Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Projet Recherche & Développement

2018-2019

Amélioration d'un protocole de rechargement de capteurs

Tuteurs académiques
Tifenn Rault
Ronan Bocquillon

Étudiant
Thomas Couchoud (DI5)

5 décembre 2018

Liste des intervenants

Nom	Email	Qualité
Thomas Couchoud	thomas.couchoud@etu.univ-tours.fr	Étudiant DI5
Tifenn Rault	tifenn.rault@univ-tours.fr	Tuteur académique, Département Informatique
Ronan Bocquillon	ronan.bocquillon@univ-tours.fr	Tuteur académique, Département Informatique

Avertissement

Ce document a été rédigé par Thomas Couchoud susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Tifenn Rault et Ronan Bocquillon susnommés les tuteurs académiques.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnait assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnait qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable des tuteurs académiques et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document

Thomas Couchoud, *Amélioration d'un protocole de rechargement de capteurs*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
    author={Couchoud, Thomas},
    title={Amélioration d'un protocole de rechargement de capteurs},
    type={Projet Recherche \& Développement},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2018-2019}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V2.2.5(2017/05/02)

Table des matières

Table des figures

Liste des tableaux

1 Introduction

1 Contexte

Nous vivons aujourd'hui dans une société dans laquelle la collecte d'information occupe une place de plus en plus importante. Cette collecte peut notamment se faire par le biais de capteurs installés à différents endroits.

Dans une quête de quantité et qualité des données, les capteurs peuvent être multipliés pour couvrir une plus grande zone et sont amenés à se retrouver dans des environnements très différents afin d'être au plus proche de la source d'information.

Du fait de leur petite taille, un moyen d'assurer leur adaptation à la diversité des emplacements consiste à les alimenter par le biais de batteries. Ce choix technique induit cependant des contrainte énergétique concernant le rechargement de ces dernières. Cette problématique de recharge de capteurs au sein d'un réseau a récemment gagné une attention considérable dans la communauté scientifique.

En effet les technologies de transfert d'énergie sans fil (wireless energy transfer (WET)) offrent de nouvelles solutions afin d'addresser le problème. Deux options s'offrent à nous, recharger en mode point-à-point ou point-à-multipoint.

- **Point-à-point** : ce mode permet de charger directement un capteur en approchant le chargeur à une distance faible. Dans ce cas le rechargement est a son efficacité maximum.
- **Point-à-multipoint** : ce procédé nous autorise à charger plusieurs capteurs avec le même chargeur. Tous les capteurs se trouvant dans un certain rayon autour du chargeur seront affectés. Cela permet des temps de chargements plus courts et moins de déplacements.

Selon les choix des outils mis en place et la politique de rechargement voulu, nous arrivons à distinguer différentes approches afin de définir le chemin que le(s) chargeur(s) devront emprunter.

- **Rechargement périodique** : le chargeur va effectuer de manière périodique une route optimale préconfigurée. Certes le chemin est optimal cependant cela implique que nous connaissons à l'avance l'état et la structure du réseau de capteurs.
- **Rechargement à la demande** : ce procédé corrige le défaut précédent en établissant des routes à la volée. En effet, chaque capteur notifie dès qu'il veut être rechargé et le chemin du chargeur sera adapté à la demande. Cela permet ainsi de s'adapter à des réseaux plus dynamiques.

- **Rechargement collaboratif** : dans ce cas de figure, le but est de prendre en compte plusieurs chargeurs. Il est donc ici question de parralléliser les rechargements de manière efficace afin de pouvoir supporter des réseaux de plus grande taille.

Cependant comme dit précédemment, il existe la technologie point-à-multipoint. Cette dernière, certes pratique, introduit une nouvelle contrainte à cause de son utilisation de radiations électromagnétiques (electromagnetic radiation (EMR)). Ces rayonnements représentent un potentiel risque de santé si leur intensité dépassent un certain seuil de sécurité. Si une telle technologie est utilisée, notamment combiné avec le rechargement collaboratif, il est nécessaire de tenir compte de cette problématique.

2 Objectifs

Ce projet de fin d'études a pour but de reprendre un papier réalisé par Mme Rault [12], enseignante-chercheuse de Polytech'Tours, afin d'étudier les améliorations possibles à la solution proposée. Ce dernier décrit une solution afin de générer des tournées de mobile chargers (MCs) de type point-à-multipoint dans un réseau de capteurs à la demande tout en tenant compte des EMRs.

