RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE *ET*

POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE

LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ 20 Août 1955 – Skikda

Faculté des Sciences Département d'INFORMATIQUE

Mémoire présenté en vue de l'obtention d'un Diplôme

Master 2

Spécialité : Génie Logiciel Avancé Et Application

THÉME :

Modélisation et Simulation d'un Parking Utilisant le Paradigme des Réseaux dans les Réseaux

Réalisé par :

- **Djilani nour eddine**
- **Walid Mezghiche**

Encadré par:

- **Dr Kissoum Yacine**

Année universitaire : 2024/2025

Remerciements

Avant tout, nous tenons, nous ***les étudiants signataires de ce mémoire,*** *D ,Nour* ***et*** *M ,Walid, à exprimer notre profonde gratitude à Dieu Tout-Puissant, pour nous avoir accordé la force, la persévérance et la santé nécessaires afin de mener à bien ce parcours académique. C'est grâce à Sa miséricorde que nous avons pu franchir les différentes étapes de notre formation, surmonter les difficultés et atteindre ce niveau d'étude. Dieu merci, pour tout.*

Nous adressons nos remerciements les plus sincères à nos familles respectives, en particulier à nos parents, pour leur amour inconditionnel, leur soutien moral constant et leurs prières précieuses. Ils ont été nos repères dans les moments de doute, nos premières sources de motivation et les piliers de notre réussite. Rien n'aurait été possible sans leurs sacrifices quotidiens, leur patience et leur encouragement silencieux. Que Dieu les garde et les protège.

Nous exprimons également notre profonde reconnaissance à ***notre encadreur,*** *Dr. KISSOUM YACINE, pour son accompagnement attentif, sa disponibilité, sa patience, ainsi que pour ses conseils avisés et ses orientations précieuses qui ont enrichi la qualité de notre travail. Sa confiance nous a énormément encouragés et son sens du professionnalisme a été pour nous un véritable modèle.*

Nous remercions chaleureusement l'ensemble des enseignants du département d'informatique, pour la qualité de l'enseignement dispensé, pour leur engagement et pour les connaissances précieuses transmises tout au long de notre parcours universitaire. Ils ont su éveiller en nous la curiosité scientifique et le sens de la rigueur.

Nos pensées vont également à nos camarades de promotion, avec qui nous avons partagé cette aventure universitaire : les moments de fatigue, de stress, mais aussi de joie et de réussite collective. Merci pour l'esprit d'entraide et les souvenirs inoubliables.

Enfin, nous remercions toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce mémoire, que ce soit par un conseil, un mot d'encouragement ou un geste de soutien.

À toutes et à tous, recevez l'expression de notre profonde reconnaissance.

Dédicace

Avant toute chose, je rends grâce à DIEU, source de force et de persévérance, qui m'a permis

d'accomplir ce modeste travail.

Je dédie ce travail du fond du cœur

À ma mère et à mon père, piliers de ma vie,

À mes sœurs, pour leur amour et leur soutien inconditionnels,

À mes amis les plus chers, compagnons fidèles de chaque étape,

Et à toutes les personnes qui m'aiment et que j'aime,

car chacune d'elles a contribué, à sa manière, à ce parcours.

Djilani Nour eddine

Dédicace

Avec une profonde gratitude, je rends grâce à DIEU, pour m'avoir guidé, soutenu et accordé la

force nécessaire tout au long de ce parcours.

Je dédie ce travail avec amour et reconnaissance

À ma chère mère et à mon père, pour leurs sacrifices et leur foi en moi,

À mes sœurs, pour leur tendresse et leur présence rassurante,

À mes amis les plus proches, pour leur soutien moral et leur encouragement,

Et à tous ceux qui m'aiment et que j'aime,

car sans eux, ce chemin n'aurait pas eu la même saveur.

Walid Mezghiche

Resumé :

Dans un contexte marqué par une urbanisation croissante et une augmentation significative du nombre de véhicules, les villes modernes sont de plus en plus confrontées à des problèmes de stationnement, tant en termes de disponibilité que de gestion efficace des espaces. Ce phénomène engendre une perte de temps considérable pour les conducteurs, une augmentation du trafic urbain, ainsi qu'un impact négatif sur l'environnement.

Face à ces défis, les **systèmes intelligents de parking** s'imposent comme une solution technologique prometteuse. En intégrant des outils innovants tels que les capteurs, l'Internet des objets (IoT), l'intelligence artificielle et les applications mobiles, ces systèmes permettent une gestion optimisée et en temps réel des places de stationnement.

Ce travail de recherche s'inscrit dans cette problématique actuelle, et vise à concevoir un **système intelligent de parking-auto** capable de faciliter la recherche de stationnement, de réduire les congestions routières et d'améliorer l'expérience utilisateur. À travers cette étude, nous explorons les différentes technologies mises en œuvre dans ce type de systèmes, ainsi que les bénéfices qu'elles peuvent apporter aux usagers et aux collectivités.

Mots clés :

Système intelligent, stationnement, gestion de parking, Internet des objets (IoT), intelligence artificielle, mobilité urbaine, optimisation, application mobile, réduction du trafic, automatisation, technologies émergentes, flux de circulation, efficacité énergétique.

Abstract :

With the rapid growth of urban areas and the continuous increase in the number of vehicles, modern cities are facing serious challenges related to parking management, including the lack of available spaces and inefficient parking processes. These issues lead to wasted time for drivers, increased traffic congestion, and negative environmental impacts.

In response to these problems, **intelligent parking systems** have emerged as promising technological solutions. By integrating modern technologies such as sensors, the Internet of Things (IoT), artificial intelligence, and mobile applications, these systems enable real-time and optimized management of parking spaces.

This graduation project aims to design and implement an **intelligent car parking system** that facilitates the search for available spaces, reduces road congestion, and improves the overall user experience. Throughout this work, we explore the key technologies used in smart parking systems and the advantages they offer to both users and urban communities.

Keywords :

Smart system, parking, parking management, Internet of Things (IoT), artificial intelligence (AI), sensors, urban mobility, optimization, mobile application, traffic reduction, emerging technologies, traffic flow, energy efficiency.

Sommaire :

Liste de figures

Introduction Générale

Introduction Générale

L'aube du XXIe siècle est indéniablement marquée par une accélération sans précédent des avancées technologiques, redéfinissant les frontières entre le monde physique et l'univers numérique. Au cœur de cette révolution se trouve l'Internet des Objets (IoT), un concept visionnaire qui a transcendé sa simple définition technique pour devenir un véritable moteur de transformation sociétale et économique. En dotant d'innombrables objets du quotidien – des capteurs industriels aux dispositifs médicaux portables, en passant par les infrastructures urbaines – de capacités de connectivité, de détection et de traitement de données, l'IoT tisse une toile d'intelligence omniprésente, promettant une optimisation sans précédent des ressources, une amélioration significative de la qualité de vie et l'émergence de nouveaux modèles d'affaires. Ce paradigme d'interconnexion n'est plus une simple perspective d'avenir, mais une réalité tangible qui façonne déjà des domaines aussi divers que l'agriculture de précision, la santé connectée, l'industrie 4.0, les réseaux énergétiques intelligents et, de manière cruciale pour notre étude, les villes intelligentes.

Dans ce contexte de numérisation croissante, les zones urbaines font face à des défis complexes et multidimensionnels, exacerbés par une urbanisation rapide et une démographie galopante. Parmi ces défis, la gestion du stationnement se dresse comme une problématique quotidienne majeure, ayant des répercussions directes sur la qualité de vie des citadins, l'efficacité des transports et la durabilité environnementale. La recherche fastidieuse de places de parking engendre non seulement des pertes de temps considérables pour les conducteurs, mais contribue également de manière significative à la congestion du trafic, à l'augmentation des émissions de gaz à effet de serre et à une utilisation sous-optimale de l'espace urbain. Face à cette réalité, l'intégration de l'IoT et des technologies d'intelligence artificielle dans la gestion du stationnement représente une voie prometteuse pour concevoir des solutions innovantes et durables. Les systèmes de parking intelligents, en exploitant la puissance des données en temps réel et des algorithmes d'optimisation, offrent la capacité de transformer la recherche de stationnement d'une corvée frustrante en une expérience fluide et efficace.

Ce mémoire s'inscrit pleinement dans cette démarche d'innovation en proposant la conception, la modélisation et l'implémentation d'un système de parking intelligent. Notre approche se distingue par l'adoption d'une architecture multi-agents, reconnue pour sa capacité à gérer des systèmes distribués et réactifs, et par l'utilisation des Réseaux de Pétri, un formalisme puissant pour la modélisation et l'analyse de systèmes à événements discrets. L'objectif central de ce travail est de développer une solution capable de gérer dynamiquement l'attribution des places de stationnement, en prenant en compte des critères de priorité pour les véhicules et en offrant une visualisation en temps réel de l'état global du système. Cette combinaison méthodologique vise à garantir non seulement l'efficacité opérationnelle, mais aussi la robustesse et la capacité d'adaptation de notre système face aux dynamiques urbaines fluctuantes.

Afin de présenter une analyse exhaustive de notre travail, le présent mémoire est structuré en cinq chapitres interconnectés, chacun abordant un aspect fondamental du système :

- **Le Chapitre 1 : "Internet des Objets"** établira le cadre technologique de notre étude. Il débutera par une définition approfondie de l'IoT, en détaillant ses couches architecturales, ses composants matériels et logiciels essentiels, et les principaux protocoles de communication qui régissent son fonctionnement. Ce chapitre explorera également les vastes domaines d'application de l'IoT, tout en mettant en lumière les défis cruciaux liés à la sécurité, à la confidentialité des données et à l'interopérabilité des systèmes, soulignant ainsi la complexité du paysage dans lequel notre solution s'inscrit.
- **Le Chapitre 2 : "Systèmes de Parking Intelligents"** se concentrera spécifiquement sur la problématique du stationnement urbain. Il détaillera les enjeux actuels liés à la congestion et à la pollution générées par la recherche de parking, avant de présenter le concept de parking intelligent comme une réponse technologique viable. Nous y analyserons les architectures génériques des systèmes de parking intelligents existants, en examinant leurs fonctionnalités, leurs avantages en termes d'efficacité et d'impact environnemental, ainsi que leurs limites et les défis persistants, ce qui permettra de positionner de manière pertinente notre proposition.
- **Le Chapitre 3 : "Réseaux de Pétri"** introduira un formalisme mathématique et graphique essentiel à notre approche : les Réseaux de Pétri. Ce chapitre expliquera les concepts fondamentaux des Réseaux de Pétri, leur sémantique et leurs capacités de modélisation pour les systèmes à événements discrets, notamment ceux caractérisés par le parallélisme et la concurrence. Nous aborderons également les différentes extensions des Réseaux de Pétri et leurs propriétés d'analyse, telles que la bornitude et la vivacité, qui sont cruciales pour garantir la correction et la performance du système modélisé.
- **Le Chapitre 4 : "Modélisation et Simulation avec Renew"** présentera la mise en œuvre pratique de notre système à l'aide de l'outil Renew (Reference Net Workshop), un simulateur basé sur les Réseaux de Pétri orientés objets. Ce chapitre détaillera l'architecture multi-agents de notre système, où chaque entité (voiture, agent de gestion, parking) est représentée comme un "Net Within Net" actif. Nous décrirons pas à pas le déroulement d'une simulation type, en explicitant les interactions entre les agents, les mécanismes de communication et les processus internes qui régissent l'allocation dynamique des places.
- **Le Chapitre 5 : "Implémentation du Système"** exposera le processus de développement et de concrétisation de notre système de parking intelligent. Ce chapitre justifiera les choix technologiques effectués, notamment l'utilisation du framework JADE (Java Agent DEvelopment Framework) pour la logique multi-agents et de JavaFX pour la conception d'une interface utilisateur graphique intuitive et réactive. Nous y détaillerons les défis techniques rencontrés lors de l'intégration des différents modules, les solutions apportées, ainsi que les résultats des simulations et les observations critiques qui valideront la faisabilité et l'efficacité de notre solution.

En définitive, ce mémoire vise à démontrer la pertinence et l'efficacité d'une approche intégrée combinant l'intelligence des agents autonomes et la rigueur des Réseaux de Pétri pour résoudre des problèmes complexes de gestion urbaine. Nous espérons que les contributions de ce travail ouvriront de nouvelles perspectives pour le développement de systèmes de gestion de stationnement plus adaptatifs, plus efficaces

et plus respectueux de l'environnement, contribuant ainsi à l'avènement de villes véritablement intelligentes et durables.

CHAPITRE 01 :
INTERNET DES OBJETS

1.1 Introduction

Le développement rapide des technologies de l'information et de la communication a ouvert la voie à une nouvelle ère de connectivité et d'interaction entre le monde physique et le monde numérique. Au cœur de cette révolution se trouve **l'Internet des Objets (IoT)**, un concept qui transforme profondément les paradigmes traditionnels de l'informatique en intégrant des objets physiques dans les réseaux numériques afin de collecter, transmettre, analyser et exploiter des données en temps réel.

L'IoT n'est plus simplement une innovation technologique ; il constitue aujourd'hui un levier majeur de transformation dans de nombreux domaines tels que la santé, l'industrie, l'énergie, les villes intelligentes ou encore la domotique. Grâce à l'évolution des protocoles de communication, à la miniaturisation des capteurs et à l'essor de l'intelligence artificielle et du cloud computing, les objets connectés deviennent de plus en plus autonomes, intelligents et interopérables.

Dans ce chapitre, nous nous intéresserons aux fondements de l'Internet des Objets, à son fonctionnement technique, à ses principaux domaines d'application ainsi qu'aux défis qu'il soulève en matière de sécurité, de gestion des données et d'intégration dans les systèmes existants. Ce travail s'inscrit dans une volonté de mieux comprendre l'impact de l'IoT sur notre quotidien et sur les structures économiques et sociales, en analysant à la fois ses opportunités et ses limites.

1.2 Internet des objets : qu'est-ce que c'est ?

L'Internet des Objets (IoT, pour Internet of Things en anglais) est une notion technologique en pleine expansion qui désigne l'ensemble des objets physiques connectés à Internet. Ces objets, aussi variés que des capteurs, des montres intelligentes, des appareils électroménagers, des voitures ou même des équipements industriels, sont capables de collecter des données depuis leur environnement, de les transmettre et parfois de les analyser ou de prendre des décisions en toute autonomie.

Chaque objet est doté d'une identité numérique propre, ce qui le rend unique et reconnaissable dans le réseau global. Cette identité numérique peut être une adresse IP, un identifiant RFID ou toute autre méthode d'identification normalisée. Cela permet à ces objets d'être interconnectés, de communiquer entre eux, et de fonctionner en synergie avec d'autres systèmes d'information.

Le concept fondamental de l'IoT repose sur la possibilité d'interagir avec les objets du monde réel par le biais de réseaux informatiques, créant ainsi une passerelle entre le monde physique et le monde numérique. Par exemple, il devient possible de suivre l'état d'un objet, d'interagir avec lui à distance via une application mobile, ou même de l'automatiser en fonction de règles définies.

Techniquement, l'Internet des Objets repose sur l'utilisation de technologies sans fil telles que le Bluetooth, le Wi-Fi, la radio-identification (RFID) ou les réseaux cellulaires, et fait appel à différents protocoles de communication (comme HTTP, MQTT ou CoAP) afin de

garantir la transmission des données entre les objets connectés et les serveurs voir figure (1.1).

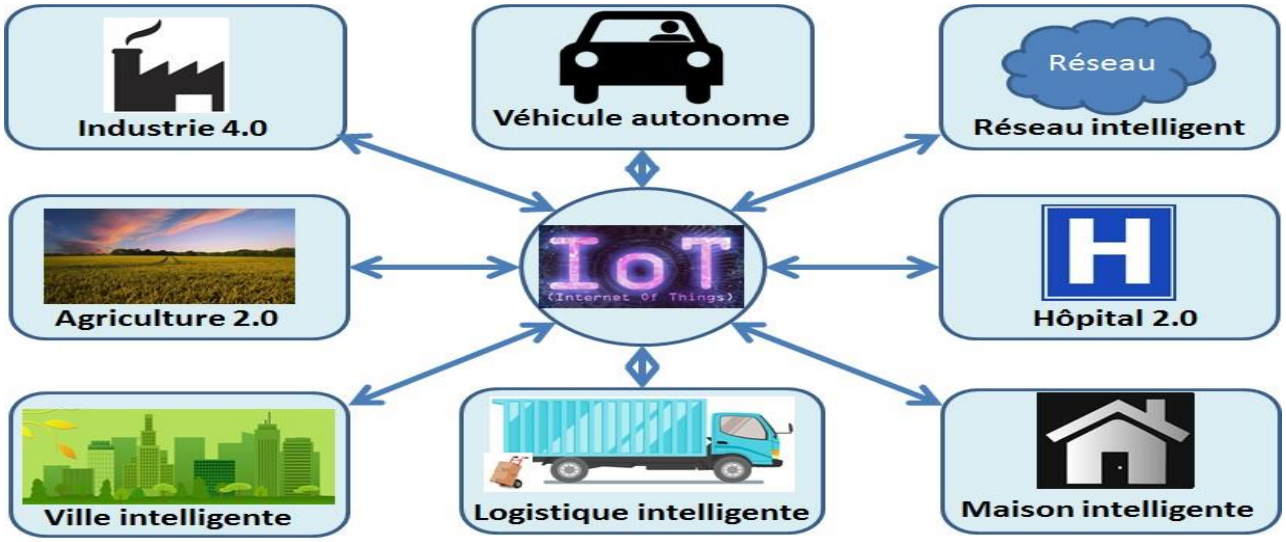

Figure 1. 1 : Internet of Things connecte des objets

1.3Comment fonctionne l'Internet des objets

Le fonctionnement de l'IoT repose sur un principe de base : chaque objet connecté possède une « carte d'identité numérique » qui permet de l'identifier et de le piloter à distance. Cette carte d'identité, souvent représentée par une adresse IP unique, permet de localiser l'objet sur le réseau et de lui envoyer des instructions à partir d'un ordinateur, d'un smartphone ou d'un serveur distant.

Les objets connectés remplissent généralement deux fonctions principales :

1. **La collecte et le stockage d'informations** : Les objets utilisent leurs capteurs pour recueillir des données depuis leur environnement (température, humidité, mouvement, etc.) qu'ils stockent localement ou envoient à un serveur distant.
2. **Le déclenchement d'actions** : En fonction des données collectées et analysées, l'objet peut initier une réponse automatique. Par exemple, un système d'arrosage connecté peut se déclencher automatiquement lorsque la température extérieure dépasse un certain seuil ou que le sol devient trop sec.

Ce processus est généralement décomposé en trois étapes fondamentales (voir figure 1.2) :

- **Things** : les « Things » sont des **éléments physiques disposant de capteurs intégrés** et qui sont connectés à Internet. Ces objets envoient des données de télémétrie (comme, par exemple, lors du passage d'une voiture sur une autoroute à l'aide des capteurs de mouvement ou de pression).

- **Insights** : il s'agit des **résultats du traitement et de l'analyse** des données brutes envoyées par les Things. Par exemple : en calculant le nombre de voitures qui passent par minute, on peut déduire l'état du trafic sur une autoroute.
- **Actions** : il s'agit de la **réponse automatisée ou manuelle** aux informations remontées par l'Insight. Par exemple : si l'Insight montre qu'il y a des embouteillages, des alertes peuvent être envoyées à divers systèmes tels que les tableaux d'affichage des autoroutes ou les différentes applications de navigation utilisées par les automobilistes.

Figure 1. 2 : étapes fondamentales de processus

1.4 Pourquoi faire de l'IoT ?

L'IoT aide les individus à mieux connaître leur environnement et donc à en maîtriser tous les composants. Avec des **objets connectés** dans la maison, on peut optimiser et automatiser plusieurs aspects de la vie de tous les jours. C'est d'ailleurs déjà le cas avec la domotique, qui propose plusieurs technologies et devises pour gérer le chauffage, l'éclairage, etc.

Pour les **entreprises**, l'IoT offre une vision en temps réel sur la totalité de leurs systèmes, fournissant ainsi des informations sur les données qu'il est important de maîtriser : des performances des machines jusqu'aux opérations de la chaîne d'approvisionnement et de la logistique. L'IoT permet alors aux entreprises d'**automatiser les processus** et de **réduire les coûts de main-d'œuvre**.

Il a aussi un impact positif sur la **réduction du gaspillage** et **l'amélioration de la prestation de services**, en rendant notamment la fabrication et la livraison des biens moins coûteuses. Pour avoir une vision plus approfondie de la manière dont les nouveaux outils digitaux, tels que l'IoT ou encore la blockchain, peuvent améliorer la performance des entreprises, découvrez quelques cas d'usages sur la digitalisation du BTP et des usines.

1.5 Les avantages de l'IoT

L'Internet des objets apporte plusieurs avantages aux entreprises. Bien que les besoins soient différents d'un secteur à l'autre, les avantages apportés restent assez similaires :

- Une **vision en temps réel** sur tous les processus de production ;
- L'**optimisation du temps**, et donc des dépenses ;
- L'amélioration de la **productivité**;
- Une **prise de décision** améliorée ;

- La possibilité de générer davantage de **revenus**.

