.printStackTrace();
          }
62      myWdcManagedDevice.setName(getObsProperty("deviceName").stringValue());
        try {
64        myIWdcmanager.removeWdcManagedDevice(myWdcManagedDevice);
        } catch (TechnicalFaultException e) {
66        // TODO Auto-generated catch block
          e.printStackTrace();
```

```
68      }
    }
70  @OPERATION public void deviceSubscribe(IWdcManager myIWdcmanager,DeviceArt myDevice,
        Wdc myWdc){
      this.myIWdcmanager=myIWdcmanager;
72    this.myDevice = myDevice;
       try {
74      if (myIWdcmanager.isDeviceManagedByWdc(getObsProperty("deviceName").stringValue())
            ==null)
          signal("Not Managed by wdc yet", getObsProperty("deviceName"));
76     } catch (TechnicalFaultException e) {
        e.printStackTrace();
78       }
      try {
80      myIWdcmanager.createWdc(myWdc);
      } catch (TechnicalFaultException e) {
82      e.printStackTrace();
        }
84    myWdcManagedDevice.setName(getObsProperty("deviceName").stringValue());//must we
          dtermin the other properties?
    }
86
    @Override
88  public void newUrgentData(DeviceData data) throws TechnicalFaultException,
        ResourceNotFoundFaultException, MisssingRequiredParameter {
90    this.originalINotificationServices.newUrgentData(data);
    }
92
    @Override
94  public void ping() {
      this.originalINotificationServices.ping();
96  }

98 }
```

# Annexe B

# M2MPlatformWorkspace

Listing B.1 – M2MPlatformWorkspace.java

```
package com.francetelecom.citypulse.mas4m2m.artifacts.workspace;

import java.util.Map;

import cartago.*;
import cartago.security.AgentIdCredential;
import cartago.util.BasicLogger;
import cartago.util.agent.Agent;

/**
 * This class is a singleton workspace in the SensCity platform for CArtAgO
 * artifacts. This workspace is accessible for distant workspace and/or agents
 */
public class M2MPlatformWorkspace {

  private ICartagoSession session;

  private String wspName;

  private static M2MPlatformWorkspace instance;

  private M2MPlatformWorkspace() {

    // TODO fetch the platform url
    // TODO type of the platform
    this.wspName = "M2MPlatformWorkspace:"/*plfType+":"+pltfURL*/;
    try {
      CartagoService.startNode(new BasicLogger());
      /* TODO : quel protocol?
       * - RMI (default)
       * - lipermi (??)
       */
      CartagoService.installInfrastructureLayer("default");
      // TODO : quel protocol?
      CartagoService.startInfrastructureService("default");
      CartagoService.registerLogger("default", new BasicLogger());

    } catch (CartagoException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }

  /**
   * initializes the workspace
   * @return
   */
  public static M2MPlatformWorkspace getInstance() {
    if (instance == null)
```

```
          instance = new M2MPlatformWorkspace();
51      return instance;
    }
53
    /**
55   * Get the name of the current workspace
     * @return the workspace name
57   */
    public String getWspName() {
59      return this.wspName;
    }
61 }
```