Cela passe par une première phase d'analyse du sujet ainsi que de la solution apportée par le papier. Une deuxième phase aura pour but de repérer les différents éléments pouvant améliorer l'algorithme actuel.

Ces changements se concrétiserons au travers d'un simulateur qui sera pensé et réalisé afin d'exécuter différents scénarios et en obtenir des métriques sur le système.

3 Bases méthodologiques

Afin de mener a bien le projet, différents outils vont être utilisés.

- La réalisation du projet se déroulera au travers d'une méthode Agile. En effet cette méthode permet de réaliser des « sprints » de manière régulière dans lesquelles un livrable est réalisé à la fin. Ce choix permet ainsi d'avoir un retour rapide sur notre rythme d'implémentation des fonctionnalités et permet une adaptabilité plus flexible à la demande. De plus avoir des livrables de manière régulière me paraît important afin d'obtenir des impressions de la part des utilisateurs tout au long de la réalisation.

- Concernant la planification des tâches, cela se fera au travers d'un système de « cartes ». Chaque carte contient un objectif précis ainsi qu'une date limite de réalisation.

  Ce choix me parait approprié car il est facile de mélanger des tâches plus théoriques avec des tâches plus pratiques. En effet on définit toutes les cartes (tâches) du projet dès qu'on les connait, les tâches théoriques (recherche, spécification, ...) auront une date de fin connue dès le départ tandis que les tâches techniques seront récupérées par groupes lors de la composition d'un sprint. Les tâches retenues pour former ce groupe dépendra ainsi des retours précédents ainsi que des priorités du client.

  Cette gestion et découpage est détaillé en Annexe B.

- L'hébergement du code se fera sur un dépôt Git. Dans mon cas j'ai préféré me tourner vers Gitlab. Ce service permet de garder une trace de tout le code réalisé dans le temps. Cela est notamment utile si l'on veut revenir en arrière dans le code si un problème majeur survient.

De plus cet hébergeur met aussi à disposition un outil CI (Continuous integration). Si le temps me le permet il sera utile de le déployer afin d'automatiser la compilation et les tests.

- Enfin le rapport sera écrit en LaTeX en suivant la classe plolytech.

2 Description générale

1 Environnement

La réalisation du simulateur va impliquer de coder l'algorithme détaillé dans le papier de Mme Rault [12]. De ce fait la résolution du travelling salesman problem (TSP) et travelling salesman problem with multiple time windows (TSPMTW) vont devoir être réalisés. Afin de ne pas perdre de temps sur la résolution de ces problèmes connus, une librairie sera utilisée. L'une d'entre elle est OR-Tools développée par Google. Cette dernière est disponible dans les languages Python, C++, C# ou bien Java ce qui permet de ne pas bloquer notre choix du language.

A la vue de cette liberté et du fait que la solution sera utilisé sur une multitude de postes potentiellement très différents les uns des autres, il peut être préférable de choisir un language tel que Python ou Java afin d'améliorer la portabilité du simulateur sur différents systèmes d'exploitation.

2 Utilisateur

Le simulateur est réalisé à destination de chercheurs dans le domaine des capteurs sans fil. Même si certains d'entre eux peuvent avoir des connaissances en informatique on ne peut généraliser cela à toute la population concernée. Il faudra donc réaliser le programme afin qu'il soit utilisable de la manière la plus simple possible, tout en permettant aux utilisateurs un peu plus avancés de personaliser le simulateur de manière plus poussée.

	Connaissance de l'informatique	Expérience de l'application	Type d'utilisateur	Périmètre d'action
Chercheur sans connaissances informatiques	Non	Non	Utilisation des fonctionnalisées basiques de manière régulière ou ponctuelle	Lancer des simulations avec un algorithme donné mais possibilité de changer les paramètres de la simulation

Chercheur avec connaissances informatiques	Oui	Non	Utilisation complète de l'application de manière régulière ou ponctuel	Lancer des simulation avec différents paramètres et implémenter son propre algorithme de routage des différents chargeurs.

Table 2 – *Caractéristiques utilisateurs*

3 Fonctionnalités

Le diagramme de cas d'utilisation est disponible ci-dessous (Figure 1).