L'IoT incite les entreprises à repenser la façon dont elles abordent leurs activités et leur donne les outils nécessaires pour **améliorer leurs stratégies commerciales**.

1.6 Les solutions IoT du marché :

Le marché de l'IoT a longtemps été totalement hétérogène, avec uniquement des solutions propriétaires. Mais depuis quelques années, plusieurs standards de communication et de solutions clé en main ont vu le jour, ce qui donne aux entreprises un large éventail de solutions pour répondre à leurs besoins. Tout cela a été rendu possible grâce au progrès fait sur des technologies et des concepts :

- **L'Intelligence artificiel et le Machine Learning** : on est maintenant capable de développer des systèmes intelligents proches du raisonnement humain, capables de s'adapter à leur environnement et d'apprendre au fil du temp.
- **L'analyse de données** : il existe sur le marché des systèmes capables de lire, analyser et traiter des millions d'informations en temp réel.
- **Les protocoles de communication** : est l'une des composantes essentielles de l'IoT. Que ce soit entre des objets connectés, avec un système central ou avec du Edge, le choix du ou des protocoles de communication qui seront utilisés est très important.

Plusieurs solutions existent sur le marché :

- **Ethernet** : dans cette solution, les objets sont connectés physiquement au réseau interne. Elle est très peu utilisée en raison de la complexité de sa mise en place avec un grand nombre d'objets et surtout à cause des restrictions géographiques qu'elle impose.
- **Wifi** : avec une portée plus large (une centaine de mètres), cette solution n'impose aucune connexion physique et facilite le déploiement sur un périmètre plus ou moins étendu.
- **4G/5G** : solution avec une couverture presque mondiale. Les objets peuvent être connectés pratiquement n'importe où dans le monde et communiquer sans problème avec les systèmes centraux.
- **LR-WPANs** : Low-rate wireless personal area networks est un protocole de communication standardisé spécialement conçu pour l'IoT. Il existe plusieurs implémentations de ce standard comme Zigbee, MiWi ou 6LoWPAN basé sur l'IpV6.
- **Z-Wave** : protocole très utilisé dans la domotique, avec un périmètre assez large pour couvrir une maison.
- **LoRa** : protocole de réseau destiné aux objets connectés fonctionnant sur batterie dans un réseau régional, national ou mondial.

1.7 La 5G pour l'Internet des Objets

Le lancement imminent de la 5G est une bonne nouvelle pour le marché de l'IoT. En effet, ce réseau va grandement améliorer les performances et la fiabilité des objets connectés.

La 5G doit apporter ce que les réseaux mobiles d'ancienne génération ne permettent pas pour l'IoT, en premier lieu l'augmentation de la durée de vie des batteries, mais aussi une couverture mobile plus étendue qu'avec les réseaux mobiles 4G.

A vec ces évolutions, la technologie 5G devrait permettre le déploiement massif des objets connectés voir (figure 1.3) .

En 2030, le nombre total d'appareils en réseau dans le monde pourrait dépasser 100 milliards. L'impact économique est évident pour les acteurs de ce marché qui envisagent un revenu d'environ 4,3 milliards de dollars .

La 5G devrait aussi permettre aux opérateurs de réseau mobile de supprimer leurs réseaux 2G. L'Internet des objets (IoT) critique concerne les équipements qui ont besoin de transmettre des informations en temps réel. Pour cela la latence de la technologie utilisée pour transmettre les données doit être très faible.

La 5G répond aux exigences de l'IoT critique, avec des temps de latence prévus de 1ms, contre 15ms pour la technologie LTE. Il s'agit du domaine où l'IoT va pouvoir prendre son envol avec la 5G.

Grâce à la 5G, la vitesse de transmission des données [1] va être grandement améliorée. Elle sera 10 fois plus rapide que les réseaux LTE actuels ; cela doit permettre aux IoT de communiquer et de transférer des données beaucoup plus rapidement qu'à l'heure actuelle.

Outre l'amélioration de la rapidité, les réseaux 5G seront plus fiables, permettant des connexions plus stables. C'est un facteur extrêmement important pour tout IoT.
En exemple d'IoT critique, on peut citer les communications entre véhicules dans le secteur automobile. Un deuxième exemple dans le secteur médical, la prise en charge de patients en mode automatisé à la suite de la collecte de données médicales récupérées par capteurs

et analysé par de l'intelligence artificielle.

Les réseaux 5G pourront gérer plus d'objets connectés, permettant aux consommateurs de bénéficier d'une meilleure fiabilité pour leurs IoT.

[1] La transmission de données est le processus de transfert de données numériques ou analogiques entre deux ou plusieurs appareils via un support de communication, tel que des fils de cuivre, des fibres optiques ou des signaux sans fil.

Figure 1.3 La 5G entre 2020 et 2030 *(*Les plateformes Cloud IoT du marché) ,

1.8 Les technologies Edge de Microsoft

En 2018, Microsoft a annoncé l'**investissement de 5 milliards de dollars dans IoT** et dans les technologies Edge, ce qui a permis de développer plusieurs solutions et services, comme l'IoT Central, Azure Sphère, IoT Hub et IoT Edge, IoT Stack et bien d'autres.

Deux ans plus tard, Microsoft était classé par Gartner comme **leader du marché des solutions IoT pour les industriels** et voyait ses solutions intégrées par plusieurs grands industriels tels que :

- **Starbucks** qui utilise Azure Sphère pour interconnecter toutes les machines pour une gestion automatisée de ces dernières, ce qui permet de libérer plus de temps aux employés pour s'occuper des clients.
- **Volkswagen** qui, depuis 2020, équipe les nouvelles voitures avec une solution Edge Microsoft afin de suivre leur état et d'assurer au mieux leur maintenance.

1.8.1 Google Cloud IoT Platform

Google propose une suite de solutions **IoT** capables de gérer les données de millions de devises en temps réel :

- **Google IoT Core** pour la gestion des devises, de l'authentification et de la communication ;
- **Cloud DataFlow** pour traiter les données en temps réel ;
- **BigQuery** pour analyser les données en temps réel.

L'un des points forts de la plateforme de Google est la data avec toutes les solutions proposées qui sont capables de gérer, stocker et traiter des millions de données en temps réel. En plus de son service Google Maps, elle propose également une vision géographique des données IoT voir figure (1.4).

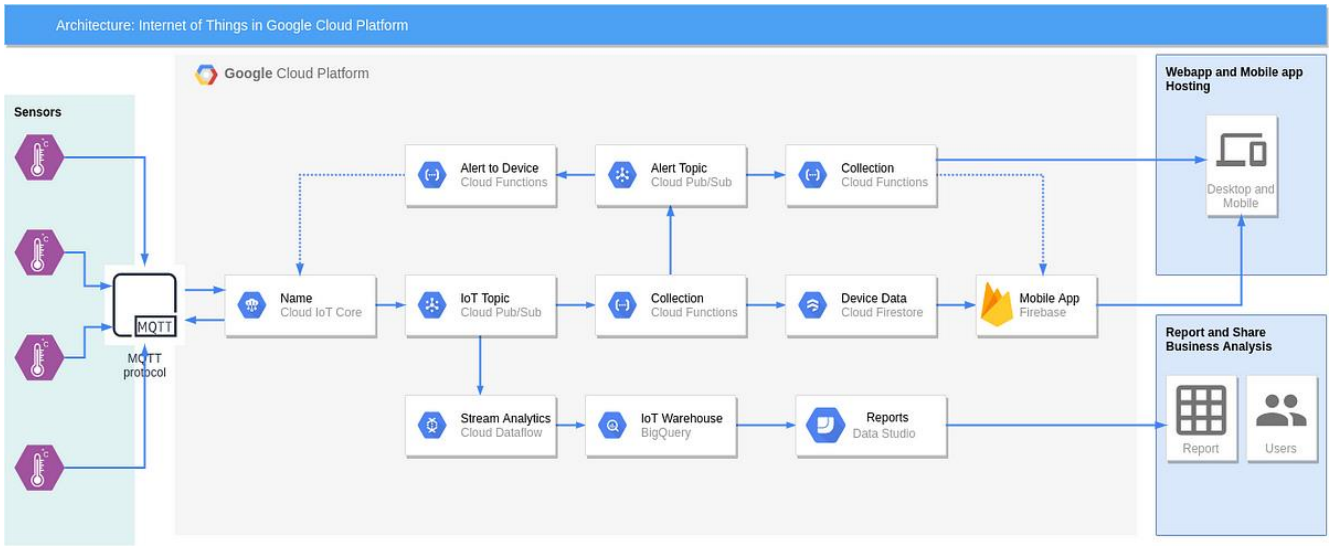

Figure 1.4 architecture Google Cloud IoT

1.8.2 AWS IoT Platform

Amazon est l'un des acteurs reconnus dans le marché de l'IoT, avec des projets comme les « Smart cities » qui ont permis à des villes telles que Chicago d'automatiser et d'optimiser toute leur administration, ou encore les maisons connectées où Amazon fait figure de leader avec ses objets connectés et son service Alexa.

Forte de son expertise du Cloud, Amazon propose une panoplie de services IoT, à commencer par IoT Core. Au cœur de la solution IoT d'Amazon, **IoT Core** gère toute la partie communication et authentification mais aussi le « Device Shadow » qui représente une gestion d'état asynchrone entre la plateforme et les devises voir figure (1.5).

D'autres solutions viennent compléter l'offre, comme le **FreeRTOS** pour le développement en Edge ou **IoT Device Management** pour la gestion de flottes de devices.

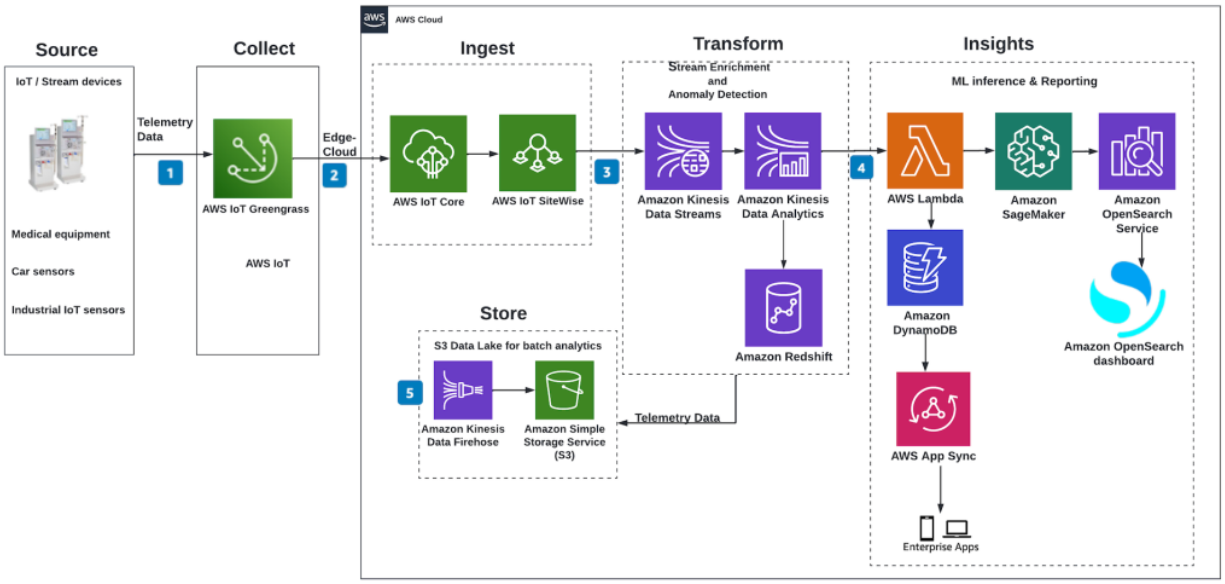

***Figure 1.5 architecture* AWS IoT**

1.9 Quels sont les domaines d'application de l'internet des objets ?

L'internet des objets offre de nombreuses applications à ses utilisateurs. Parmi ces applications nous citons :

1.9.1Les Villes Intelligentes

Beaucoup de grandes villes ont été soutenues par des projets intelligents, comme Séoul, New York, Tokyo, Shanghai, Singapour, Amsterdam et Dubaï. Les villes intelligentes (voir Figure (1.6) peuvent encore être considérées comme des villes de l'avenir et la vie intelligente, et par le taux d'innovation[1] de la création de villes intelligentes d'aujourd'hui, il sera devenu très faisable pour entrer la technologie IoT dans le développement des villes.

La demande exige une planification minutieuse à chaque étape, avec l'appui de l'accord des gouvernements, citoyens à mettre en œuvre la technologie d'Internet des objets dans tous les aspects. Par l'IoT, les villes peuvent être améliorées à plusieurs niveaux, en améliorant les infrastructures, en améliorant les transports .

[1] **Le taux d'innovation,** dans son sens le plus général, mesure la proportion de projets réussis par rapport à l'ensemble des projets entrepris.

Figure 1.6 Représentation des constituants d'une ville intelligente

1.9.2 Le Smart Grid

L'un des domaines d'application de l'IoT est le secteur de la distribution d'énergie intelligente, dit « Smart Gr id » (voir figure1.7). En France, ERDF est très actif dans le développement de ce domaine, où un besoin clair en récupération d'information à différents points du réseau électrique est devenu nécessaire pour une meilleure intégration des différentes sources d'énergies et une meilleure gestion de la distribution jusqu'aux utilisateurs finaux.

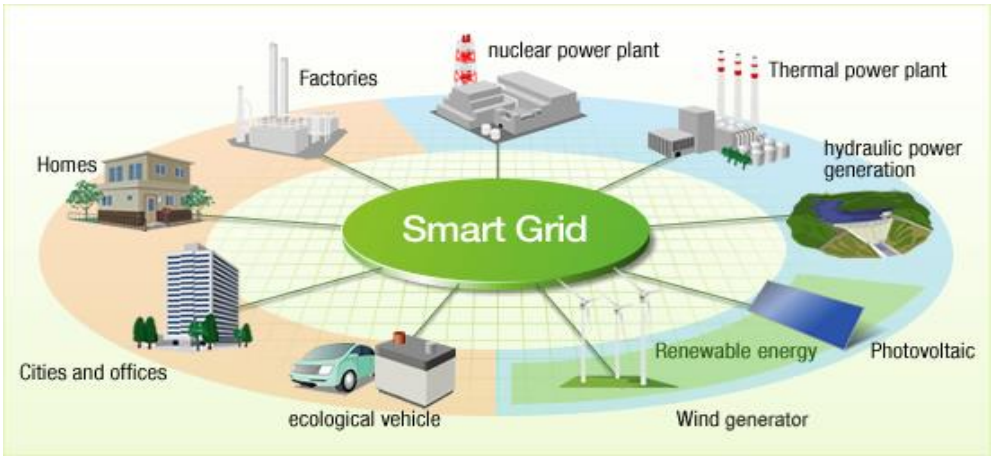

Figure 1.7 Représentation des constituants d'une smart Grid

1.9.3 Le Système De Santé Electronique

Des appareils intelligents (voir figure 1.8) dans les soins de santé sont utilisés pour stocker et gérer les paramètres de soins clés et pour gérer les données sur les maladies capturées. Ils sont principalement déployés pour fournir des solutions de conditionnement

physique en suivant les activités ciblées et des dispositifs de diagnostic utilisés pour stocker des données de dispositifs. Principalement, ils sont utilisés comme des solutions de fitness pour suivi des activités du patient et des appareils de diagnostic intelligents tels que les dispositifs de tension matérielle, les podomètres[1], Google Glass[2], etc. utilisé pour capturer les données des capteurs, pour une analyse plus approfondie par le médecin.

1 Un **podomètre** est un dispositif sensible au mouvement permettant de mesurer en temps réel le nombre de pas d'une personne,
2**Google Glass** est une paire de lunettes avec une réalité augmentée

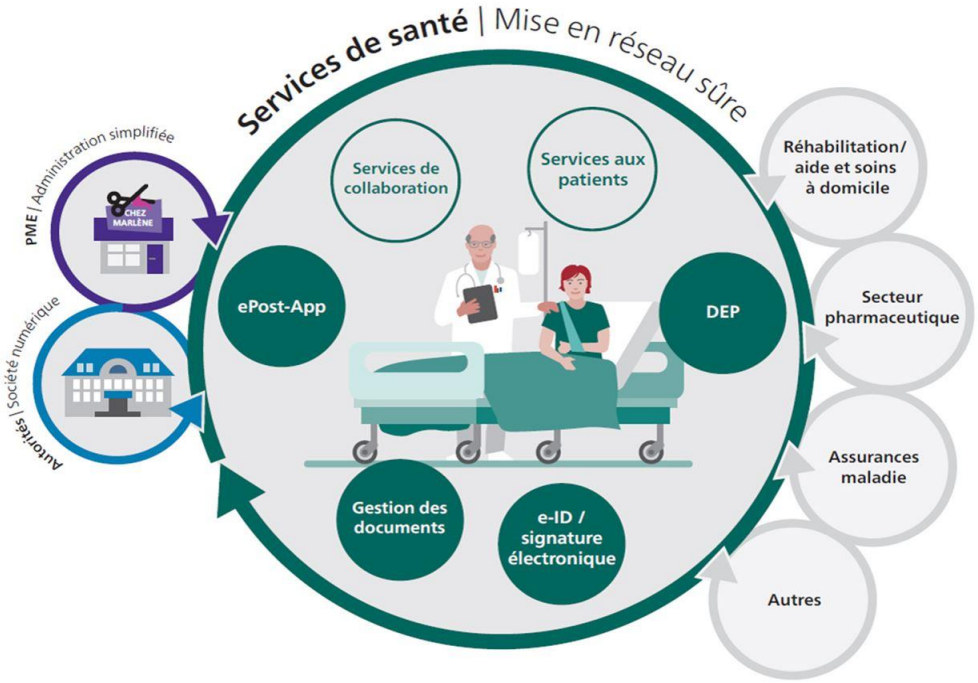

Figure 1.8 un système de santé électronique

1.9.4 Les Maisons et Les Bâtiments Intelligents

Les technologies Wi-Fi dans la domotique ont été principalement utilisées pour plusieurs raisons qui sont :

a. Les appareils électroniques tels que les téléviseurs, les appareils mobiles…., généralement pris en charge cette technologie.

b. Le taux croissant d'adoption de dispositifs informatiques mobiles comme les téléphones intelligents, les tablettes.

c. Les appareils mobiles garantissent que les consommateurs peuvent

accéder aux « contrôleurs » des appareils portables connectés à un réseau.

A l'aide du concept de l'internet des objets, les maisons et les bâtiments peuvent exploiter alors de nombreux appareils et objets intelligemment (voir figure 1.9 1.10). Comme un exemple d'application intéressante de l'IoT dans les maisons intelligentes et les bâtiments on trouve : l'éclairage intelligent, le contrôle de l'air et de chauffage central, la gestion de l'énergie et la sécurité.

Les réseaux de capteurs sans fil (WSN) Avec intégration de la technologie de l'internet des objets fourniront une gestion intelligente de l'énergie dans les bâtiments. D'autre part, l'internet avec des systèmes de gestion de l'énergie aussi offre la possibilité d'accéder aux systèmes d'information et de contrôler l'énergie d'un bâtiment par un ordinateur portable ou un smartphone placé n'importe où dans le monde.

Figure 1.9 : les maisons intelligentes

Figure 1.10 : Bâtiment intelligent

1.9.5 Les Usines et L'Industrie Intelligente

L'usine intelligente a ajouté une nouvelle valeur dans la révolution de la fabrication en intégrant l'intelligence artificielle, l'apprentissage automatique et l'automatisation du travail et la communication M2M avec le processus de fabrication. L'usine intelligente va changer fondamentalement, comment les produits sont inventés, fabriqués et expédiés. En même temps, cela améliorera la sécurité des travailleurs et protège l'environnement un faible incident de fabrication. Ces progrès dans la façon dont les machines et autres objets communiquent, et la manière dont la prise de décision passe des humains aux systèmes techniques signifie que la fabrication devient "plus intelligente".

La révolution des industries et de la fabrication est devenue l'une des plus technologies développées de nos jours, la croissance de l'évolution de l'industrie a pris de nombreuses générations. La première génération liée aux machines mécaniques en plus de la puissance de l'eau et du courant. La deuxième génération de l'industrie traite de la production de masse, des chaînes de montage et de l'électricité. Dans la fin du dernier siècle, les industries sont exploitées sous le contrôle des ordinateurs et de l'automatisation qui est reconnu par la troisième génération d'industries.

L'industrie intelligente c'est la quatrième génération connue par l'industrie 4.0 est basée sur les systèmes de chiffrement physiques qui est capables de se connecter à Internet.

Le concept de l'industrie 4.0 avec l'Internet des objets peut atteindre de grandes attentes pour les accords de résolution des industries avec de nombreux aspects sont illustrés à la figure voir figure (1.11) .