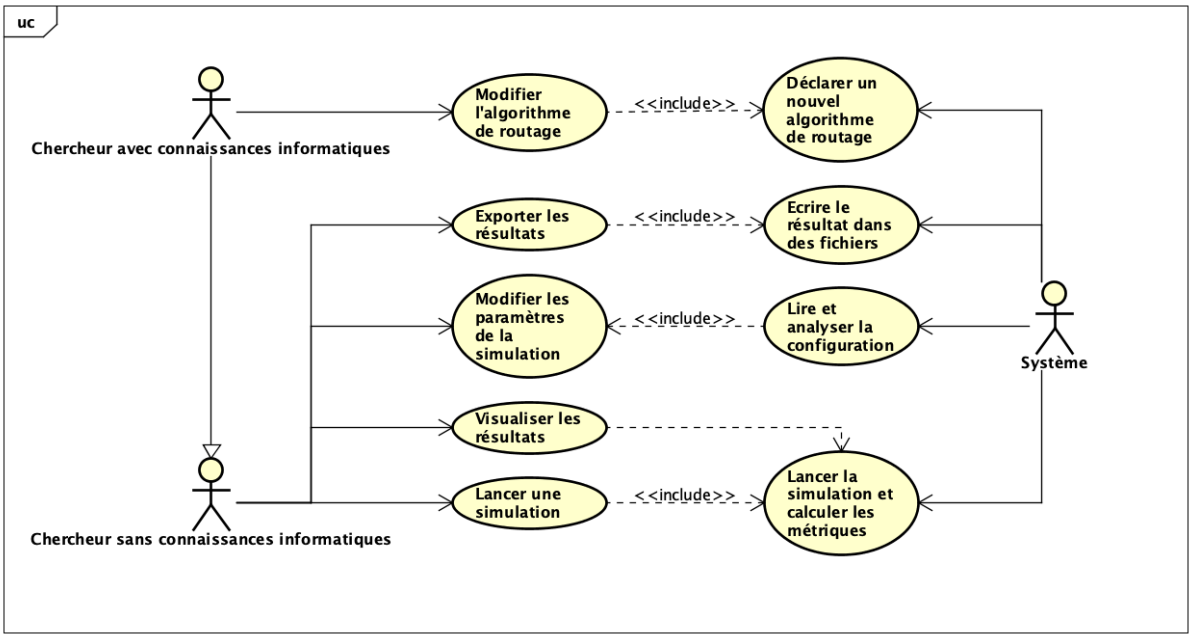

Figure 1 – *Diagramme de cas d'utilisation*

On peut y observer les deux roles vu dans la Section 2 ainsi que le système.

Le chercheur sans connaissances informatiques va pouvoir effectuer les actions de base :

- Modifier les paramètres de la simulation. On y retrouve tout ce qui touche au capteurs ainsi qu'aux chargeurs. En voici une liste non exhaustive :
  - Nombre de capteurs.
  - Propriétés par « type » de capteur :
    - Capacité maximale.
    - Puissance minimum de reception.
  - Nombre de chargeurs.
  - Propriétés par « type » de chargeur :

        * Capacité.
        * Vitesse maximale.
        * Fonction de la consommation pour le trajet.
        * Fonction de l'efficacité du chargement.
        * Puissance de transmission.
        * Rayon d'action.
    - Méthode de routage.

- Une fois les paramètres réglés, l'utilisateur aura la possibilité de démarrer/arrêter la simulation.
- Lorsque la simulation est terminée, les différentes métriques pourront être visualisées directement dans l'interface. On y retrouve par exemple :
    - Temps de panne des nœuds.
    - Temps d'attente.
    - Énergie dépensée par les chargeurs.
    - Distance parcourue.
- Enfin, afin de ne pas perdre les résultats obtenus il sera possible d'exporter les résultats (métriques obtenues).

Vient ensuite le chercheur avec les connaissances informatiques. Nous incluons dans cette catégorie tout ceux qui sont capables de programmer dans le language de notre application. Ces derniers auront les même possibilités que les chercheurs sans connaissances informatiques avec en plus :

- Écriture d'algorithmes de routage, capteurs, chargeurs, etc. et enregistrement de ces derniers auprès du système

Enfin le système sera chargé de gérer le déroulement de la simulation :

- Tout commence par la lecture et interprétation de la configuration ainsi que des algorithmes de routage.
- En fonction de ce qui a été lu, le système exécute la simulation en tenant compte de ce qui a été défini par l'utilisateur.
- Enfin les données de la simulation pourront conduire à des métriques qui seront affichées dans l'interface et exportés dans des fichiers.

4 Structure générale

Afin de représenter la structure interne, un diagramme de composant a été réalisé et est disponible ci-dessous.

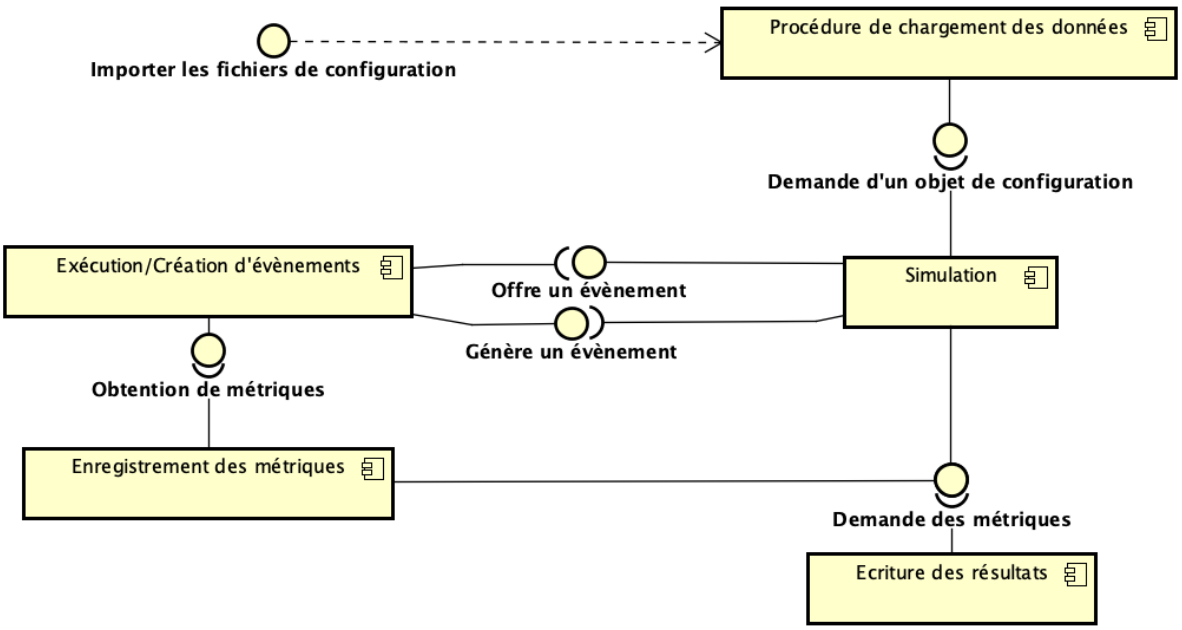

Figure 2 – *Diagramme de composant*

On y retrouve 5 composants principaux :

- Chargement des données : cette partie aura pour but de lire le fichier de configuration qui est fourni par l'utilisateur afin de rendre ces données accessibles au reste du programme.
- Simulation : cette partie a comme objectif de gérer les différents évènements a réaliser. Cela comprend notamment de les ranger dans l'ordre de leur occurence, éventuellement priorité ainsi que générer les évènements de début/fin.
- Exécution/création d'évènements : dans cette partie on exécute le code associé à chaque évènement. C'est ici que des modifications sur l'état du système et génération de nouveaux évènements vont être réalisés.
- Enregistrement des métriques : A partir de l'exécution des évènements nous pouvons extraire des valeurs sur des métriques ; cette partie a pour but d'enregistrer ces dernières afin de pouvoir les synthétiser par la suite.
- Écriture des résultats : lors de la fin de la simulation et avec les données des métriques enregistrées, ce composant offrira à l'utilisateur des données plus synthétisées/agrégées afin de faciliter la lecture de ces dernières. On peut par exemple construire un graphique.

3 État de l'art

Nous avons évoqué en introduction que cette problématique de chargement d'un réseau de capteurs avait récemment suscité une attention particulière de la part de la communauté scientifique. Cela a donc mené à un certain nombre de papiers couvrant différent cas de résolution. Extrait du papier de Mme Rault [12] on peut observer les différents champs couverts :

Référence	Chargement à la demande	Chargement point-à-multipoint	Chargeurs multiples	Contrainte EMR
[16] [7]	non	oui	non	non
[13] [11]	oui	non	non	non
[6]	oui	oui	non	non
[8] [9] [15] [5]	oui	non	oui	non
[4]	non	non	oui	non
[10] [2] [1] [3]	non	oui	non	oui
[12]	oui	oui	oui	oui

Table 2 – *Cas étudiés par d'autres papiers*

Beaucoup de travaux ont été réalisés sur des combinaisons de paramètres différents mais les recherches combinant tous ces derniers se font rares. Le papier de Mme Rault, qui est notre point de départ, propose une solution prenant en compte toutes les contraintes précédentes, c'est à dire générer des routes à la demande pour un réseau de capteurs avec plusieurs chargeurs point-à-multipoint tout en considérant la problématique des EMR.