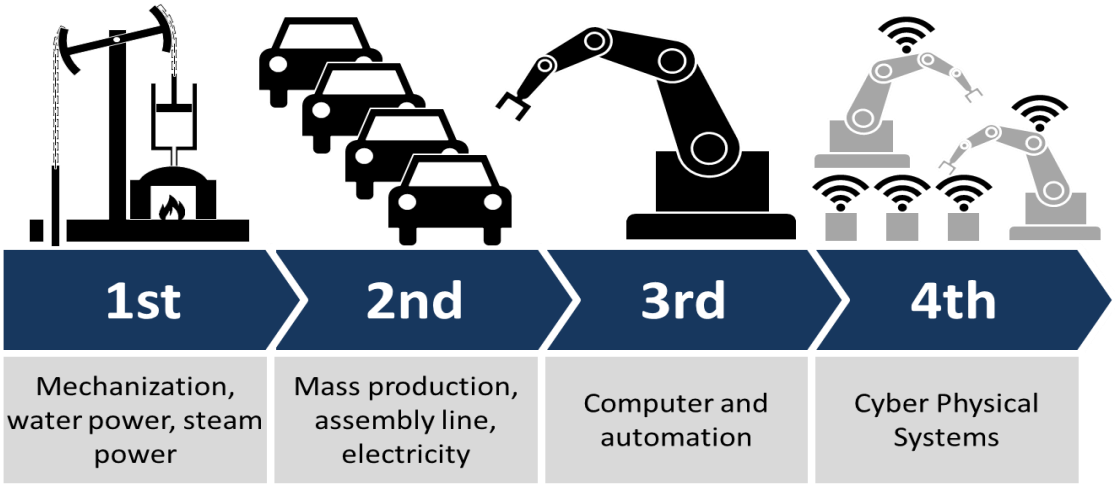

Figure 1.11 Schéma de l'industrie 4.0

1.10 Conclusion

L'Internet des Objets (IoT) représente une avancée technologique majeure qui transforme en profondeur notre manière d'interagir avec le monde physique. En intégrant des objets connectés capables de collecter, analyser et transmettre des données, l'IoT ouvre la voie à de nouveaux usages dans des domaines aussi variés que la santé, les villes intelligentes, l'industrie, l'énergie ou encore la domotique.

Grâce à l'évolution rapide des technologies telles que la 5G, l'intelligence artificielle, le cloud computing et les protocoles de communication, l'IoT devient de plus en plus performant, fiable et accessible. Les avantages qu'il offre – gain de temps, optimisation des ressources, automatisation des processus, amélioration des services – en font un levier stratégique pour les entreprises et les collectivités.

Cependant, l'adoption massive de cette technologie soulève aussi des défis importants, notamment en matière de sécurité, de confidentialité des données et d'interopérabilité entre les systèmes.

Ainsi, une compréhension approfondie de l'IoT, de ses composants et de ses enjeux est indispensable pour tirer pleinement parti de son potentiel tout en anticipant les risques liés à son déploiement à grande échelle.

Chapitre 02 :

Systèmes intelligents et gestion de parking-auto

2.1 Introduction

Ce chapitre explore l'application de l'Internet des Objets (IoT) dans la gestion intelligente des parkings automobiles. Face à l'urbanisation croissante et à l'augmentation du nombre de véhicules, les villes sont confrontées à des défis majeurs en matière de stationnement. Les systèmes de parking intelligents, intégrant des technologies avancées, offrent des solutions innovantes pour optimiser l'utilisation des espaces de stationnement et améliorer l'expérience des usagers.

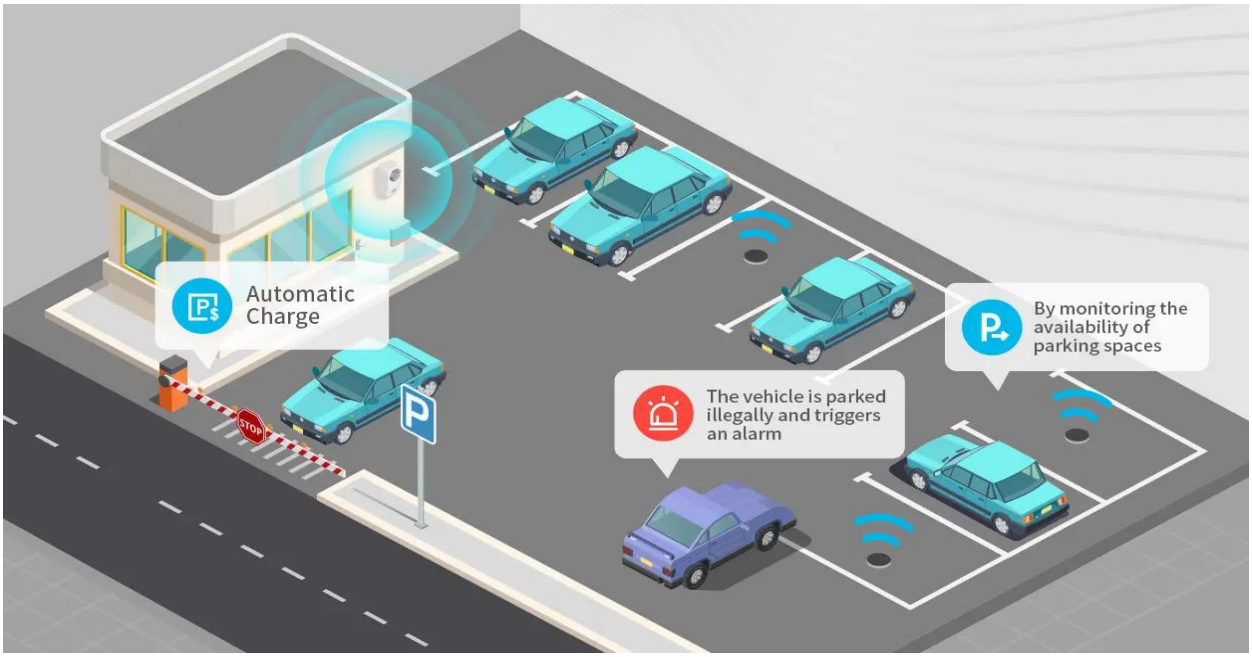

Figure 2.1 : la gestion intelligente des parkings automobiles.

2.2 Problématique du stationnement urbain

La croissance rapide du parc automobile a entraîné une saturation des espaces de stationnement, particulièrement dans les zones urbaines denses. Cette situation engendre plusieurs problèmes :

- **Temps de recherche prolongé** : Les conducteurs passent en moyenne 20 minutes à chercher une place, ce qui représente environ 72 heures par an .
- **Impact environnemental** : La circulation à la recherche de stationnement contribue à environ 60 % de la pollution urbaine .
- **Occupation inefficace de l'espace public** : À Paris, par exemple, 50 % de l'espace public est dédié à la voiture, qui ne représente que 13 % des déplacements .
- **Coûts économiques** : Les infrastructures de stationnement nécessitent des investissements importants de la part des municipalités, tant en termes de construction que de maintenance .

2.3 Présentation du concept de Parking Intelligent

Le parking intelligent (ou *smart parking* en anglais) désigne un système intégré exploitant les technologies de l'Internet des Objets (IoT), de la télémétrie, de l'intelligence artificielle (IA) et des interfaces mobiles afin d'optimiser la gestion des places de stationnement dans les zones urbaines ou privées.

L'objectif principal est de réduire le temps de recherche de places, fluidifier la circulation, diminuer la pollution et améliorer l'expérience utilisateur(figure 2.2).

2.3.1 Fonctionnalités principales

Un système de parking intelligent propose généralement les services suivants :

- **Détection automatique des places disponibles**

  Grâce à des capteurs (ultrasons, infrarouge, magnétiques) ou des caméras intelligentes, chaque emplacement de stationnement peut être surveillé en temps réel pour savoir s'il est occupé ou libre.
- **Affichage et guidage en temps réel**

  Les données collectées sont transmises à une application ou à des panneaux LED

indiquant aux conducteurs la position exacte des places libres, réduisant ainsi le temps de recherche.

- **Réservation de place à l'avance**
  Certains systèmes permettent à l'usager de réserver une place de parking depuis une application mobile avant son arrivée, évitant ainsi les files d'attente.
- **Paiement automatisé**
  Les plateformes de smart parking intègrent des solutions de paiement via carte bancaire, portefeuille électronique ou QR Code, rendant le processus fluide et sans contact.
- **Surveillance et sécurité**
  Des caméras intelligentes peuvent surveiller les allées et venues, lire les plaques d'immatriculation (LPR/ANPR), et détecter les comportements anormaux (stationnements prolongés, tentatives de vol...).

2.3.2 Technologies mobilisées

Pour fonctionner, le système intelligent de parking s'appuie sur une combinaison de technologies :

- **Capteurs IoT** : pour détecter la présence ou l'absence d'un véhicule.
- **Réseaux de communication** : comme LoRaWAN, ZigBee, NB-IoT ou Wi-Fi pour transmettre les données au serveur central.
- **Plateforme cloud** : pour centraliser, traiter et analyser les données en temps réel.
- **Applications mobiles/Web** : pour interagir avec l'utilisateur final.
- **Bases de données intelligentes** : pour stocker les historiques de stationnement, générer des rapports, ou même prédire la disponibilité des places à certaines heures.

2.3.3 Avantages du parking intelligent

1. **Gain de temps** : le conducteur trouve une place rapidement.

2. **Réduction du trafic et de la pollution** : moins de circulation inutile.
3. **Utilisation optimisée des ressources** : moins de places vides, meilleure rotation.
4. **Sécurité accrue** : surveillance vidéo, contrôle des accès.
5. **Intégration avec les villes intelligentes (Smart Cities)** : le système peut communiquer avec les transports publics, les services d'urgence ou les plateformes de mobilité.

Figure 2.2 L'objectif principal de système intelligent de parking .

2.4 Composants d'un système de parking intelligent

Un système de parking intelligent se compose généralement des éléments suivants (figure 2.3):

- **Capteurs de présence** : Utilisant des technologies telles que les ultrasons, l'infrarouge ou les capteurs magnétiques pour détecter l'occupation des places .
- **Caméras intelligentes** : Équipées de logiciels de reconnaissance de plaques (ANPR) pour surveiller les entrées/sorties et assurer la sécurité .
- **Barrières et systèmes d'accès automatisés** : Contrôlés à distance pour gérer l'entrée et la sortie des véhicules.
- **Applications mobiles/web** : Permettent aux utilisateurs de localiser, réserver et payer leur stationnement.
- **Plateformes cloud** : Centralisent les données collectées pour une analyse en temps réel et une gestion efficace .

Figure 2.3 Composants d'un système de parking intelligent

2.5 Architecture générale d'un système intelligent de parking

L'architecture typique d'un système de parking intelligent comprend :

1. **Capteurs** : Installés sur chaque place pour détecter la présence d'un véhicule.

2. **Passerelle IoT** : Collecte les données des capteurs et les transmet au cloud via des protocoles comme MQTT ou HTTP .
3. **Serveur cloud** : Stocke et analyse les données pour déterminer la disponibilité des places.
4. **Application utilisateur** : Affiche en temps réel les informations sur les places disponibles et permet la réservation/paiement.
5. **Interface d'administration** : Offre aux gestionnaires une vue d'ensemble pour surveiller et optimiser le système.

Un schéma illustratif de cette architecture peut être inclus pour une meilleure compréhension(figure 2.4).

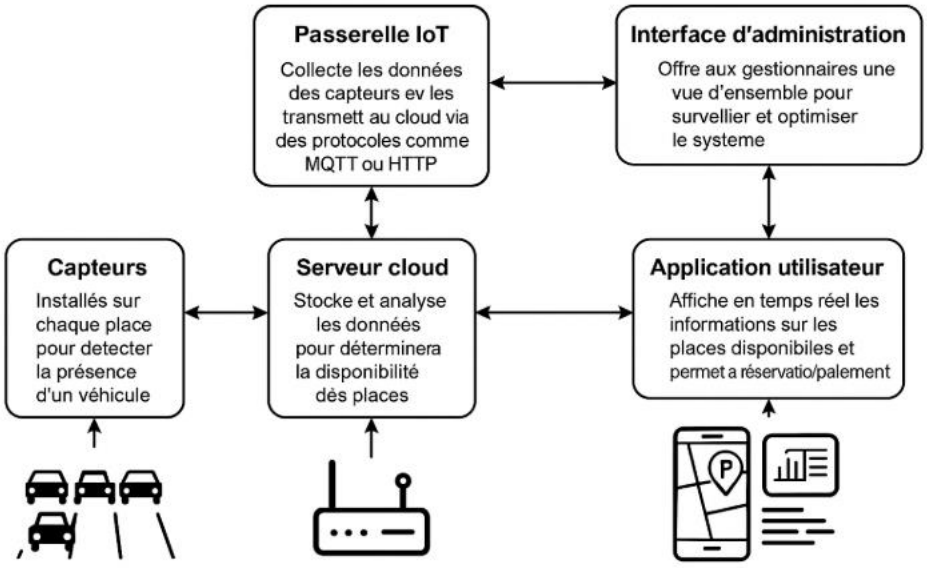

Figure 2.4 Architecture générale d'un système

2.6 Technologies utilisées

Les technologies clés dans les systèmes de parking intelligents incluent :

- **Protocoles de communication** : MQTT, HTTP, LoRaWAN pour la transmission des données.
- **Matériel** : Microcontrôleurs comme Raspberry Pi ou Arduino pour le traitement local des données .
- **Logiciels** : Applications développées en React, Angular ou Flutter pour les interfaces utilisateur.
- **Cloud computing** : Services comme AWS, Azure ou Firebase pour l'hébergement et l'analyse des données.
- **Intelligence artificielle** : Utilisée pour la reconnaissance de plaques, la prédiction de la disponibilité des places et l'optimisation du stationnement .

2.7 Exemples de systèmes existants

Plusieurs entreprises ont développé des solutions de parking intelligent :

- **Bosch** : Propose un système basé sur l'IA pour détecter les places disponibles et guider les conducteurs en temps réel .

- **Cisco & Streetline** :
  Cisco, en partenariat avec Streetline, a mis en place des capteurs enfouis dans le sol pour détecter l'occupation des places et transmettre ces données en temps réel aux conducteurs via une application mobile.
- **Smart Parking Ltd (Nouvelle-Zélande)** :
  Fournit des solutions complètes de gestion de parking incluant capteurs, systèmes de paiement automatisés et tableau d'analyse pour les opérateurs.
- **Parkopedia** :
  Une plateforme globale qui utilise des données collectées via les utilisateurs et des capteurs pour informer en temps réel de la disponibilité de places dans plus de 15 000 villes.

- **Solutions universitaires** :
  Plusieurs projets étudiants, notamment dans les universités techniques (INSA, Polytech, etc.), proposent des maquettes de systèmes de parkings intelligents intégrant des Raspberry Pi, capteurs ultrasons et interfaces web.

2.8 Limites des solutions actuelles

Même si les systèmes intelligents de stationnement sont prometteurs, certaines limites persistent :

- **Coût d'installation élevé** :
  Les capteurs, caméras, réseaux de communication et plateformes cloud représentent un investissement initial important, surtout pour les grandes villes.
- **Maintenance régulière** :
  Les capteurs peuvent être sensibles aux conditions météorologiques, aux interférences électromagnétiques ou au vandalisme, nécessitant un entretien fréquent.
- **Besoin de couverture réseau fiable** :
  Les systèmes connectés dépendent d'une bonne couverture Internet ou réseau bas débit (LoRa, NB-IoT), ce qui peut être un obstacle dans certaines zones.
- **Sécurité et confidentialité des données** :
  Le traitement de données personnelles (plaque d'immatriculation, géolocalisation) soulève des questions de conformité au RGPD (en Europe notamment).
- **Interopérabilité** :
  Les solutions propriétaires ne sont pas toujours compatibles entre elles, rendant difficile l'intégration à grande échelle dans les villes intelligentes.

2.9 Conclusion

En conclusion, les systèmes de parking intelligents, basés sur l'Internet des Objets (IoT), sont une réponse innovante aux défis du stationnement urbain, tels que la recherche prolongée de places, la pollution et l'utilisation inefficace de l'espace public. Ces systèmes optimisent la gestion des places de stationnement en intégrant des technologies comme les capteurs, l'IA et les interfaces mobiles.

Les avantages sont multiples : gain de temps pour les conducteurs, réduction du trafic et de la pollution, utilisation optimisée des ressources, sécurité accrue et intégration facilitée dans les villes intelligentes. L'architecture de ces systèmes repose sur des capteurs, une passerelle IoT, un serveur cloud, une application utilisateur et une interface d'administration. Des entreprises comme Bosch, Cisco & Streetline, Smart Parking Ltd, et Parkopedia ont déjà développé des solutions de parking intelligent.

Cependant, des défis subsistent, notamment le coût d'installation élevé, la nécessité d'une maintenance régulière, l'exigence d'une couverture réseau fiable, les préoccupations concernant la sécurité et la confidentialité des données, ainsi que les problèmes d'interopérabilité entre les différentes solutions. Malgré ces limites, le parking intelligent représente une avancée significative vers une gestion urbaine plus efficace et durable.

Chapitre 03:

Les Réseaux de Pétri

3.1 Introduction :

De nos jours les systèmes informatiques sont devenus d'une grande complexité. En plus les systèmes dynamiques ne peuvent pas être décrits en ne prenant en compte que leurs états initiaux et finaux. En effet, on doit tenir compte de leur comportement permanent qui est une séquence d'états, pour qu'on puisse parler d'une description bien fondée. Parmi le grand nombre des techniques formelles qui ont déjà été proposées pour spécifier, analyser et vérifier ce genre de systèmes, les réseaux de pétri sont l'une des plus utilisés

Les réseaux de pétri offrent un outil formel (Sémantique et Mathématique) et une bonne représentation graphique qui permettent de modéliser et d'analyser les systèmes discrets, particulièrement les systèmes concurrents et parallèles. La facette graphique des réseaux de pétri, nous aide à comprendre aisément le système modélisé. Par ailleurs, leur puissance d'expression mathématique permet de simuler des activités dynamiques et concurrentes. L'intérêt majeur de ces réseaux réside dans leurs possibilité d'analyser les systèmes modélisés, grâce aux modèles de graphes et aux équations algébriques (aspect mathématique). Les réseaux de pétri sont des outils de modélisation utilisés généralement en phase préliminaire de conception de système afin de réaliser leur spécification fonctionnelle, modélisation et suivre leur évaluation. Grâce à leur expressivité et à leur souplesse, ils sont utilisés dans une large variété de domaines. Ils permettent notamment : La modélisation des systèmes informatiques, l'évaluation des performances des systèmes discrets, des interfaces homme-machine, la commande des ateliers de fabrication, la conception de systèmes temps réel, la modélisation des protocoles de communication et la modélisation des chaines de production (de fabrication), les systèmes distribués et l'architecture des ordinateurs, … etc

3.2Définition de RdP

Un Réseau de Pétri (RDP) est un graphe composé de deux sortes de nœuds :

- Les places :*représentées par des cercles* qui permettent la description des états *conditions*, possibles du système. 
- Les transitions : *représentées par des barres* qui permettent la description des événements ou les actions qui causent le changement de l'état.

- Satisfaction d'une condition: modélisée à l'aide d'un jeton
  - Condition vraie 
  - Condition fausse 

Les Réseaux de Pétri sont un outil formel efficace pour modéliser et analyser les systèmes à événements discrets. Ils permettent de distinguer clairement entre la structure statique du système et son comportement dynamique. Grâce à leur représentation graphique intuitive, ils sont facilement compréhensibles par les utilisateurs.

L'analyse d'un réseau de Pétri permet de dégager des propriétés essentielles du système, tant sur le plan structurel que comportemental. Ces résultats servent à évaluer le fonctionnement global du système et facilitent sa modification ou son amélioration voir figure (3.1).

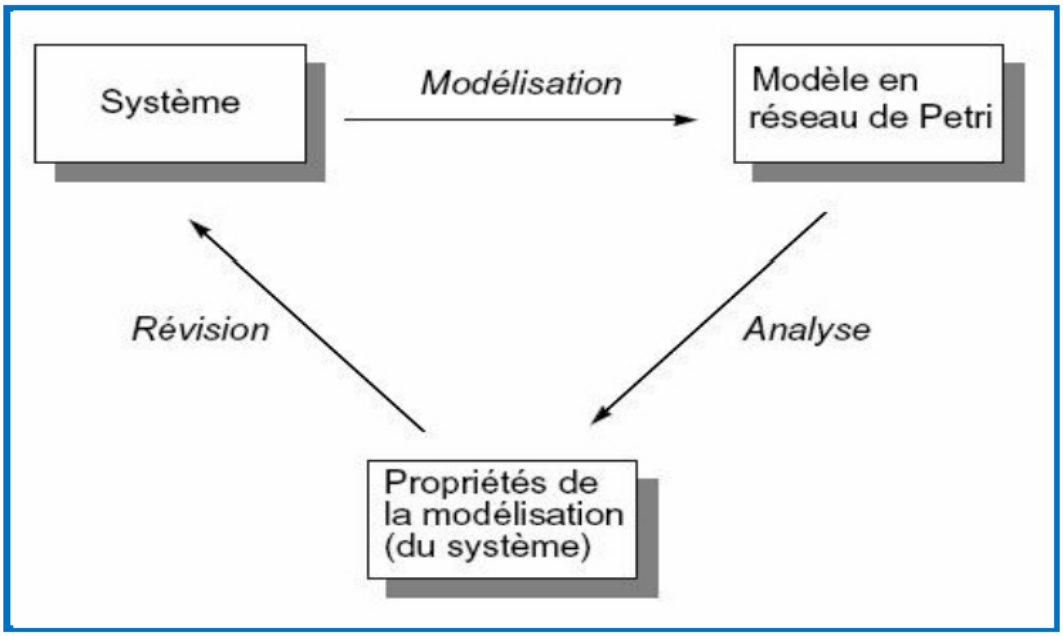

Figure 3. 1 : modélisation et analyse basée sur les rdp.

3.3 Concepts de base pour les RDP

Il existe trios concepts de base :

- **La Condition**

Une **condition** (1) représente une **situation, un état partiel ou une propriété logique** du système à un instant donné. Elle agit comme une **variable binaire** pouvant prendre deux états :

- **Vraie**, si la condition est satisfaite (par exemple : "la machine est prête", "la porte est fermée").
- **Fausse**, si elle ne l'est pas.

Les conditions permettent donc de **décrire l'état global** du système sous forme d'un ensemble de faits ou desituations. Dans un RDP, les conditions sont **représentées graphiquement par des cercles**, et leur état est marqué par des **jetons**:

- La présence d'un jeton dans une place signifie que la condition correspondante est vraie.
- L'absence de jeton signifie que la condition est fausse.

Ainsi, l'**état du système** à un instant donné est défini par l'ensemble des places contenant un ou plusieurs jetons. Cet état est appelé **marquage**.

- **L'Événement**

Un **événement** (2) désigne une **action, un changement ou un processus** qui peut se produire dans lesystème.
Un événement est **responsable de la transformation de l'état du système** : lorsqu'il se produit, il modifie la valeur de certaines conditions (certaines deviennent fausses, d'autres deviennent vraies).

Graphiquement, un événement est **représenté par un rectangle ou un trait vertical**. Il est **lié aux conditions** par des arcs dirigés :

- **Des arcs entrants** relient les **conditions préalables** (pré-conditions) à l'événement.
- **Des arcs sortants** relient l'événement aux **nouvelles conditions** (post-conditions) qui deviendront vraies après le déclenchement.

Un événement ne peut se produire que si les **conditions requises sont réunies**, c'est-à-dire si les places en entrée contiennent un nombre suffisant de jetons.

➢ **Le Déclenchement (pré-conditions (3) et post-conditions (4))**

Le **déclenchement** d'un événement est le mécanisme central de l'évolution dans un Réseau de Pétri. Il dépend directement des **pré-conditions** :

- Les **pré-conditions** sont les **conditions nécessaires pour qu'un événement soit déclenchable**.
  Cela signifie que l'ensemble des places connectées à l'entrée de la transition (événement) doivent contenir les jetons requis.
  Si ce n'est pas le cas, l'événement reste inactif.
- Une fois l'événement **déclenché**, il **consomme les jetons** des places d'entrée (les pré-conditions deviennent fausses), et **produit des jetons** dans les places de sortie (les post-conditions deviennent vraies).
  Ce processus correspond à une **évolution de l'état du système**.

(1)Condition État logique du système, représenté par une place (présence d'un jeton = vrai)
(2)Événement Action qui peut modifier l'état du système (représenté par une transition)
(3) Pré-condition Condition qui doit être vraie pour permettre le déclenchement d'un événement
(4) Post-condition Condition qui devient vraie après le déclenchement d'un événement

3.4 Aspect structurel

3.4.1. Définition formelle

Un réseau de Pétri est défini comme un quadruplet :

PN = (P, T, E, S)

- P : Ensemble fini de places $\{p_1, p_2, ..., p_m\}$
- T : Ensemble fini de transitions $\{t_1, t_2, ..., t_n\}$
- E : Fonction d'incidence avant ($E : P \times T \rightarrow \mathbb{N}$), relie transitions à places
- S : Fonction d'incidence arrière ($S : P \times T \rightarrow \mathbb{N}$), relie places à transitions → Les ensembles P et T sont disjoints ($P \cap T = \emptyset$)

3.4.2 . Représentation graphique <Définition informelle>

- Places : cercles, contiennent des jetons
- Transitions : rectangles, représentent des événements
- Arcs : entre places et transitions uniquement
- Marquage : distribution des jetons dans les places

Exemple d'application :

Addition de deux entiers naturels (n1 et n2)voir figure (3.2) :

- P1 et P2 contiennent les valeurs de n1 et n2 (en nombre de jetons)
- Transitions T1 et T2 déplacent les jetons vers P3

- Résultat dans P3 = n1 + n2 (nombre de jetons)

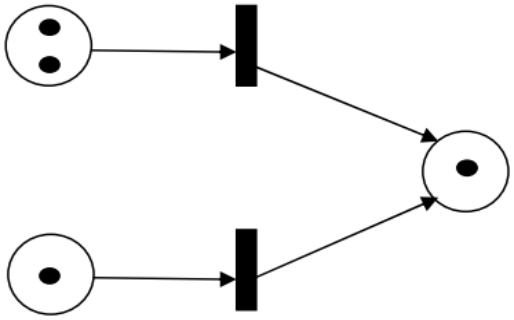

Figure 3.2 : Exemple d'addition de 2 entiers par un rdp.

3.5 Aspect comportemental

3.5.1. Marquage

Un réseau de Pétri marqué est un couple {R, M}, où :

- R : réseau de Pétri
- M : P → ℕ :

fonction de marquage *associe à chaque place un nombre de jetons*.

Un réseau marqué est souvent noté Σ = (P, T, W, M_0) :

- M_0 : marquage initial
- W : pondérations des arcs

Ci-dessous un exemple de marquage est schématisé voir figure (3.3) :

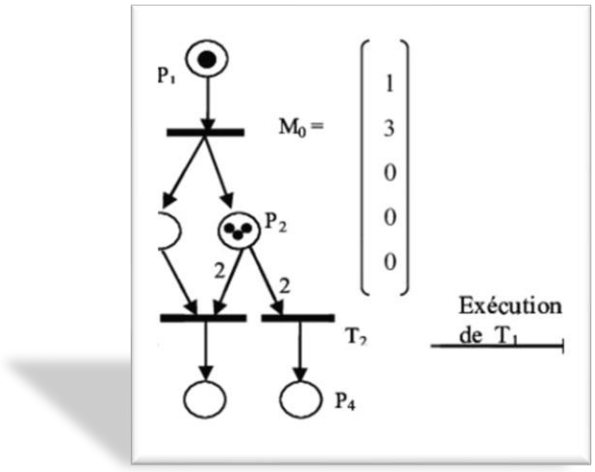

Figure 3.3 : Le marquage.

3.5.1.1. Marquage accessible

L'ensemble des marquages est l'ensemble Mi(figure 3.4) qui peuvent être atteint par le franchissement d'une S à partir du marquage initial **M0. *M0 = {Mi tel que Mi[s >Mj]}** Comme il est représenté par l'exemple suivant :

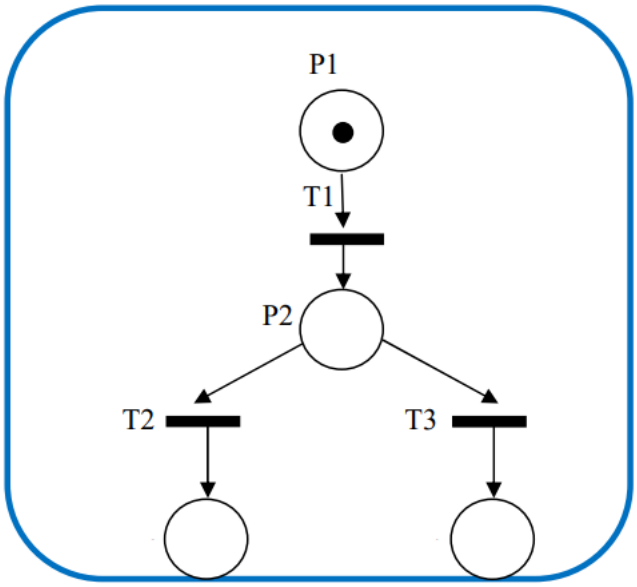

Figure 3.4 : Ensemble des marquages accessibles

Soit le marquage initial :
M0=[1 0 0 0]

Les marquages atteignables à partir de M0 sont :

- M1=[0 1 0 0]
- M2=[0 0 1 0]
- M3=[0 0 0 1]

Ainsi, l'ensemble des marquages accessibles est :
{M0,M1,M2,M3}

3.5.1.2. Graphe de marquage accessible

Lorsqu'un réseau de Petri présente un nombre fini de marquages accessibles, on peut représenter leur évolution par un graphe de marquage. Ce graphe permet de visualiser la dynamique du réseau, en montrant les relations entre les différents marquages accessibles et les transitions qui les relient.

Dans l'exemple précédent(figure 3.5), ce graphe reflétera les passages successifs de M0 vers M1 , de M1 vers M2, et ainsi de suite, jusqu'à M3.

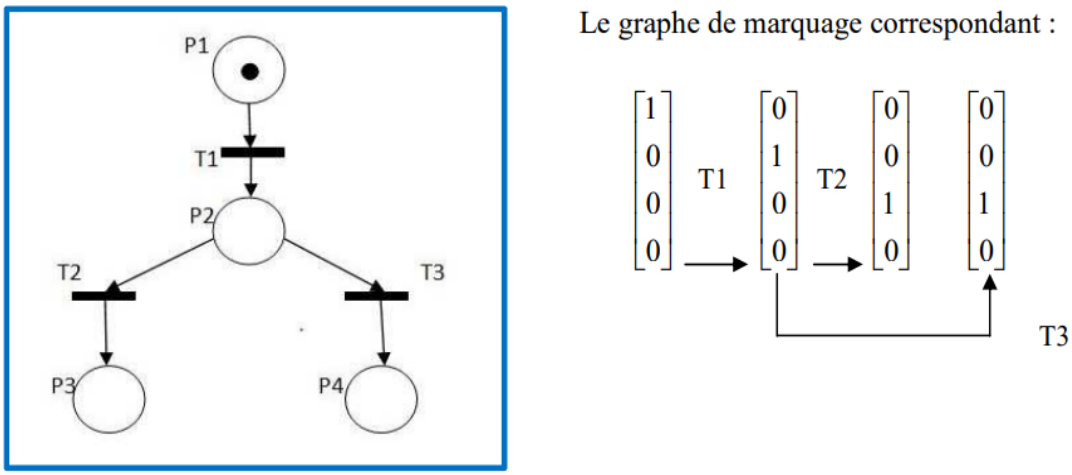

Figure 3.5 : Le graphe de marquage accessible.

3.5.2 Franchissement d'une transition

- Transition franchissable : si chaque place d'entrée contient au moins un jeton
- Franchissement (figure 3.6)
  - Retrait d'un jeton dans chaque place d'entrée
  - Ajout d'un jeton dans chaque place de sortie

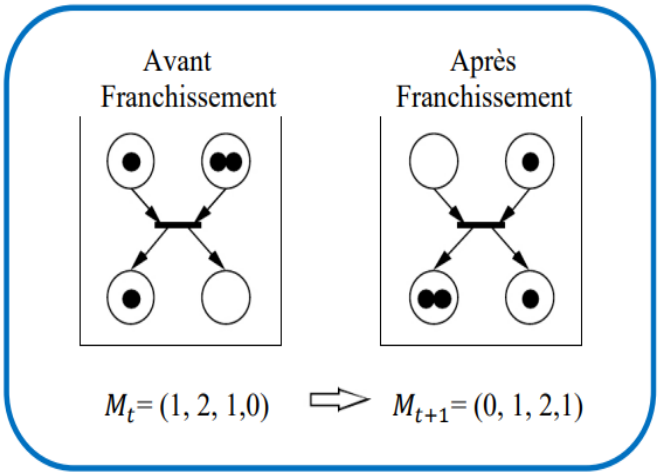

Figure3.6 : Le franchissement d'une transition.

3.5. 2.1 Règles générales de fonctionnement

A. Une transition est validée si toutes ses places en entrée ont au moins un jeton.

B. Une seule transition peut être franchie à la fois parmi celles validées.

C. Le franchissement est indivisible et instantané .

3.5.2.2 Séquence de franchissement

Dans un réseau de pétri une séquence de franchissement S est une suite de transition Ti, Tj,….,Tk qui peuvent être franchis successivement à partir d'un marquage donné. Une seule transition peut être franchie à la fois(figure 3.7).

On note : Mi[S>Mj] à partir du marquage Mi, le franchissement de la séquence S aboutit au marquage Mj.

Exemple : T1 T2 et T1 T3 sont deux séquences de franchissement.

M0 [T1 T2>M1] et M0 [T1 T3>M2] avec M1= [0 0 1 0] t et M2= [0 0 0 1] t.

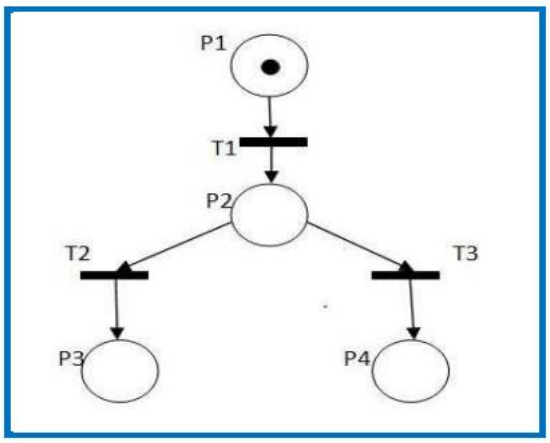

Figure 3.7 : Une séquence de franchissement.

3.6 Les différents types de Réseaux de Pétri

Il existe plusieurs variantes des Réseaux de Pétri permettant de modéliser des aspects spécifiques des systèmes dynamiques. Parmi les plus couramment utilisés, on distingue :

3.6.1 Les Réseaux de Pétri Temporisés

Un réseau de Pétri classique permet de modéliser des relations causales entre événements de manière qualitative : un événement « a » précède un autre événement « b » si « a » en est la cause. Toutefois, cette modélisation ne tient pas compte du temps réel de manière explicite.

Les Réseaux de Pétri temporisés introduisent la dimension temporelle de manière quantitative, en associant des durées aux composants du réseau. Le temps devient ainsi une partie intégrante du contrôle du système, et non une simple donnée non structurée (figure 3.8).

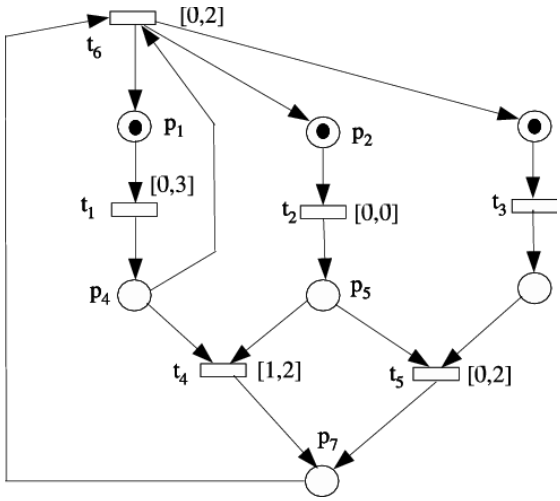

Figure 3.8 :exemble de Réseaux Pétri temporisés.

3.6.1.1Temps associé à une place

Lorsqu'une place représente une activité, on lui associe une durée correspondant au temps nécessaire à la réalisation de cette activité. Pour modéliser ce comportement, la place peut être décomposée en une séquence :

place – transition – place.

- La première place représente l'activité en cours.
- La transition intermédiaire correspond à l'écoulement du temps.
- La seconde place indique une attente potentielle ou une **s**ynchronisation avec d'autres activités.

Pendant l'activité, le jeton est non disponible :

il ne peut pas activer d'autres transitions. À la fin de l'activité (lorsque le temps est écoulé), le jeton redevient disponible, permettant ainsi le franchissement de la transition suivante(figure 3.9).

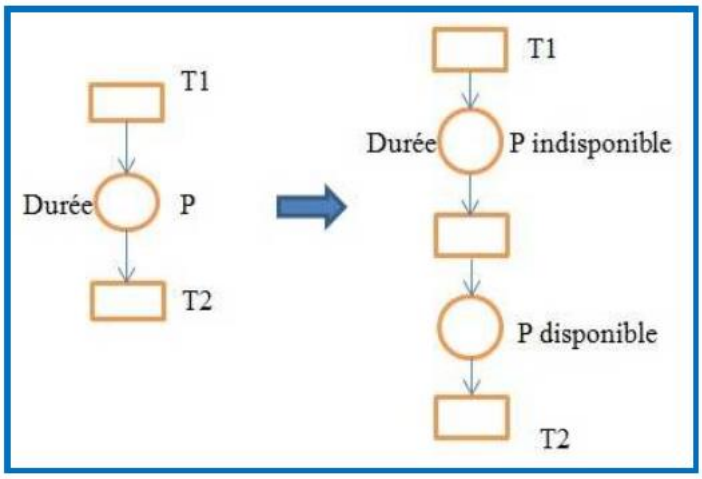

Figure 3.9: Temps Associé à une place.

3.6.1.2Temps associé à une transition

L'association temporelle à une transition suppose que celle-ci soit interprétée comme une **activité de durée finie**, et non comme un simple événement instantané. On la modélise alors par la séquence : transition – place – transition.

- La première transition représente le début instantané de l'activité (retrait des jetons).
- La place intermédiaire conserve l'information sur l'activité en cours.
- La seconde transition symbolise la fin de l'activité (réinjection des jetons dans les places de sortie).

Les jetons sont réservés dès le franchissement de la première transition : ils ne peuvent plus être utilisés ailleurs jusqu'à ce que la deuxième transition soit franchie, moment où ils sont libérés(figure 3.10).

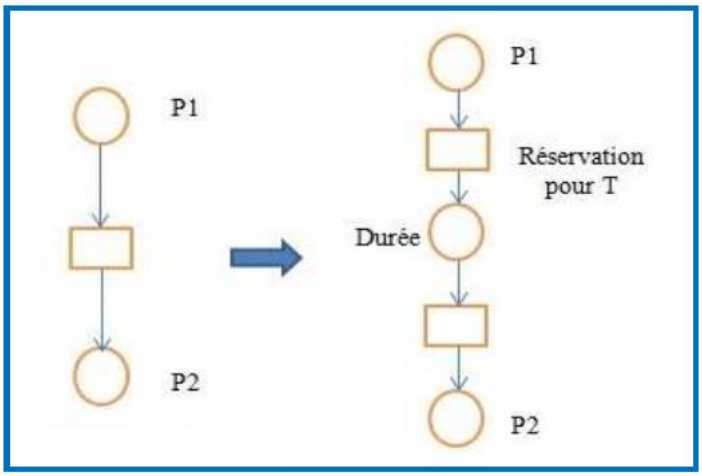

Figure 3.10: Temps Associé à une transition.

3.6.2. Les Réseaux de Pétri Stochastiques

Les Réseaux de Pétri stochastiques, introduits par Natkin [Natkin80] et Molloy [Molloy81], visent à intégrer la variabilité aléatoire des délais dans la modélisation, notamment dans les systèmes industriels.

Contrairement aux réseaux temporisés, les délais associés aux transitions sont aléatoires, et modélisés par des variables aléatoires. La loi exponentielle est la plus fréquemment utilisée, car elle permet d'associer le graphe des marquages à un processus de Markov homogène.

Ces réseaux sont largement utilisés dans les domaines de la fiabilité et de la sûreté de fonctionnement, où l'analyse probabiliste est essentielle(figure 3.11).

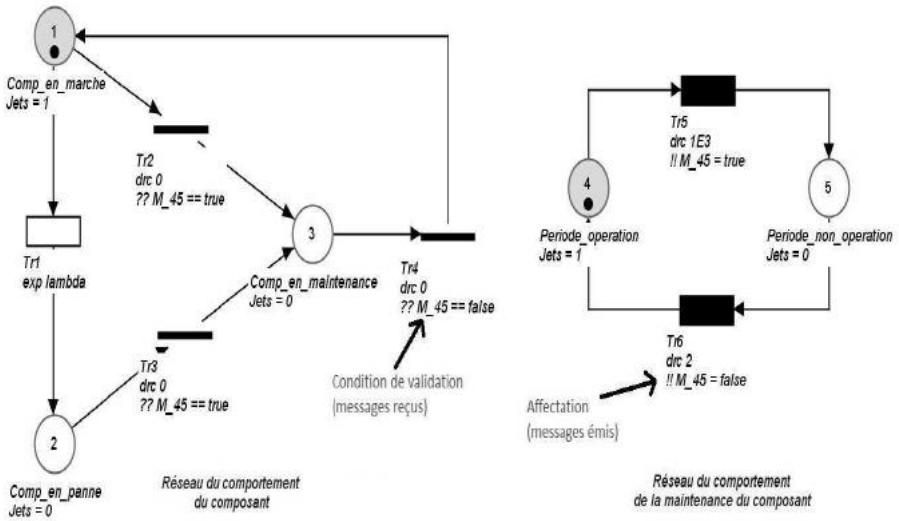

Figure 3.11: Réseaux de Pétri stochastiques.