Commençons tout d'abord par expliquer la solution proposée par Mme Rault puis nous détaillerons certains éléments qui ont pu être utilisés dans ce papier.

1 Solution de Mme Rault

La solution proposée se découpe comme suit :

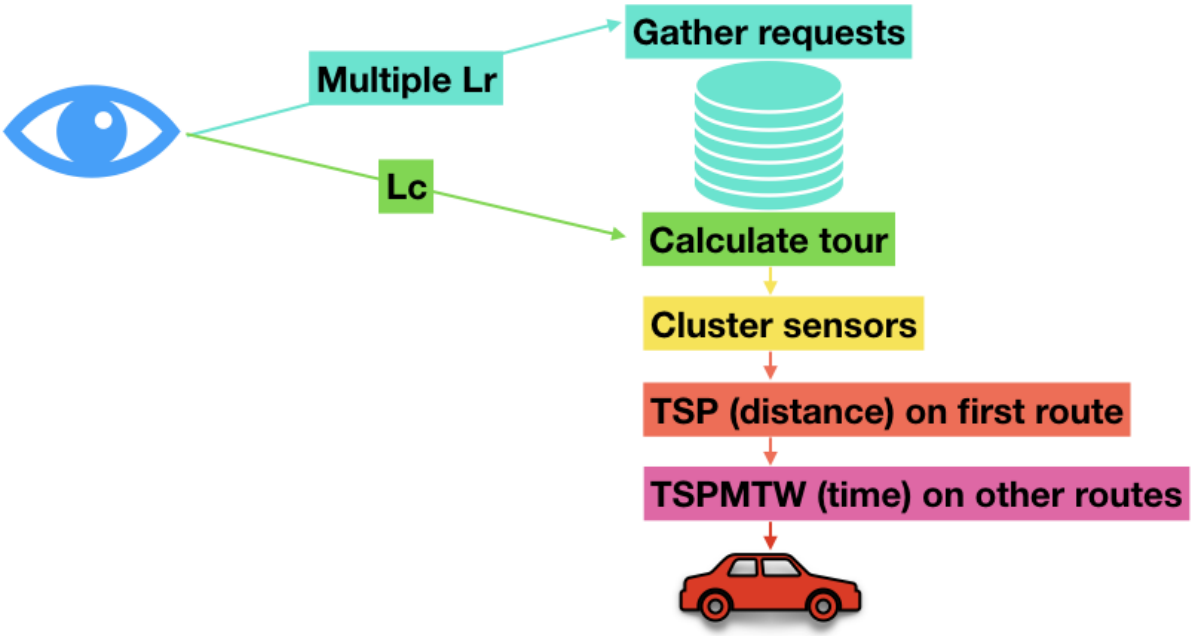

Figure 1 – *Enchainement des étapes de la solution*

Le point de départ se situe en haut à gauche avec nos différents capteurs. La première étape est de proposer un service de chargement à la demande. Il est donc nécessaire de collecter des demandes provenant des capteurs. Pour cela une solution proposée par [6] est utilisée. Cette dernière repose sur un système où deux messages, L_r et L_c, sont émis par les capteurs. Lorsque le niveau de batterie de ces derniers tombe en dessous d'un certain seuil ils envoient un signal L_r au point gérant le réseau. Ces signaux ne sont pas alarmant et sont donc stockés jusqu'à ce qu'un signal L_c d'un capteur est reçu. Ce dernier indique un niveau plus critique d'une batterie et nous lançons alors la génération d'une nouvelle tournée de rechargement.

La génération des routes se fait en trois étapes :

- Les différents capteurs sont clusterisés, c'est à dire répartis dans plusieurs groupes, en fonction du rayon d'action R_c des EMR. Ces derniers sont formés d'une manière à minimiser le nombre points d'arrêts que le véhicule aura a effectuer [6]. Minimiser le nombre d'arrêts nous permet de générer des clusters rechargeant le maximum de capteurs à la fois.