3.6.3 Les Réseaux de Pétri Colorés

Les Réseaux de Pétri colorés sont une extension puissante des RDP classiques, conçue pour modéliser des systèmes de grande taille et complexes.

Leur particularité réside dans l'ajout de couleurs (ou identificateurs) aux jetons, ce qui permet de différencier les types d'informations transportées dans le réseau. Chaque jeton possède une couleur qui représente son identité ou sa nature.

Le franchissement des transitions dépend alors non seulement de la présence de jetons dans les places d'entrée, mais aussi de la correspondance entre les couleurs et les conditions définies par le modèle. Des fonctions de transition établissent les règles entre les couleurs des jetons en entrée et en sortie, offrant ainsi une plus grande expressivité dans la modélisation des systèmes(figure 3.12).

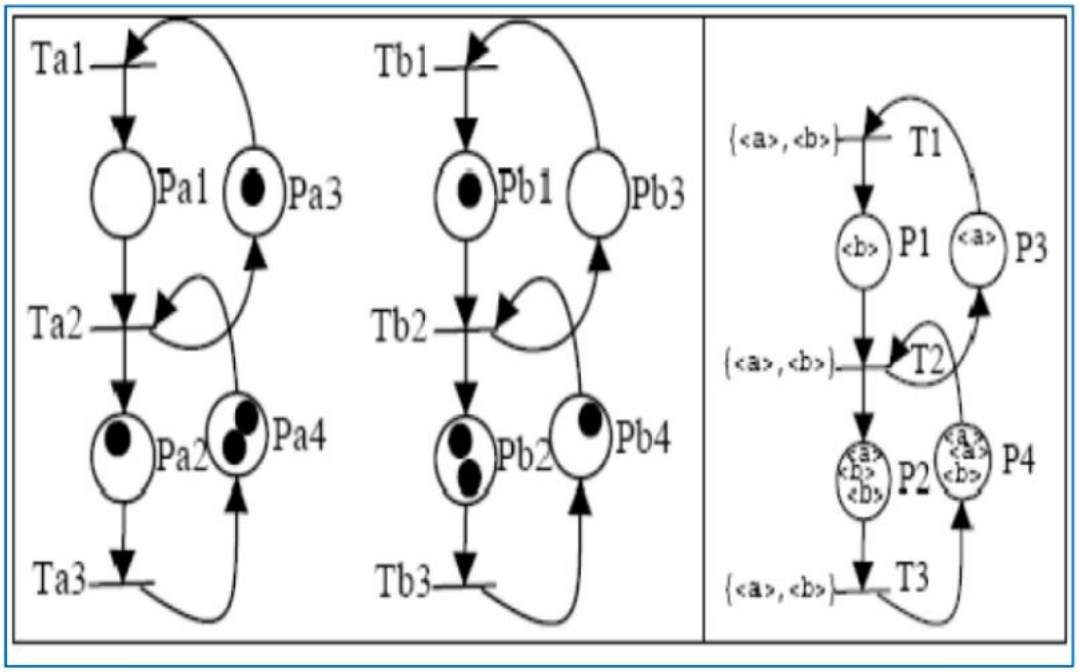

Figure 3.12 : RdP (gauche) et RdP coloré (droite)

3.7 Propriétés des Réseaux de Pétri

Les réseaux de Pétri possèdent plusieurs propriétés fondamentales permettant de juger de la qualité et du comportement dynamique du système modélisé. Parmi les plus importantes, on distingue la bornitude, la vivacité et l'absence de blocage.

3.7.1 RdP Borné

Un réseau de Pétri est dit *borné* s'il existe une limite supérieure au nombre de jetons que chaque place peut contenir, quelle que soit l'évolution du système.

- Une place PiP_iPi est dite bornée pour un marquage initial M0M_0M0 s'il n'existe aucun marquage accessible dans lequel le nombre de jetons dans PiP_iPi devient infini. Autrement dit, dans tous les marquages accessibles depuis M0M_0M0, le nombre de jetons de PiP_iPi reste fini.
- Un réseau de Pétri est donc borné pour un marquage initial M0M_0M0 si toutes ses places sont bornées pour ce marquage.
- Plus précisément, si pour tout marquage M∈∗M0M \in *M_0M∈∗M0 (ensemble des marquages accessibles depuis M0M_0M0), on a :

  M(Pi)≤KM(P_i) \leq KM(Pi)≤K

  avec K∈NK \in \mathbb{N}K∈N, alors la place PiP_iPi est dite K-bornée. Si cette condition est vérifiée pour toutes les places du réseau, alors le RdP est dit K-borné.

3.7.2 RdP Sauf (ou binaire)

Un réseau de Pétri est qualifié de sauf, également appelé binaire, lorsqu'il est 1-borné, c'est-à-dire que chaque place ne peut contenir au maximum qu'un seul jeton à tout instant, pour tous les marquages accessibles. Ce type de réseau est particulièrement utile pour modéliser des systèmes à états binaires (vrai/faux, actif/inactif, etc.).

3.7.3 Vivacité

La vivacité est une propriété comportementale essentielle qui garantit que le système reste actif.

- Une transition TjT_jTj est dite vivante pour un marquage initial M0M_0M0 si, pour tout marquage accessible Mi∈∗M0M_i \in *M_0Mi∈∗M0, il existe une séquence de transitions partant de MiM_iMi et qui permet ultérieurement le franchissement de TjT_jTj.
- En d'autres termes, la transition TjT_jTj ne sera jamais définitivement bloquée, peu importe l'évolution du système. Cela assure qu'une fonctionnalité ou une action représentée par cette transition pourra toujours se produire à terme.

3.7.4 Blocage

Le blocage représente une situation indésirable où le système est incapable de continuer à évoluer.

- Un blocage, également appelé état puits, correspond à un marquage dans lequel aucune transition n'est activée. Le système est alors dans un état d'inertie complète.
- Un RdP est dit sans blocage pour un marquage initial M0M_0M0 si tous les marquages accessibles Mi∈∗M0M_i \in *M_0Mi∈∗M0 ne sont pas des états de blocage.

.

3.8 Méthodes d'analyse des réseaux de Pétri

La modélisation d'un système à l'aide des réseaux de Pétri ne se limite pas à une simple représentation graphique ; elle constitue également une base solide pour une analyse rigoureuse et formelle des propriétés structurelles et comportementales du système. Grâce à leur fondement mathématique et leur expressivité, les réseaux de Pétri offrent un cadre particulièrement adapté à la description, la vérification et l'évaluation des systèmes à événements discrets.

Plusieurs méthodes d'analyse sont utilisées pour exploiter les capacités des réseaux de Pétri, notamment (figure 3.13) :

• Le graphe de marquages

Cette méthode consiste à générer un graphe décrivant l'ensemble des marquages accessibles à partir du marquage initial du réseau. Chaque nœud représente un marquage, et chaque arc une transition activée. Ce graphe permet de retracer toutes les évolutions possibles du système modélisé. Son analyse offre la possibilité de vérifier des propriétés essentielles telles que :

- **l'accessibilité** d'un état donné .
- **la vivacité** (absence de blocages) .
- **la bornitude** (limitation du nombre de jetons).
- **l'atteignabilité** d'une configuration spécifique du système.

Cette méthode, bien que très informative, peut devenir rapidement coûteuse en termes de complexité computationnelle, notamment lorsque le nombre d'états est très élevé (problème d'explosion d'état).

• L'équation de matrice

L'approche matricielle repose sur la représentation du réseau sous forme de matrices, en particulier la matrice d'incidence, qui traduit les relations entre les places et les transitions. En utilisant les outils de l'algèbre linéaire, il est possible d'étudier des invariants de place (quantités conservées dans le système) et des invariants de transition (séquences de transitions qui laissent le système inchangé globalement). Ces invariants fournissent des indications cruciales sur les comportements cycliques ou stables du système, sans qu'il soit nécessaire de parcourir tout l'espace d'états.

• La réduction des réseaux de Pétri

Face à la complexité croissante de certains réseaux, notamment ceux représentant des systèmes industriels ou informatiques, l'analyse directe peut s'avérer inapplicable. La **réduction** consiste alors à transformer un réseau en une version équivalente, mais plus simple, à l'aide de règles de simplification (suppression de transitions ou de places inutiles, fusion de structures similaires, etc.).

Cette méthode permet de diminuer la taille du modèle sans altérer ses propriétés fondamentales,facilitant ainsi les analyses ultérieures.

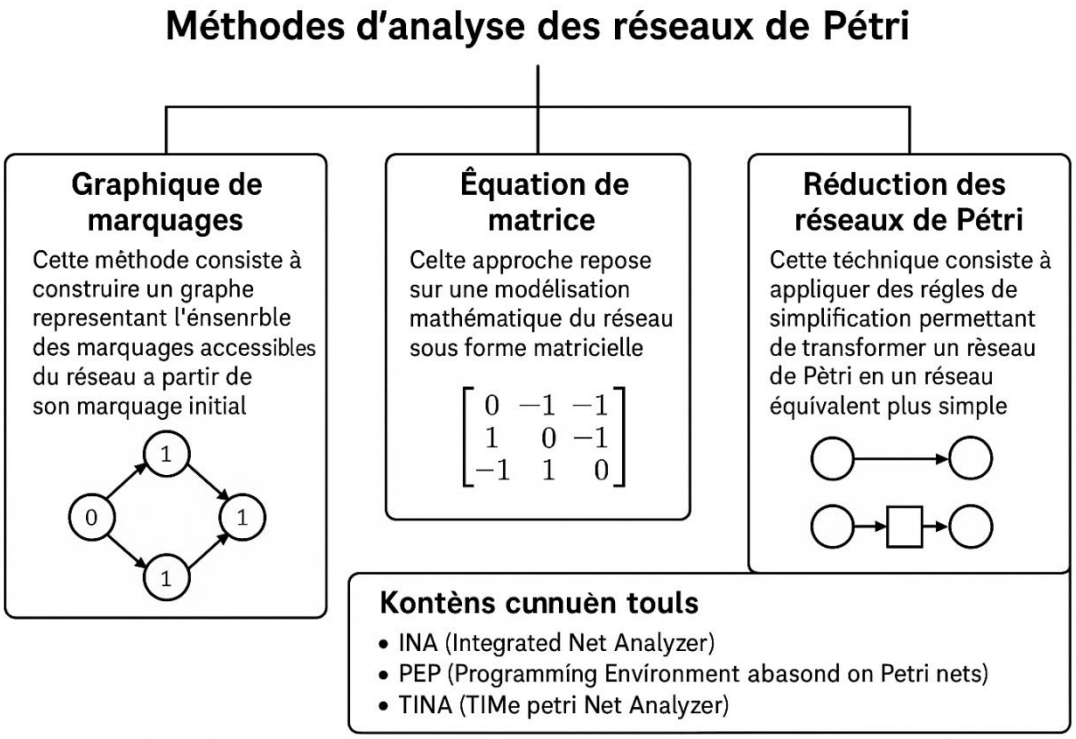

Figure 3.13 : méthodes d'analyse pour exploiter le capacité de rdp

3.9 Outils informatiques d'analyse

La richesse des méthodes analytiques s'accompagne d'un ensemble d'outils logiciels puissants permettant leur mise en œuvre pratique. Ces outils facilitent la simulation, la vérification formelle et l'analyse des propriétés dynamiques des modèles. Parmi les plus connus, on peut citer :

- INA (Integrated Net Analyzer) : spécialisé dans l'analyse des invariants et des propriétés structurelles.
- PEP (Programming Environment based on Petri nets) : environnement de développement orienté modélisation, simulation et vérification.
- TINA (TIme petri Net Analyzer) : adapté aux réseaux temporels, il permet d'intégrer la dimension temps dans les analyses.

Grâce à ces instruments, les ingénieurs et chercheurs peuvent modéliser, tester et valider des systèmes complexes avec précision et fiabilité, ce qui rend les réseaux de Pétri incontournables dans les domaines tels que l'automatisation industrielle, les systèmes embarqués, ou encore les protocoles de communication.

3.10 Conclusion

Les Réseaux de Pétri s'imposent comme un outil de modélisation puissant et rigoureux pour l'analyse des systèmes à événements discrets. Grâce à leur double nature — graphique et mathématique — ils permettent non seulement une représentation claire des structures et des comportements d'un système, mais aussi une vérification formelle de ses propriétés essentielles telles que la bornitude, la vivacité ou l'absence de blocage.

Les différentes extensions des RdP, comme les réseaux temporisés, stochastiques ou colorés, offrent une souplesse accrue pour s'adapter à des systèmes complexes, variés et dynamiques. De plus, l'existence de méthodes d'analyse bien établies et d'outils informatiques spécialisés renforce leur pertinence dans les domaines industriels, informatiques ou embarqués.

Ainsi, les Réseaux de Pétri constituent une base solide pour la spécification, la simulation et l'évaluation de systèmes, en garantissant une meilleure maîtrise de leur conception et de leur fonctionnement.

Chapitre 4 : Renew – Conception et Simulation du Système Intelligent de Parking

4.1 Introduction

Dans ce chapitre, nous décrivons la simulation du système intelligent de gestion de parking à l'aide de l'outil Renew (Reference Net Workshop), un simulateur basé sur les réseaux de Petri orientés objets. Le système a été conçu selon une architecture multi-agents, où chaque entité (voiture, agent de gestion, parking) est représentée par un agent encapsulé dans un *net*. Nous allons ici présenter le déroulement complet d'une exécution type du système, appuyée par une capture d'écran annotée, tout en détaillant les interactions, les communications et les mécanismes internes mis en œuvre.

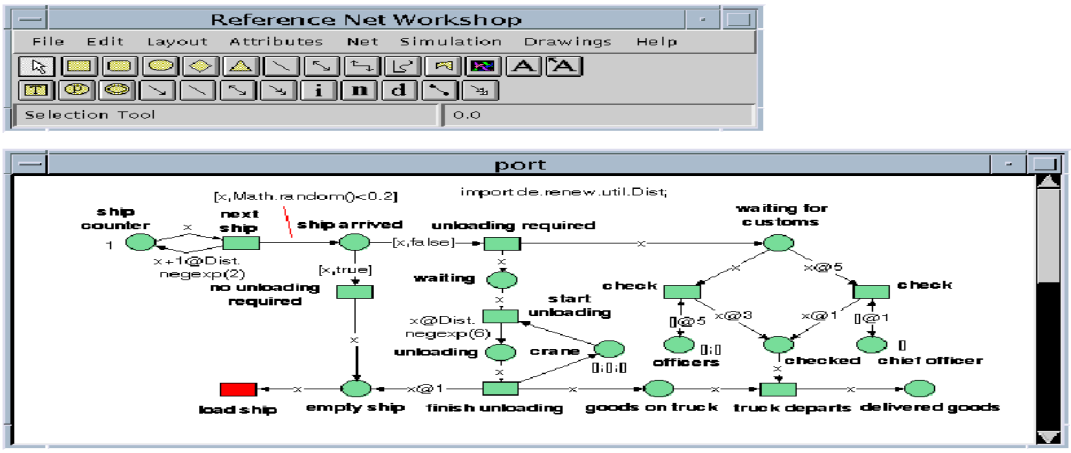

Figure 4.1 : Renew interface

4.2 Concept de Nets Within Nets

Notre système de parking intelligent s'appuie sur le concept des *Nets Within Nets*, ou réseaux imbriqués, une approche puissante offerte par Renew. Plutôt que d'utiliser de simples jetons passifs, chaque entité importante du système — comme les voitures ou les agents est elle-même un réseau de Petri actif, vivant à l'intérieur du réseau principal.

Concrètement, cela signifie qu'une voiture n'est pas juste une donnée qui se déplace, mais un petit programme autonome avec son propre comportement, capable de communiquer, de prendre des décisions et de réagir à l'environnement. Ce principe donne au modèle une grande souplesse : chaque composant garde sa logique interne, tout en interagissant avec les autres.

Grâce à cette structure imbriquée, nous avons pu reproduire de manière réaliste les échanges entre voitures et parkings, gérer les priorités, simuler des délais, et rendre le tout facilement extensible pour de futures évolutions du système.

4.3 Lancement de la simulation

L'exécution débute par l'activation du net principal nommé **Parkings**, via le raccourci Ctrl + I dans Renew. Cette action initialise le système, notamment le réseau responsable de la gestion globale et de la génération des entités du système (agents et voitures) figure (4.2).

Figure 4.2 : Simulation Parking intelligent dans Renew

4.4 Création de l'agent gestionnaire

La première transition manuellement activée est p1:new Ag(this, "3;5;7;9"), qui crée un nouvel agent gestionnaire (Ag[n]). Ce dernier est responsable de la coordination avec les parkings. Le paramètre "3;5;7;9" représente la connaissance initiale de l'agent, c'est-à-dire le nombre de places disponibles dans les quatre parkings (3 places pour le premier, 5 pour le deuxième, etc.). Cette information est stockée dans sa knowledge base, ce qui lui permet de raisonner et de prendre des décisions selon l'état de disponibilité.

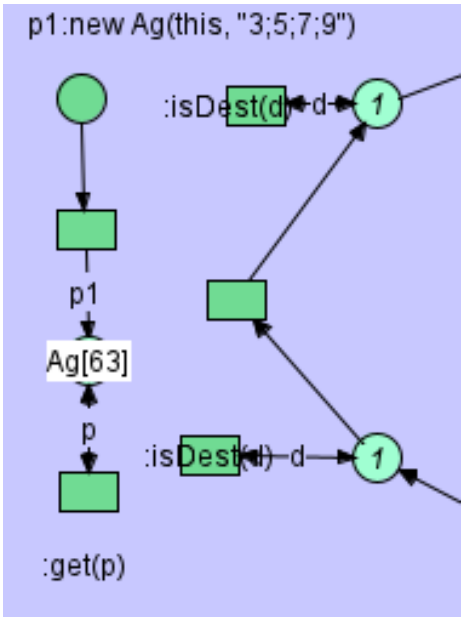

Figure 4.3 : Géneration d'un Agent

4.5 Génération d'une voiture (Car)

Une fois l'agent Ag[n] créé, nous activons la transition v:new Car(this, ag) pour créer un agent voiture. Cette entité reçoit une référence directe vers l'agent Ag[n] à travers this:get(ag). Cette étape simule l'arrivée d'un véhicule qui va entrer en interaction avec le système pour tenter de se garer.

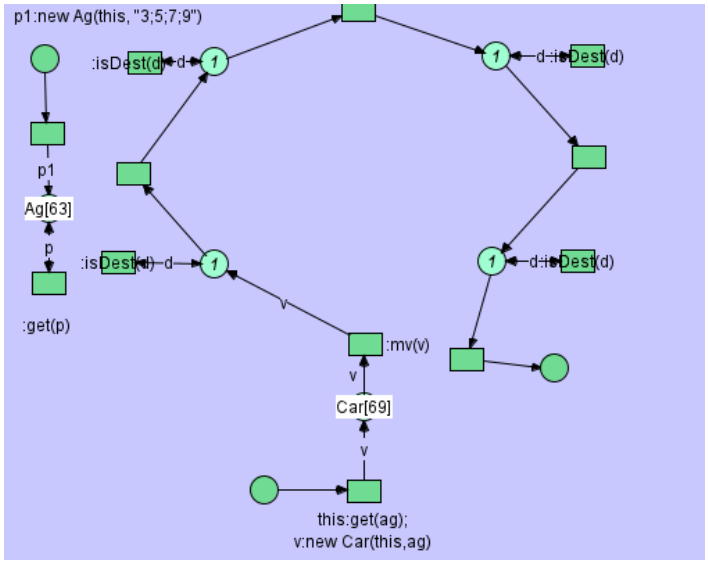

Figure 4.4 : Géneration d'une Voiture (Car)

L'apparition d'une voiture provoque l'ouverture automatique de deux nouveaux réseaux :

- Car[n] : l'agent voiture en lui-même
- PrCar[n] : son protocole de communication

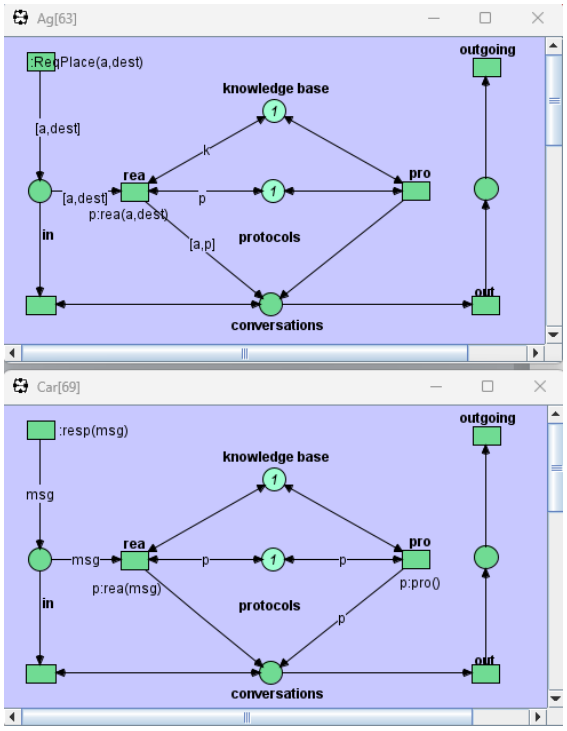

Figure 4.5 : Réseaux Agent et Voiture

Simultanément, le protocole PrAg[n] est aussi actif pour gérer les échanges côté agent gestionnaire.