  Une fois ces points d'arrêt définis, il nous faut les affecter à l'un des chargeurs disponibles. Pour ce faire nous commençons par assigner un premier point de passage, tiré aléatoirement, à chaque chargeur. Ensuite nous sélectionnons le chargeur avec le moins de temps accumulé (somme des temps de recharge) et lui affectons le point de passage le plus proche n'étant pas déjà assigné. De ce fait nous minimisons le temps de voyage tout en équilibrant les temps de recharge entre les chargeurs.

- Maintenant que nous avons les différents points de passage pour chaque chargeur il nous faut leur donner un ordre. C'est ici que notre contrainte des EMR a une grande influence. En effet lors du calcul de ces routes nous devons nous assurer qu'il n'y ait aucun chargeur qui rentre en conflit avec un deuxième.

  Pour prendre cela en compte, nous introduisons la notion de points de chargement en conflit. Deux points i et j sont en conflits si $dist(i,j) < R_{c_i} + R_{c_j}$. Notre problème se ramène donc à calculer nos routes de manière à ce que deux points en conflit ne soient pas chargés en même temps.

  – La solution proposée consiste à calculer dans un premier temps un premier TSP sur les points d'un premier chargeurs. La fonction objectif de ce TSP est de minimiser la distance parcourue.

– Dans un second temps nous calculons les routes des autres chargeurs séquentiellement grâce à un TSPMTW. Les fenêtres de temps sont calculés en retirant pour chaque zones de conflit les moments où un autre chargeur est présent. Ainsi pour chaque chargeur i passant par un point de conflit, nous modifions la fenêtre de temps W_j par $W_j \leftarrow W_j \setminus \{[a_i - t_j, a_i + t_i]\}$ où a_x re présente la date d'arrivé du chargeur x sur le point et t_x le temps d'arrêt du chargeur x au point. Le TSPMTW est ensuite lancé avec pour but de minimiser le temps de trajet.

  En effet il nécessaire de minimiser le temps d'attente au niveau des points de conflit. Si le TSPMTW à pour objectif de minimiser la distance, il est fort possible qu'un conflit apparaisse ce qui entrainera un temps d'attente. Il est plus profitable de parcourir un peu plus de distance et commencer à charger un cluster plutôt que d'attendre que la zone se libère.

2 Génération des clusters

La génération des clusters est un point important dans l'algorithme. En effet c'est ce dernier qui va définir les points d'arrêt de nos MCs. Il est donc nécessaire d'optimiser cette partie afin de ne pas se retrouver dans un cas où le chargeur recharge les capteurs un par un (la fonction du chargement point-à-multipoint serait annulée).

Afin d'obtenir une solution efficace, le papier [6] propose de minimiser le nombre de ces points d'arrêt. En effet, plus cette donnée est minimale, plus l'on a regroupé les chargeurs entre eux.

Le problème est posé comme suit :

Soit $\mathcal{N}$ l'ensemble des nœuds ayant effectués une requête. On pose $|\mathcal{N}| = n$. Chaque capteur i dispose dispose d'un disque dans lequel le chargeur peut être placé afin de recharger ce dernier, on le note D_i.

Il est a noter que dans cet algorithme on suppose que le rayon du disque de rechargement du MC est constant.

Afin de déterminer les points d'arrêts, il faut déterminer des intersections entre les différents disques. Le problème revient donc à trouver le minimum de zones d'intersections tel que tout les disques appartiennent à au moins l'une de ces zones.

Ces régions R_j sont définies par l'intersection de plusieurs disques D_i ou bien par un disque D_i lui même. On obtient alors $\mathcal{N}_j = \bigcup_{i=1}^{n} \{i | R_j \cap D_i \neq 0\}$ l'ensemble des nœuds participant à une zone d'intersection. On a aussi $\mathcal{N} = \bigcup_{j=1}^{m} \mathcal{N}_j$ du fait que chaque nœud doit participer à une intersection.

La résolution du problème se fait en posant deux nouvelles variables :

$A = [a_{i,j}]$ un matrice $n \times m$ où $a_{i,j} = \begin{cases} 1 & \text{si le nœud } i \in \mathcal{N}_j \\ 0 & \text{sinon} \end{cases}$.

x_j variable binaire de décision telle que $x_j = \begin{cases} 1 & \text{si la zone } R_j \text{ est dans le chemin minimum} \\ 0 & \text{sinon} \end{cases}$.

La fonction objectif devient alors $\min \sum_{j=1}^{m} x_j$ (on minimise le nombre de zones d'arrêt) avec une contrainte : $\sum_{j=1}^{m} a_{i,j} x_j \geq 1; \forall i \in \{1, \ldots, n\}$ (chaque chargeur appartient à au moins une zone d'arrêt faisant parti du chemin considéré).

3 TSP

Le TSP [14] est un problème très connu dans la communauté scientifique concernant l'optimisation combinatoire. Il pose la question suivante : « Ayant une liste de villes et connaissant les distances entre elles, quelle est la plus courte distance telle qu'un voyageur visite chaque ville une seule fois et revienne à sont point de départ ? ». De manière parallèle on peut aussi s'intéresser au chemin tel que le temps de parcours est le minimum. Cela s'applique bien à notre problème, les villes correspondent aux différents points d'arrêt et le voyageur est un MC.

Ce problème fait cependant partie de la classe NP-difficile. C'est à dire qu'il n'existe pas d'algorithme déterministique s'exécutant en temps polynomial. Il y a donc un choix à faire quant aux solutions que nous voulons obtenir : peu de temps de calcul mais une solution qui peut être loin de l'optimal, ou beaucoup de temps de recherche avec une solutions plus proche de l'optimal.

Dans notre cas nous devons réaliser la tournée de nos MCs de manière rapide, le premier choix sera donc de préférence.

Le problème se traduit par un model Integer Linear Programming (ILP). On numérote les villes $1,\dots,n$ et définissons $x_{i,j} = \begin{cases} 1 & \text{il existe un chemin de } i \text{ à } j \\ 0 & \text{sinon} \end{cases}$.

Pour tout $i,j \in [1,\dots,n]$ on pose u_i une variable muette et $c_{i,j}$ la distance entre la ville i et j. Le problème aura pour fonction objectif :

$$\min \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} c_{i,j} x_{i,j} \tag{1}$$

Avec comme contraintes :

$$0 \leq x_{i,j} \leq 1 \tag{2}$$

$$u_i \in \mathbb{Z}$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} x_{i,j} = 1 \qquad \forall j \in \{1,\dots,n\} \tag{3}$$

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} x_{i,j} = 1 \qquad \forall i \in \{1,\dots,n\} \tag{4}$$

$$u_i - u_j + n x_{i,j} \leq n-1 \qquad 2 \leq i \neq j \leq n \tag{5}$$

$$0 \leq u_i \leq n-1 \qquad 2 \leq i \leq n \tag{6}$$

L'Equation 1 définit notre fonction objectif cherchant à minimiser la distance parcourue. Si l'on veut changer la métrique à réduire, il suffit de changer ce que représente $c_{i,j}$. Dans notre cas cela pourrait être le temps de parcours entre deux nœuds.

L'Equation 2 nous permet de restreindre $x_{i,j}$ aux valeurs 0 et 1 (on est en nombre entiers).

Les équations Equation 3 et Equation 4 comptent respectivement le nombre de chemins sortants ou entrant sur un nœud donné. Ces deux valeurs deivent être égal à 1 pour n'avoir qu'un seul chemin passant par chaque point.

Enfin les équations Equation 5 et Equation 6 assurent qu'il n'y ait qu'un seul chemin parcourant l'ensemble des nœuds (et non pas plusieurs chemins disjoints).

4 TSPMTW

Le TSPMTW est similaire au TSP mais introduit une contrainte supplémentaire. Le « MTW » supplémentaire dans le nom indique « multiple time window » soit « fenêtre de temps multiple » et c'est sur ce concept que la contrainte supplémentaire va se baser.

Le principe est de définir :

- Une (ou plusieurs) fenêtre(s) de temps par nœud dans lesquelles ce dernier est disponible.
- Un temps de service β_i correspondant au temps d'arrêt nécessaire pour le nœud i.