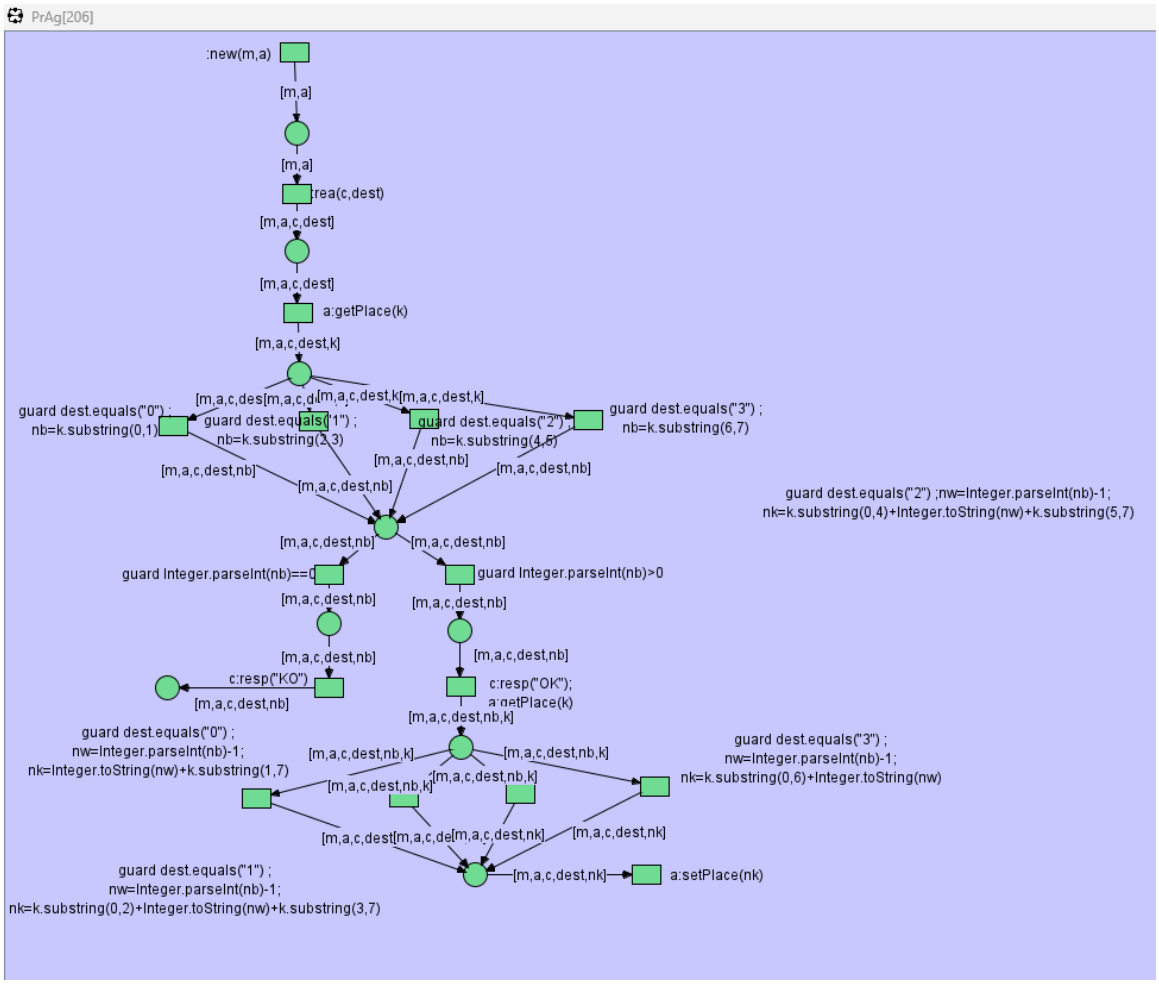

Figure 4.6 : Protocol d'Agent

4.6 Comportement proactif de la voiture

Une fois la voiture en place, on déclenche la transition pro dans le net Car[n]. Celle-ci représente le comportement proactif de l'agent voiture. La voiture initie une demande en utilisant le protocole PrCar[n] :

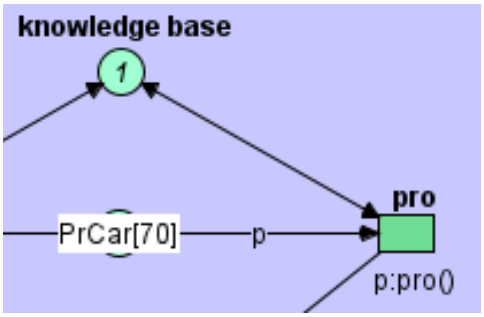

Figure 4.7 : Comportement proactif

- Elle construit une requête vers un parking spécifique, ici sous la forme p:ReqPlace(a, dd="n"), où dd est un identifiant correspondant au numéro du parking ciblé.

- Cette requête contient les références de la voiture (m), de l'agent (a), et du parking visé (dd), formant ainsi un triple [m,a,dd].

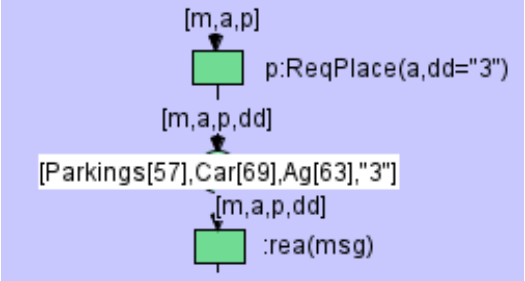

Figure 4.8 : Création d'une Requéte d'une place

4.7 Traitement par l'agent Ag

L'agent gestionnaire reçoit la requête via la transition rea dans Ag[n], puis la transmet au protocole PrAg[n], qui applique une logique de sélection du parking.

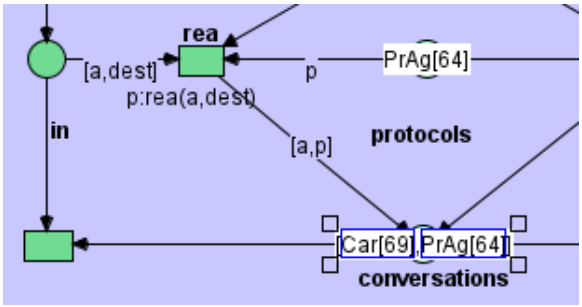

Figure 4.9 : Réaction d'Agent vers la requéte

Le protocole PrAg[n] appelle ensuite la transition a:getPlace(k) pour évaluer le nombre de places disponibles. À ce moment, le système effectue une analyse conditionnelle à l'aide de *guards* (gardes logiques) comme :

- guard dest.equals("0"), guard dest.equals("1"), etc., pour identifier le bon parking.
- Puis, en extrayant le nombre de places disponibles à partir de la chaîne "3;5;7;9" grâce à des sous-chaînes (substring), par exemple k.substring(0,1) pour le parking 0.

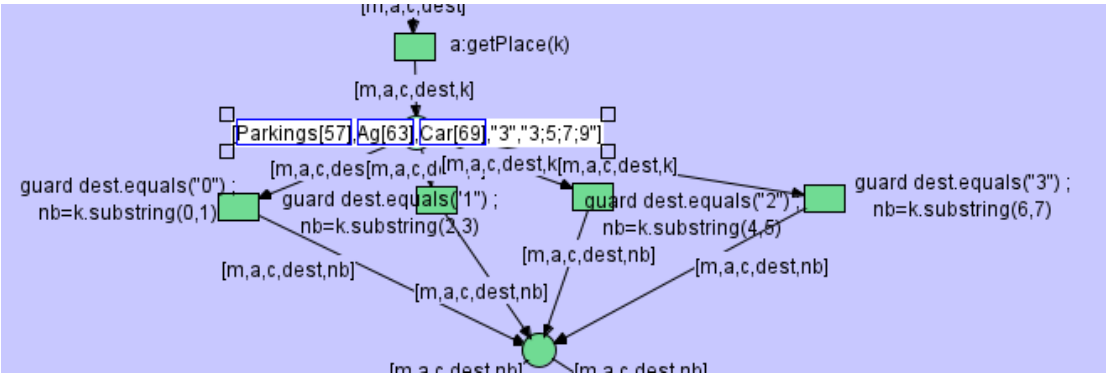

Figure 4.10 : Collecte d'information Concernant nombre de places du parking choisi

Chaque chemin mène à une transition avec un garde supplémentaire :

- Integer.parseInt(nb) > 0 : si des places sont disponibles
- nb == 0 : si le parking est plein

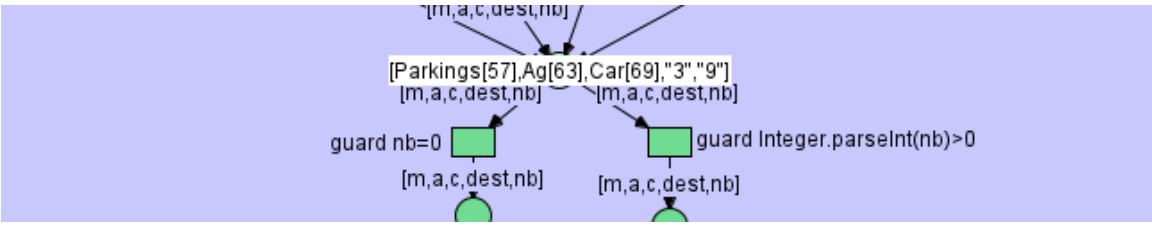

Figure 4.11 : Confirmation de disponibilité des places libres

4.8 Réponse à la voiture et stationnement

Si une place est disponible, la transition déclenche c:resp("OK"), ce qui notifie la voiture que sa demande est acceptée. Le protocole PrCar[n] traite ensuite ce message à travers :

- m_rea(msg) → guard msg.equals("OK")

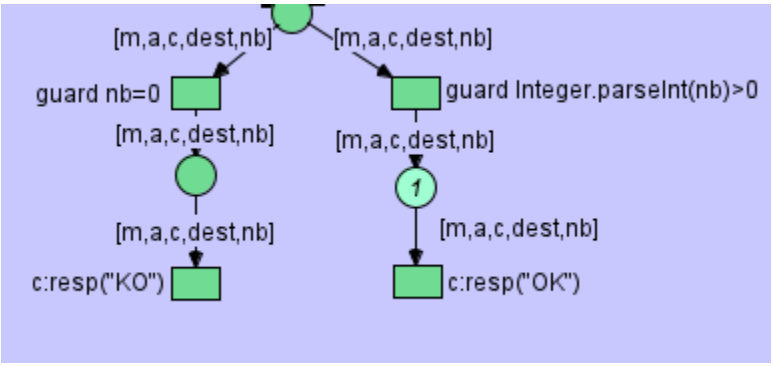

Figure 4.12 : Permission d'entrée

Enfin, du côté des modules Parking, une nouvelle transition isDest(a,d) a été ajoutée. Elle permet à un parking de comparer dynamiquement la position actuelle d'une voiture à la destination souhaitée. Cette vérification fonctionne en coordination avec la boucle dans PrCar, jusqu'à ce que la correspondance soit atteinte.

Une fois la réponse reçue, si le message est "OK", la voiture commence un processus de vérification de destination dynamique. Un nouveau loop de comparaison a été introduit : la voiture compare en continu sa destination souhaitée dd avec l'emplacement actuel d du parking où elle se trouve. Si les deux ne correspondent pas (!dd.equals(d)), la voiture continue de se déplacer visuellement via m:mv(a) jusqu'à atteindre la bonne destination (dd.equals(d)), où elle s'arrête.

Cela permet une simulation réaliste du mouvement de la voiture vers sa destination cible.

Cela renforce la cohérence entre la position réelle de la voiture et son objectif final, améliorant ainsi la crédibilité de la simulation.

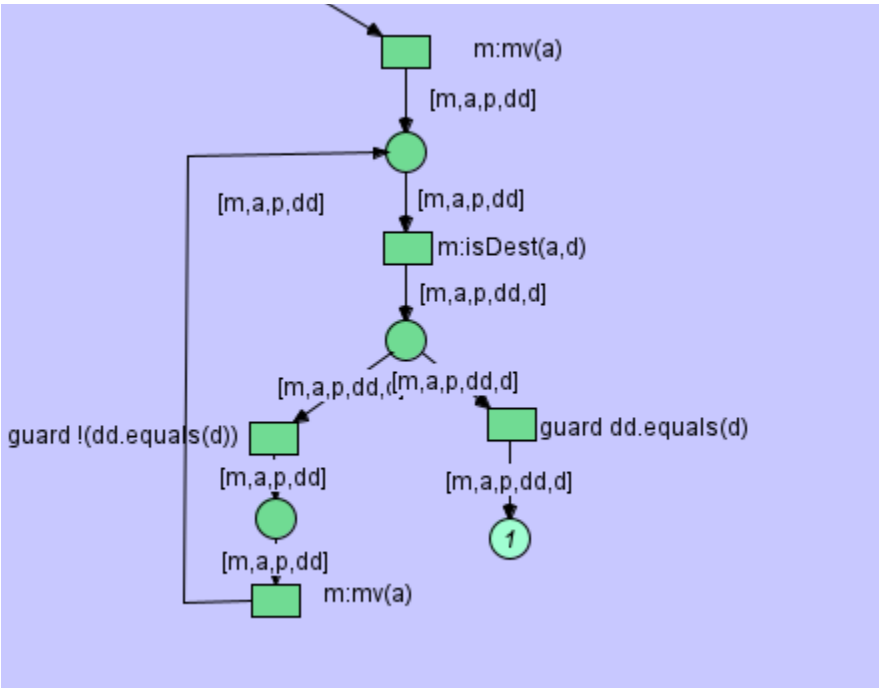

Figure 4.13 : Boucle de choix de la destination desirée

- Ensuite la voiture effectue la transition m:mv(a) pour se déplacer vers le parking assigné.

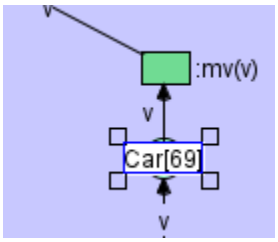

Figure 4.14 : La Voiture vers la destination

Ce mouvement est visualisé dans Renew par le marquage du token dans la place correspondante au parking (dans notre cas, parking 3 représenté par "3").

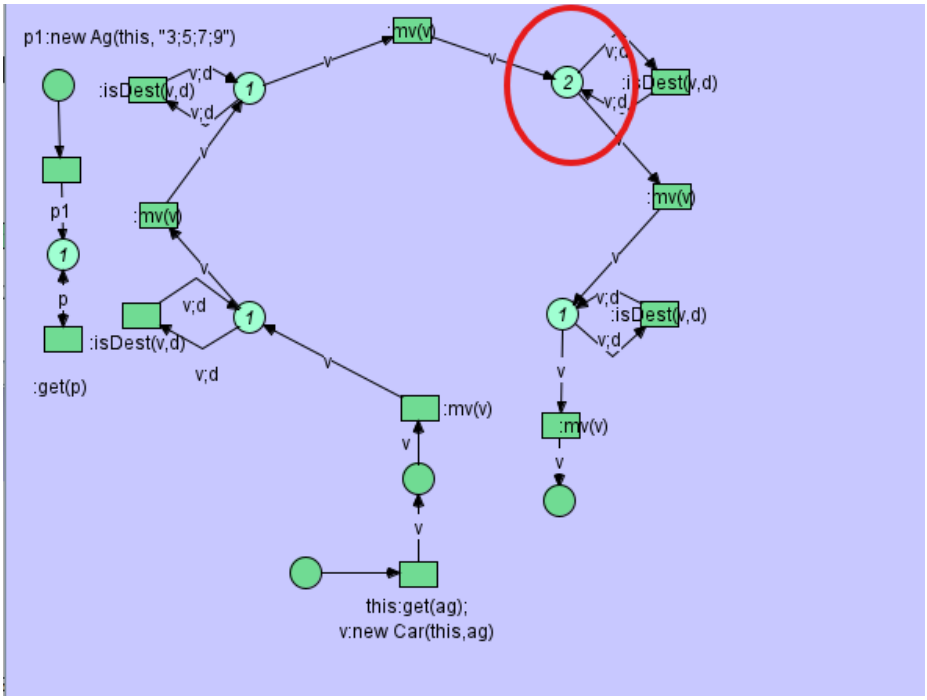

Figure 4.15 : Parking de la voiture

Le nombre de places disponibles est décrémenté automatiquement dans la connaissance de l'agent.

Après avoir envoyé une réponse "OK" à une voiture, une nouvelle étape a été ajoutée pour **mettre à jour dynamiquement la base de connaissance**. L'agent détecte quel parking est concerné (0, 1, 2 ou 3), extrait la bonne sous-chaîne de caractères, soustrait une place disponible (nbk = Integer.parseInt(nb) - 1) et **reconstruit la chaîne** pour représenter les nouvelles disponibilités. Ensuite, il applique a:setPlace(nk) pour sauvegarder cette nouvelle information.

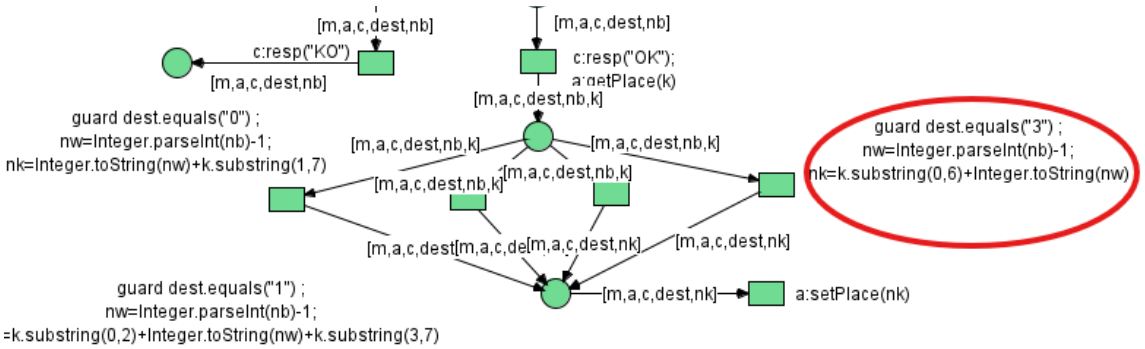

Figure 4.16 : Condition de substraction d'une place

Cette opération est répétée via quatre gardes différentes, chacune correspondant à un parking possible.

L'agent principal conserve la **connaissance globale du système**. Grâce à la fonction setPlace(nk) nouvellement ajoutée, il est maintenant capable de modifier sa base de connaissance après chaque décision, garantissant ainsi que toutes les informations restent à jour. Cela permet à l'ensemble du système d'évoluer dynamiquement selon l'occupation réelle des parkings.

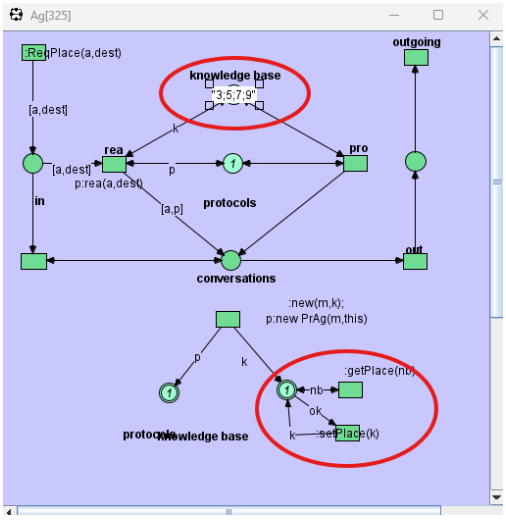

Figure 4.17 : Avant la mise a jour de knowledge base de l'agent Ag

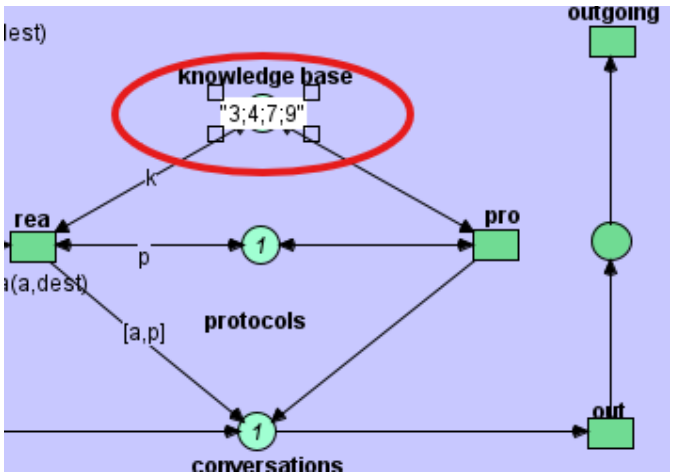

Figure 4.18 : Après la mise a jour de knowledge base

4.9 Observation dynamique

Ce processus peut être répété pour créer d'autres voitures et agents, permettant une simulation plus large avec saturation progressive des parkings.

Mise en file lors de saturation

Quand une voiture se voit répondre "KO" (parking plein), elle ne se contente pas de quitter le système. Le modèle Renew peut être étendu pour que cette voiture reste dans une file d'attente interne (non visible directement dans le modèle de base mais simulable) en attente de disponibilité dans le parking cible.

Cela correspondrait, dans une version enrichie, à un marquage de la transition ou à un cycle de requêtes automatiques. Dans notre version actuelle, l'agent Car[n] peut relancer manuellement une nouvelle requête vers le même agent Ag[n], ou vers un autre agent en fonction d'une stratégie de reroutage (par exemple en demandant Ag de reconsidérer l'option de stationnement avec une autre valeur de dd une autre destination).

Réattribution dynamique

Dès réception d'un message "OK", la voiture effectue la transition vers la place de parking. Cela est visible dans le modèle Renew à travers la transition m:mv(a) dans le protocole PrCar[n], qui symbolise le mouvement physique de la voiture vers le parking. L'état du parking est ensuite mis à jour à nouveau (décrémenté), et le cycle continue.

Cette boucle d'attente-réattribution permet de maintenir un flux régulé et fluide de stationnement, même dans des conditions de saturation . C'est ce mécanisme qui garantit le caractère réactif et adaptatif du système.

4.10 Conclusion du Chapitre

Ce chapitre a permis de documenter en détail l'exécution concrète du système de parking intelligent modélisé avec Renew. À travers l'analyse pas-à-pas de l'ensemble des agents, protocoles, transitions et échanges de messages, nous avons montré comment un environnement distribué, réactif et autonome pouvait être simulé efficacement à l'aide des *high-level Petri nets*.

Chapitre 5 : Implémentation

5.1 Introduction

L'implémentation du système de parking intelligent repose sur l'intégration cohérente de plusieurs technologies modernes et puissantes, choisies avec soin pour répondre aux exigences du projet. Ce chapitre détaille le processus de mise en œuvre, en expliquant les raisons ayant motivé le choix des outils, les caractéristiques techniques des technologies adoptées, ainsi que le fonctionnement interne du système, aussi bien du point de vue logique que de celui de l'expérience utilisateur.

5.2 Choix des technologies

Le développement du système s'est appuyé principalement sur quatre éléments technologiques majeurs : **JADE**, **JavaFX**, **FXML**, et l'environnement de développement **IntelliJ IDEA**. Chacune de ces technologies joue un rôle essentiel dans l'architecture globale du projet et a été sélectionnée pour ses qualités spécifiques, sa compatibilité avec les autres outils, et sa capacité à modéliser un système intelligent en temps réel.

5.2.1 JADE : Java Agent DEvelopment Framework

JADE (Java Agent DEvelopment Framework) est un framework open source conforme à la spécification FIPA (Foundation for Intelligent Physical Agents). Il permet de concevoir et simuler des systèmes multi-agents en Java, de façon standardisée et modulaire. Dans notre contexte, JADE a été utilisé pour gérer la logique des agents intelligents, représentant à la fois les véhicules et les parkings(figure 5.1).

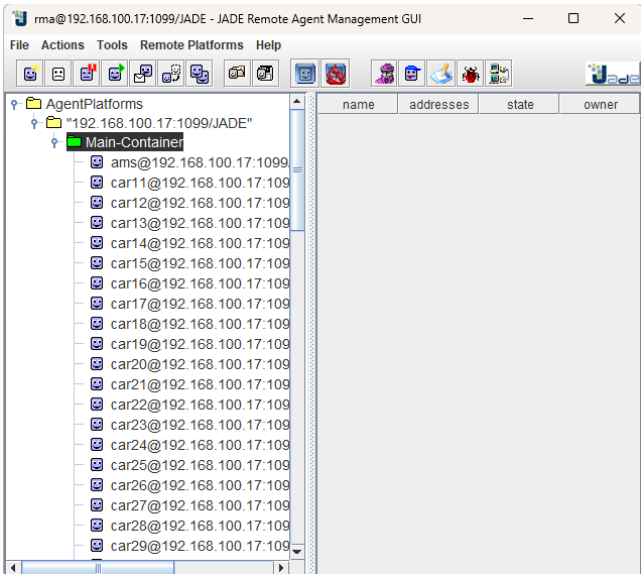

Figure 5.1 : Interface du Jade

Ce choix s'est imposé naturellement, compte tenu de la nature distribuée et autonome des entités dans notre système. Chaque **voiture** est modélisée comme un agent capable de prendre des décisions (telles que choisir un parking ou attendre en file), tandis que chaque **parking** est également un agent qui gère ses ressources (places disponibles) et sa communication avec les voitures. Grâce à JADE, il a été possible de simuler des comportements réalistes, des négociations, et des files d'attente basées sur des critères de priorité.

JADE offre aussi une interface graphique(voir figure 5.2) optionnelle (sniffer) qui permet de visualiser les messages échangés entre agents, ce qui facilite énormément la phase de test et de débogage du système multi-agents.

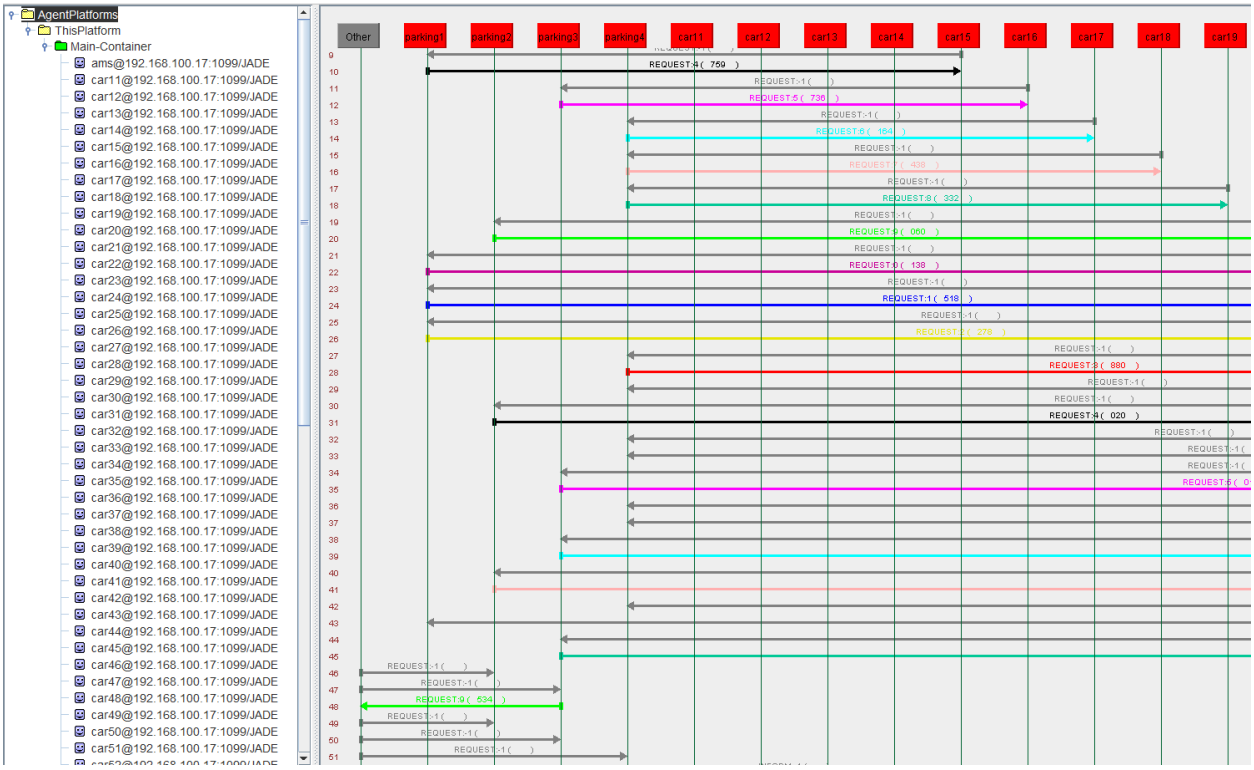

Figure 5.2 : Interface agent Sniffer du Parking Intelligent

5.2.2 JavaFX : Interface graphique moderne et dynamique

JavaFX est le successeur de Swing et l'une des bibliothèques les plus puissantes pour le développement d'interfaces utilisateur graphiques (GUI) en Java. Son adoption dans ce projet a été motivée par plusieurs critères clés : sa capacité à créer des interfaces modernes, responsives et esthétiques ; sa compatibilité directe avec le langage Java ; et sa facilité d'intégration avec FXML, qui sépare clairement la logique de présentation et le code métier.

Dans notre application, JavaFX a été utilisé pour visualiser en temps réel :

- l'état de chaque parking (places disponibles / occupées),
- la liste des voitures actives dans le système,
- la file d'attente en cas de saturation,
- et le déroulement de la simulation (via un bouton déclencheur).

L'interaction entre JADE (qui gère la logique des agents) et JavaFX (qui gère l'affichage utilisateur) (voir figure 5.3) a nécessité un pont de communication, que nous avons réalisé à travers une classe dédiée appelée UIBridge. Celle-ci permet de déclencher des mises à jour de l'interface à chaque changement d'état dans les agents, en s'appuyant sur la classe Platform.runLater() de JavaFX, qui garantit l'exécution sécurisée de code graphique depuis un thread non-JavaFX.

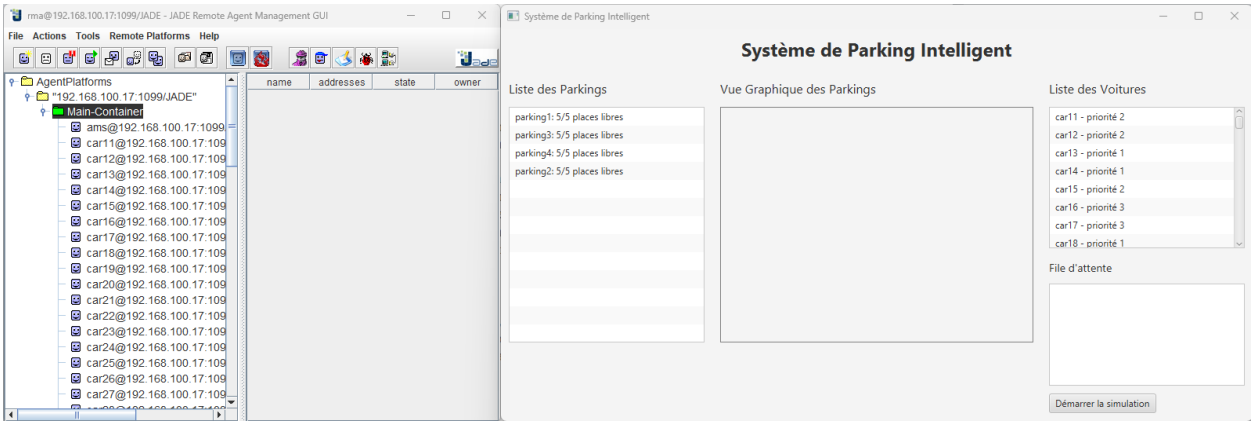

Figure 5.3 : Interaction JADE JavaFX

5.2.3 FXML : Définition déclarative de l'interface

FXML est un langage basé sur XML, utilisé pour décrire l'interface utilisateur d'une application JavaFX de manière déclarative. Il sépare les préoccupations de l'interface et de la logique métier, ce qui permet une meilleure maintenabilité et lisibilité du code. En d'autres termes, l'interface graphique est décrite dans un fichier .fxml, tandis que sa logique est implémentée dans une classe contrôleur en Java.

Dans ce projet, le fichier MainView.fxml structure toute l'interface utilisateur : affichage des listes, grille graphique des parkings, file d'attente, et boutons de filtrage. Le contrôleur associé, MainController.java, est responsable de l'interaction avec la logique du système et les mises à jour dynamiques de l'affichage.

Cette approche a permis un développement plus propre et organisé, où les rôles de chaque composant sont clairement définis.

5.2.4 IntelliJ IDEA : Environnement de développement

Le choix d'**IntelliJ IDEA** comme environnement de développement intégré (IDE) a été déterminant pour la réussite de l'implémentation. Cet IDE est reconnu pour sa robustesse, son support étendu de Java, sa complétion de code intelligente, sa gestion avancée des projets, et sa compatibilité avec des outils externes comme JADE ou JavaFX (figure5.4).

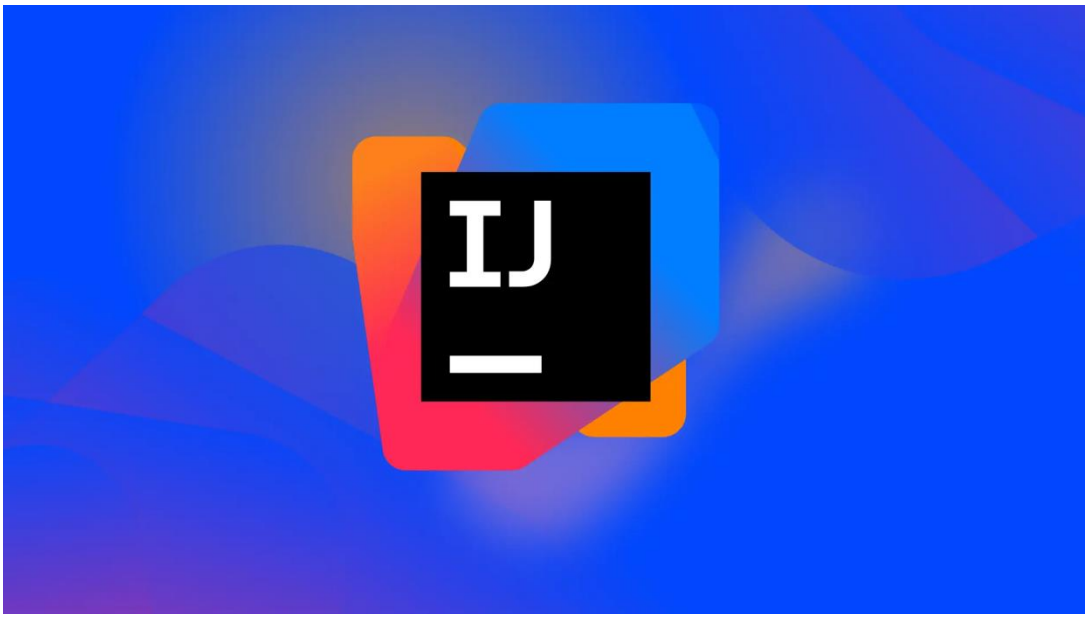

Figure 5.4 : Intellij IDEA IDE

L'utilisation d'IntelliJ a notamment permis :

- une gestion efficace des dépendances (même sans Maven),
- une exécution simplifiée du programme principal,
- et une intégration fluide avec les ressources JavaFX et FXML.

Le débogage y est également très intuitif, grâce à un système de points d'arrêt graphique, des inspections de variables en temps réel, et un navigateur de classes efficace.

5.3 Architecture du système

L'architecture de notre système de parking intelligent repose sur une approche distribuée, multi-agents et orientée événements. Chaque composant du système joue un rôle bien défini et communique avec les autres par le biais de messages asynchrones. Cette structure favorise la modularité, la scalabilité et la capacité à réagir dynamiquement aux changements dans l'environnement.

Le système est constitué de deux grandes couches :

la couche logique, gérée par JADE (Java Agent DEvelopment Framework), et la couche de présentation, développée en JavaFX. Cette séparation claire entre la logique métier (agents, décisions, coordination) et l'interface utilisateur (affichage des parkings, des voitures, des files d'attente) permet une maintenance facilitée et une évolution plus aisée du système.

5.3.1 Couche agentielle

La couche agentielle repose sur JADE, qui permet de créer des agents autonomes capables de percevoir leur environnement, de prendre des décisions et d'interagir entre eux. Deux types d'agents sont modélisés :

- **Les agents Parking** : chaque parking est représenté par un agent distinct qui gère un nombre donné de places. Cet agent reçoit les demandes de réservation, vérifie la disponibilité et décide d'accepter ou de rejeter une demande. En cas d'indisponibilité, il ajoute les voitures dans une file d'attente gérée par priorité.
- **Les agents Car** : ces agents représentent les voitures. Chaque voiture tente de se garer en envoyant une demande à tous les parkings. Si un parking accepte, la voiture s'y gare virtuellement pour une durée simulée, puis libère la place.

Ces interactions sont implémentées via des messages ACL (Agent Communication Language), un format standardisé de communication entre agents dans JADE. L'utilisation de JADE permet d'implémenter aisément des comportements cycliques, des stratégies de décision, et une communication asynchrone fiable.

5.3.2 Couche graphique

La couche graphique est implémentée en JavaFX, un framework moderne pour le développement d'interfaces utilisateur en Java. JavaFX offre de nombreux avantages par rapport à l'ancien système Swing, notamment :

- Une meilleure séparation entre la logique et la vue, grâce au support du langage FXML.
- Une grande souplesse pour la création d'interfaces dynamiques, interactives et modernes.
- Une gestion native des animations, transitions, et composants graphiques avancés.
- Une intégration facilitée avec les API Java, ce qui permet une synchronisation directe avec le comportement des agents JADE.

L'interface permet de visualiser en temps réel :

- L'état de chaque parking (nombre de places libres).
- La liste des voitures générées avec leur priorité.
- La file d'attente des voitures en attente.
- La vue graphique des parkings sous forme de grille.

La synchronisation entre les agents (JADE) et l'interface (JavaFX) est assurée via une classe passerelle appelée UIBridge. Elle permet aux agents de mettre à jour l'interface en appelant des méthodes JavaFX sur le thread graphique de manière sécurisée.

5.4 Communication entre agents et gestion des priorités

Dans notre système de parking intelligent, la communication inter-agents constitue l'un des piliers fondamentaux du fonctionnement. Elle est rendue possible grâce au langage ACL (Agent Communication Language) proposé par JADE, qui fournit un modèle de messagerie structuré permettant aux agents de dialoguer de manière autonome, sécurisée et asynchrone.

Chaque agent Car, après sa création, initie une requête de stationnement en envoyant un message à tous les agents Parking connus dans la plateforme. Le message contient des informations essentielles telles que le nom de la voiture et sa priorité, encodées dans un format simple mais lisible, comme par exemple :

car3:4, signifiant que la voiture nommée *car3* souhaite une place avec une priorité de 4 sur 10.

5.4.1 Format et traitement des messages

Les messages ACL de JADE contiennent plusieurs champs, dont les plus importants dans notre système sont :

- **Performative** : indique le type du message (ici, toujours REQUEST pour une demande ou INFORM pour une réponse).
- **Sender / Receiver** : identifie l'expéditeur et le destinataire du message.
- **Content** : contient les données spécifiques de la demande.

Un exemple de traitement typique est le suivant :

1. L'agent Car envoie une requête de réservation à tous les parkings.
2. Chaque agent Parking qui reçoit cette requête vérifie s'il dispose de places libres.
3. S'il y a une place disponible, il répond avec ACCEPTED.
4. Si aucune place n'est libre, le ParkingAgent place la voiture dans une file d'attente interne, triée par ordre de priorité croissante.

Ce mécanisme garantit que les voitures ayant une priorité plus élevée sont traitées avant celles de priorité inférieure.

5.4.2 La file d'attente prioritaire

L'un des éléments les plus innovants de notre architecture est l'intégration d'une **file d'attente avec gestion des priorités** (figure 5.5) dans chaque agent Parking. Cette file, implémentée comme une PriorityQueue en Java

permet de trier automatiquement les voitures selon leur priorité :

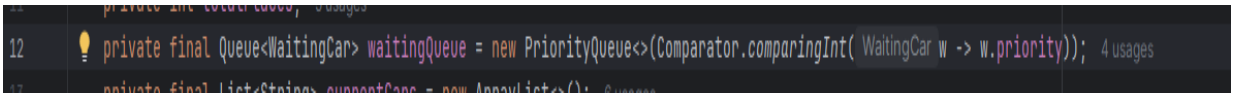

Figure 5.5 : Gestion Priorité

Ce choix technique permet une insertion et un traitement efficaces, avec une complexité logarithmique, tout en garantissant que la voiture ayant la priorité la plus élevée sera toujours traitée en premier dès qu'une place se libère.

Quand une place se libère dans un parking (après que le délai de stationnement simulé soit écoulé), l'agent Parking vérifie s'il y a des voitures en attente. Si oui, il envoie immédiatement une autorisation à la voiture prioritaire suivante, simulant une réallocation dynamique des ressources.

5.4.3 Simulation du temps de stationnement

Une fois acceptée, la voiture est "stationnée" virtuellement pour une durée déterminée, simulée par un Thread.sleep() d'une durée aléatoire comprise entre 5 et 10 secondes. Cette simulation permet de reproduire un comportement réaliste : les voitures n'occupent pas les places indéfiniment, ce qui crée un roulement dans les parkings. Une fois le délai écoulé, la place est libérée et l'agent Parking actualise automatiquement l'interface et traite la prochaine voiture en attente si besoin.

Ce comportement crée une dynamique d'occupation et de désengorgement très proche d'un système réel, où des voitures arrivent, se garent, puis quittent le parking.

5.4.4 Synchronisation avec l'interface utilisateur

Toutes les mises à jour d'état (occupation des parkings, files d'attente, état des voitures) sont synchronisées avec l'interface JavaFX via la classe UIBridge. Cette classe statique centralise les appels de mise à jour en veillant à ce qu'ils soient exécutés sur le bon thread (Platform.runLater), ce qui est essentiel pour ne pas provoquer d'erreurs de thread dans JavaFX(figure 5.6).

Par exemple, lorsqu'un agent Parking accepte une voiture, il appelle :

```
45      public void updateParkingState(String name, int total, int free) {  1 usage
46          Platform.runLater(() -> {
47              parkingFreeSpots.put(name, free);
48              parkingList.getItems().removeIf( String item -> item.startsWith(name + ":"));
49              parkingList.getItems().add(name + ": " + free + "/" + total + " places libres");
50          });
```

Figure 5.6 : Méthode d'actualisation parking

De même, lorsqu'il ajoute une voiture dans la file d'attente :

```
53      public void updateWaitingList(String parkingName, List<String> queueDisplay) {  1 usage
54          Platform.runLater(() -> {
55              waitingList.getItems().clear();
56              waitingList.getItems().addAll(queueDisplay);
57          });
58      }
```

Figure 5.7 : Méthode d'actualisation la file d'attente

Ce système assure une cohérence parfaite entre la logique multi-agents et la visualisation en temps réel.

5.5 Développement de l'interface JavaFX (FXML et logique contrôleur)

L'interface utilisateur constitue une partie essentielle de notre système, car elle permet à l'utilisateur final de visualiser en temps réel l'état des parkings, des voitures, et des files d'attente. Pour concevoir une interface fluide, moderne et facilement maintenable, nous avons choisi d'utiliser **JavaFX**, un framework graphique de nouvelle génération pour Java, combiné avec **FXML** pour la définition déclarative des vues.

5.5.1 Pourquoi JavaFX ?

JavaFX s'impose comme un choix naturel dans les projets modernes en Java pour plusieurs raisons :

- **Séparation claire entre interface et logique** : grâce au support du langage FXML, JavaFX permet de séparer complètement le design de l'interface (le « frontend ») de la logique de traitement (le « backend » ou contrôleur).
- **Support avancé des composants graphiques** : JavaFX propose une vaste bibliothèque de composants modernes (ListView, GridPane, VBox, etc.) et permet la personnalisation aisée de leur apparence.
- **Compatibilité avec les threads et tâches asynchrones** : très utile dans le cadre de notre projet multi-agents, où les mises à jour doivent être faites en temps réel sans bloquer le fil d'exécution principal.
- **Support des animations, transitions et binding dynamique** : des éléments qui permettent d'améliorer l'expérience utilisateur.

5.5.2 Utilisation de FXML pour la définition de la vue

FXML est un langage basé sur XML qui permet de décrire l'interface graphique de manière déclarative. Au lieu de créer chaque composant JavaFX via du code Java classique, nous avons défini l'interface dans un fichier .fxml. Cela rend la structure de l'interface plus lisible, plus modulaire et plus facile à maintenir.

Voici un extrait simplifié de notre fichier FXML :

```
<right>
    <VBox spacing="10">
        <padding><Insets top="10" right="10" bottom="10" left="10"/></padding>
        <Label text="Liste des Voitures" style="-fx-font-size: 16px;"/>
        <ListView fx:id="carList" prefHeight="200" prefWidth="250"/>
        <Label text="File d'attente" style="-fx-font-size: 14px; -fx-padding: 5 0 0 0;"/>
        <ListView fx:id="waitingList" prefHeight="150" prefWidth="250"/>
        <Button fx:id="startSimulation" text="Démarrer la simulation" onAction="#startSimulation"/>
    </VBox>
</right>
```

Figure 5.8 : Un extrait du code de MainView.fxml

Ce fichier est ensuite chargé au démarrage de l'application via la méthode FXMLLoader.load(), qui instancie automatiquement tous les composants et les relie à un contrôleur Java.

5.5.3 Le contrôleur JavaFX (MainController.java)

Le fichier FXML est associé à une classe Java appelée **contrôleur**, dans notre cas MainController.java. Ce contrôleur contient toutes les méthodes qui manipulent l'interface utilisateur, en réponse aux événements ou aux changements d'état du système.

Les éléments définis dans le FXML sont injectés automatiquement dans le contrôleur via l'annotation @FXML. Cela nous permet de les manipuler directement, par exemple pour mettre à jour la liste des voitures ou la grille des parkings.

Voici quelques exemples de méthodes importantes dans le contrôleur :

- **startSimulation()** : appelée lorsque l'utilisateur clique sur le bouton de démarrage. Elle lance un thread qui génère progressivement des agents Car, chacun avec une priorité aléatoire.
- **updateParking()** : appelée par les agents Parking via la classe UIBridge, cette méthode met à jour l'état affiché d'un parking donné dans la grille et la liste.
- **updateWaitingQueue()** : met à jour la file d'attente visible à l'écran, en fonction des données renvoyées par les agents.

L'utilisation de Platform.runLater() est essentielle ici pour garantir que toutes les mises à jour graphiques soient exécutées sur le thread JavaFX, évitant ainsi les erreurs de synchronisation.

5.5.4 Affichage dynamique et interactif

Grâce à JavaFX, nous avons pu représenter graphiquement les parkings dans une **grille dynamique (GridPane)**, où chaque parking est matérialisé par une boîte contenant son nom, son nombre total de places et son nombre de places disponibles.

À chaque changement d'état (arrivée ou départ d'une voiture), les agents mettent à jour les valeurs dans l'interface. Cela permet à l'utilisateur de suivre en temps réel l'occupation de chaque parking, mais aussi de voir les voitures en attente dans une **ListView dédiée à la file d'attente**, avec un affichage qui précise la priorité de chaque voiture.

5.5.5 Intégration entre JADE et JavaFX

L'interconnexion entre JADE et JavaFX a été réalisée à l'aide d'une **classe passerelle UIBridge**, conçue pour faire le lien entre le monde des agents et l'interface. Les agents n'interagissent jamais directement avec les composants JavaFX : ils appellent des méthodes de UIBridge, qui à son tour relaye l'appel vers le contrôleur.

5.6 Simulation complète du système et démonstration

La phase de simulation représente l'étape cruciale de validation du système de gestion intelligente de parking que nous avons conçu. Elle permet non seulement de tester les différents scénarios d'interaction entre agents, mais également d'illustrer visuellement le comportement dynamique de notre application dans un contexte proche de la réalité.

5.6.1 Déroulement général de la simulation

Lorsque l'utilisateur clique sur « Démarrer la simulation », dix voitures sont créées progressivement, chacune représentée par un agent CarAgent doté d'une priorité aléatoire (de 1 à 10). Chaque voiture envoie une requête aux quatre agents ParkingAgent pour obtenir une place.

Si un parking est disponible, il accepte la voiture, réduit le nombre de places et démarre un timer simulant le stationnement (entre 5 et 10 secondes). Sinon, la voiture est placée dans une file d'attente prioritaire (PriorityQueue), assurant que les véhicules les plus urgents seront traités en premier.

À la fin du stationnement, la place est libérée et le parking affecte automatiquement cette place à la voiture prioritaire suivante dans la file.

5.6.2 Visualisation dynamique dans l'interface

Tout le processus décrit ci-dessus est **reflété en temps réel dans l'interface JavaFX**. Plusieurs composants visuels sont mis à jour dynamiquement :

- La **liste des voitures (ListView)** affiche chaque voiture créée avec son niveau de priorité.
- La **liste des parkings** indique pour chacun le nombre total et le nombre de places libres.
- Une **grille centrale** (GridPane) présente chaque parking sous forme de carte, avec son nom, le nombre de places et les disponibilités mises à jour en temps réel.
- La **file d'attente visible (ListView)** est également synchronisée avec l'état des agents : elle affiche les voitures en attente, ordonnées selon leur priorité, dans chaque parking concerné.

Cette visualisation dynamique permet à l'utilisateur d'avoir une vision claire de la circulation, de la saturation des parkings, et du traitement différencié selon la priorité des véhicules.

5.6.3 Délai, animation et comportement naturel

Pour rendre la simulation plus crédible, nous avons intégré des **temps d'attente aléatoires** entre les arrivées de voitures. Cela permet d'éviter une arrivée instantanée de tous les véhicules et reproduit plutôt un flux réaliste et continu, comme on pourrait l'observer dans une situation réelle.

De même, la durée de stationnement est elle aussi variable, simulant des véhicules ayant des temps de passage différents. Cette variabilité permet également de tester les capacités d'auto-régulation du système, en particulier la gestion intelligente des files d'attente.

5.6.4 Résultat attendu

À l'issue d'une simulation complète, l'utilisateur peut observer plusieurs points clés :

- Le système attribue correctement les places aux voitures en fonction de leur arrivée et de la disponibilité des parkings.
- Les files d'attente sont gérées de manière ordonnée et priorisée.
- Les voitures entrent et sortent automatiquement, libérant dynamiquement des places pour d'autres véhicules.
- L'état des parkings est toujours reflété fidèlement dans l'interface, grâce à la communication continue entre agents et composants JavaFX.

Cela permet de démontrer l'efficacité du système multi-agents à s'auto-organiser en fonction d'un flux variable de demandes, dans un cadre graphique lisible, ergonomique et représentatif.

5.6.5 Robustesse et stabilité

Tout au long de la simulation, les communications entre agents JADE et l'interface JavaFX se font via la classe UIBridge, qui joue un rôle de passerelle intermédiaire. Cela garantit que toutes les mises à jour visuelles sont faites dans le bon contexte de thread (Platform.runLater()), évitant les plantages ou comportements imprévisibles. Le système se montre stable, même en présence de nombreuses requêtes ou d'occupation complète des parkings.

5.7 Difficultés rencontrées et solutions apportées

D Plusieurs défis ont été surmontés durant l'implémentation. Le principal concernait la synchronisation entre JADE et JavaFX : comme JavaFX exige des mises à jour graphiques sur son thread principal, une classe passerelle UIBridge utilisant Platform.runLater() a été introduite.

Des erreurs liées au lancement trop rapide des agents ont également été corrigées en retardant l'activation de la simulation jusqu'à la fin de l'initialisation JADE. Par ailleurs, la file d'attente a été convertie en PriorityQueue pour traiter les voitures selon leur niveau de priorité, conformément à la logique intelligente souhaitée.

Enfin, des problèmes d'affichage ont été réglés par une centralisation systématique des mises à jour graphiques après chaque changement d'état. Grâce à ces solutions, le système est devenu fluide, fiable et conforme aux attentes.

5.8 Résultats obtenus et observations

L'implémentation du système a démontré son efficacité dans la gestion dynamique des voitures selon leur priorité. Les simulations montrent que les voitures arrivent aléatoirement, sont dirigées vers les parkings disponibles, ou placées en file d'attente lorsque ceux-ci sont pleins. La priorité inversée est bien respectée : les voitures les plus urgentes sont prises en charge en premier.

L'interface JavaFX, reliée aux agents via UIBridge, permet de suivre l'état des parkings et des files d'attente en temps réel. Ce retour visuel a été déterminant pour valider le bon fonctionnement du système et son comportement fluide, même en situation de saturation.

Aucune anomalie critique n'a été relevée lors des tests. Le système s'est montré stable, réactif et conforme aux attentes, validant ainsi l'efficacité de l'approche multi-agents et des choix technologiques adoptés.

5.9 Conclusion

Ce chapitre a décrit l'implémentation du système de parking intelligent, en mettant en œuvre JavaFX pour l'interface graphique et JADE pour la logique multi-agents. Le système permet de simuler dynamiquement l'arrivée de voitures et leur répartition selon la priorité. Les outils choisis ont permis une architecture modulaire, fluide et extensible, posant les bases pour de futures améliorations.

Conclusion Génerale

Conclusion générale

Dans un monde où la mobilité urbaine devient de plus en plus complexe, la nécessité de solutions intelligentes et autonomes pour gérer les infrastructures publiques n'a jamais été aussi pressante. Le présent mémoire s'est inscrit dans cette dynamique en proposant un système intelligent de gestion de parkings, basé sur une approche multi-agents combinée à une interface graphique moderne. L'objectif principal était de concevoir un système capable de simuler, de manière réaliste et fluide, l'allocation dynamique de places de stationnement selon des critères de priorité et de disponibilité en temps réel.

Au fil de ce travail, plusieurs étapes majeures ont été franchies. D'abord, une étude approfondie de l'**Internet des Objets (IoT)** a permis de contextualiser le système dans une vision plus large d'interconnexion et de communication entre entités intelligentes. Ensuite, une modélisation comportementale a été réalisée grâce à l'outil **Renew**, selon le formalisme de *Nets Within Nets*, pour visualiser l'interaction entre les voitures et les parkings dans une architecture distribuée. Cette modélisation a servi de base logique pour la phase d'implémentation.

L'implémentation elle-même a été réalisée avec soin, en s'appuyant sur des choix technologiques pertinents. La plateforme **JADE** a offert un environnement robuste pour simuler des agents autonomes communiquant via messages ACL, tandis que **JavaFX** a permis de concevoir une interface graphique claire, interactive et capable de refléter les états du système en temps réel. Le langage **Java**, bien adapté à la programmation orientée objet et à l'intégration de bibliothèques tierces, a facilité la structuration du projet. Enfin, **IntelliJ IDEA** a été choisi comme environnement de développement pour sa richesse fonctionnelle, sa stabilité et sa compatibilité avec les frameworks utilisés.

Le résultat obtenu est un système dynamique, capable de recevoir des requêtes de voitures, de les classer selon leur priorité, d'allouer les places de parking disponibles et de gérer une file d'attente lorsque les capacités sont saturées. L'utilisateur peut visualiser en temps réel l'état de chaque parking, les files d'attente, ainsi que la liste des voitures classées par priorité. Cette approche allie rigueur algorithmique, intelligence décentralisée et souci d'ergonomie.

Au-delà des aspects techniques, ce projet a représenté une véritable montée en compétences, tant sur le plan conceptuel que pratique. Il a exigé une compréhension approfondie des paradigmes orientés agents, une maîtrise des interfaces modernes et une capacité à faire dialoguer des composants hétérogènes dans un système cohérent.

Cependant, certaines limites subsistent. Le système reste une simulation et ne prend pas encore en compte les contraintes spatiales réelles, les capteurs physiques, ni l'intégration à des données de géolocalisation. De plus, les mécanismes de décision restent relativement simples, basés uniquement sur la priorité, sans tenir compte de critères plus complexes comme le temps d'attente ou la distance au parking. Ces aspects constituent autant de pistes de **perspectives futures**, qui pourraient faire évoluer le système vers une version

connectée à l'IoT, optimisée par des algorithmes d'apprentissage automatique ou intégrée dans une plateforme mobile destinée aux utilisateurs finaux.

En conclusion, ce mémoire témoigne de la faisabilité et de la pertinence d'un système intelligent de gestion de parkings basé sur des agents autonomes. Il ouvre la voie à des réflexions plus larges sur la manière dont les technologies distribuées, l'intelligence artificielle et les interfaces modernes peuvent transformer l'expérience urbaine. Ce projet constitue ainsi un socle solide pour des applications futures à fort potentiel dans le domaine des villes intelligentes (*smart cities*).

Bibliographie

Bibliographie :

- Bellifemine, F., Caire, G., & Greenwood, D. (2007). Developing Multi-Agent Systems
  - with JADE. John Wiley & Sons.
  - → Un guide complet sur le framework JADE, essentiel pour la création de systèmes
  - distribués basés sur les agents.
- Wooldridge, M. (2009). An Introduction to MultiAgent Systems (2nd ed.). Wiley.
  - → Référence de base pour la compréhension théorique des systèmes multi-agents et de
  - leur modélisation.
- Schmidt, W., Moldt, D., & Wienberg, F. (2000). Modeling Dynamic Systems Using Nets
  - within Nets. In Applications and Theory of Petri Nets 2000 (pp. 205–224). Springer.
  - → Article de référence sur la modélisation hiérarchique utilisant Renew et les Nets within
  - Nets.
- Jennings, N. R., & Wooldridge, M. (1998). Agent Technology: Foundations,
  - Applications, and Markets. Springer.
  - → Fournit un aperçu des différentes applications de la technologie des agents intelligents.
- Huhns, M. N., & Singh, M. P. (1998). Readings in Agents. Morgan Kaufmann Publishers.
  - → Recueil d'articles fondamentaux sur les agents logiciels, leurs interactions et
  - architectures.
- Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A Survey. Computer
  - Networks, 54(15), 2787–2805.
- → Revue approfondie du concept d'IoT, ses protocoles et son intégration dans des
  - systèmes intelligents.
- Ferber, J. (1999). Multi-Agent Systems: An Introduction to Distributed Artificial
  - Intelligence. Addison-Wesley.
  - → Présente les fondements de l'intelligence distribuée à travers les systèmes multi-
  - agents.
- Reisig, W. (2013). Understanding Petri Nets: Modeling Techniques, Analysis Methods,
  - Case Studies. Springer.
  - → Manuel technique pour comprendre et appliquer les réseaux de Petri classiques et
  - étendus.
- JADE Framework Documentation. (2024). Java Agent DEvelopment Framework.
  - Site officiel : https://jade.tilab.com
  - → Fournit la documentation API et les guides pour implémenter et déployer des agents
  - JADE.
- JavaFX Documentation (OpenJFX). Oracle & OpenJFX Project.
  - Site officiel : https://openjfx.io
  - → Documentation pour la création d'interfaces graphiques modernes en Java.
- Java Platform, Standard Edition (Java SE) Documentation. Oracle.
  - Site officiel : https://docs.oracle.com/javase
  - → Guide de référence pour les APIs Java utilisées dans tout le projet.
  - FXML Documentation. Oracle.
- Guide XML pour la définition déclarative d'interfaces JavaFX.

  - Site : https://docs.oracle.com/javase/8/javafx/fxml-tutorial/
- Renew Tool – Petri Net Modeling. University of Hamburg.
  - Site officiel : https://www.renew.de
  - → Outil de modélisation de réseaux de Petri, utilisé pour représenter la logique du

- système de réservation.

- IntelliJ IDEA Documentation. JetBrains.
  - Site officiel : https://www.jetbrains.com/idea/documentation/
  - → Environnement de développement intégré (IDE) utilisé pour coder, compiler et
  - simuler le système.
- GeeksForGeeks. (2024). JavaFX Tutorial – Complete Guide for Beginners.
  - Site : https://www.geeksforgeeks.org/javafx-tutorial/
  - → Tutoriel étape par étape pour comprendre et maîtriser JavaFX.
- Baeldung. (2023). Guide to JavaFX, Threads and Platform.runLater().
  - Site : https://www.baeldung.com/javafx-platform-runlater
  - → Explication claire de la gestion des threads et de la synchronisation JavaFX.
- Medium.com. (2023). How to Build Smart Agent Systems with JADE Framework.
  - → Expériences de développeurs sur l'utilisation pratique de JADE dans des systèmes
  - intelligents.
- Stack Overflow. (Consulté en 2024).
  - → Nombreuses discussions sur les problèmes liés à JADE, JavaFX, FXML et threads
  - Java.
- GitHub. Repositories open-source de projets similaires utilisant JADE & JavaFX :
  - https://github.com/search?q=jade+javafx
  - → Source d'inspiration pour la structure, les agents, les comportements, et les
  - intégrations.

- Chamseddine, A. (2020). Système intelligent de gestion de trafic basé sur les agents
  - JADE. Mémoire de Master, Université de Boumerdes.
  - → Projet similaire avec agents JADE et logique distribuée.
- Tchana, C. (2021). Conception d'un système multi-agents pour la gestion des parkings
  - intelligents. Université de Lyon 1.
  - → Implémentation concrète d'un système intelligent de parking avec JADE.
- Boudjemaa, F. (2019). Utilisation de JavaFX pour le développement d'interfaces
  - interactives intelligentes. Mémoire, USTHB.
  - → Étude sur l'usage de JavaFX et FXML pour des interfaces utilisateurs réactives.

- Bah, S. (2005). Modélisation et évaluation des performances des systèmes à événements discrets par les réseaux de Pétri.
- Boimond, J.L. (2004). Cours sur les réseaux de Pétri, Université du Havre.
- Wikipédia, article : Réseaux de Pétri.
- Références pédagogiques générales Réseaux de Pétri.
- Kaiser, C. (2001). Systèmes à événements discrets et Réseaux de Pétri, Université de Technologie de Compiègne
- latribune.fr
- b2b.2raventure.com+1demainlaville.com+1
- core.ac.uk
- databridgemarketresearch.com
- peakboard.com
- bosch.com
- Memoire VF ADJAOUT Karima_16-01-2022
- https://elearning.centre-univ-mila.dz/ chapitre 2 réseau petri