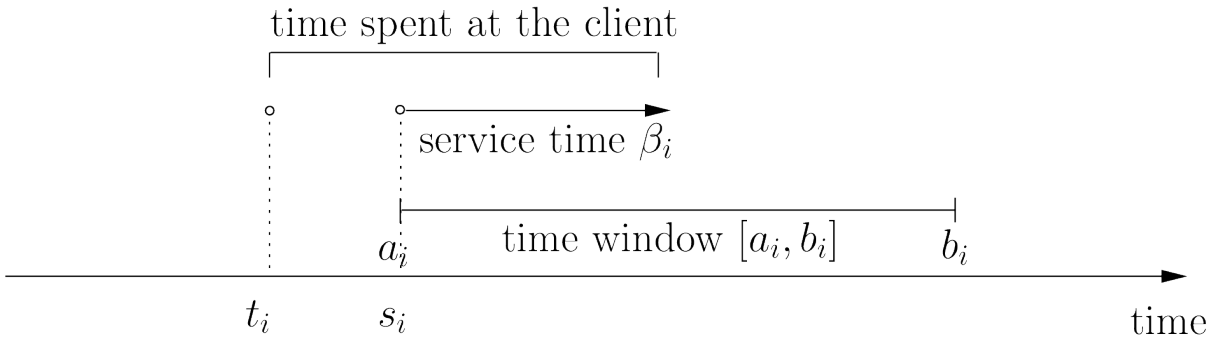

Figure 2 – *Illustration d'une fenêtre de temps (source)*

On peut observer tout en bas du schéma une fenêtre de temps. Cette dernière est représenté par une date de début et une date de fin $[a_{i,j}; b_{i,j}]$ avec i étant l'indice du nœud et j l'indice de la fenêtre. Ainsi en considérant plusieurs fenêtres de temps, l'interval de disponibilité d'un nœud devient $[a_i; b_i] = \bigcup_j [a_{i,j}; b_{i,j}]$.

Dans la timeline on observe deux valeurs : t_i étant la date d'arrivé au nœud et s_i la date de début du service. Comme dit précédemment le temps de service correspond à β_i.

Ainsi le temps total passé au nœud i correspond à la partie supérieure, prenant en compte le temps de service et le temps d'attente de la disponibilité du noeud.

Le TSPMTW est utilisé dans notre cas afin d'éviter la surcharge des EMRs. En effet, chaque nœud débute avec une fenêtre de temps $[a_i; b_i] = [0; +\infty[$. À chaque chargement d'un de ces nœuds par un chargeur, on retire la fenêtre de temps d'occupation de ce dernier mais aussi des autres nœuds qui sont en conflit. Nos fenêtres de temps deviennent alors $[a_i; b_i] \longleftarrow [a_i; b_i] \setminus \{[s_j - \beta_i; s_j + \beta_j]\}$, pour tout chargeur $j \neq i$ avec j le chargeur qui vient de faire son service. De cette manière nous bannissons au fur et à mesure les moments où les EMRs nous posent problème.

4 Analyse et conception

Dans ce chapitre nous allons nous intéresser à l'analyse de notre problème et à la conception de notre simulateur afin de pouvoir dégager des axes d'amélioration de l'algorithme qui pourront par la suite être testés dans le simulateur.

1 Analyse théorique

La première étape est donc l'analyse mathématique du problème. Si nous reprenons le schéma global de l'algorithme décrit dans le papier de Mme Rault [12] (voir Figure 1(Chapitre 3)) on remarque que ce dernier est découpé en plusieurs étapes, lesquelles font appel à leur propre algorithmes. Nous avons donc ici des opportunités d'amélioration à plusieurs niveau de manière indépendantes. Cependant il ne faudra pas négliger les effets pouvant se produire si l'on combine plusieurs de ces améliorations (deux améliorations positives, mises ensemble, peuvent mener à une dégradation de manière générale).

Nous allons ici détailler plusieurs axes. Ces derniers ne seront peut être pas tous testés par la suite.

1.1 Seuil L_c

On rappelle que L_c représente le signal critique envoyé par un capteur et déclenche une tournée des chargeurs. Nous allons nous intéresser à deux cas extrêmes :

- L_c **proche de** L_r **(seuil de demande de rechargement)** : Dans ce cas dès qu'un capteur juge qu'il nécessite d'être rechargé il lancera la tournée des chargeurs. Cela aura pour effet d'annuler la collecte d'une liste de capteur pour aller les recharger en groupe. Chaque tournée consistera à aller charger un capteur unique. Dans notre cas cela n'a que très peu d'intérêt. Il sera donc préférable de garder L_c relativement distant de L_r afin de pouvoir constituer des listes de nœuds à charger assez conséquentes.

- L_c **proche de 0** : Une valeur proche de 0 lancerait les tournées lorsque le capteur est sur le point d'être hors d'usage. C'est encore une fois un point que nous souhaitons éviter. Il est donc nécessaire de s'écarter du 0 en prenant une marge suffisante pour couvrir le temps que les chargeurs vont mettre pour venir charger notre nœud.

On remarque donc que la valeur de L_c doit faire un compromis entre s'assurer que les pannes des capteurs sont inexistantes (valeur haute) et laisser de la marge pour grouper les appels des différents nœuds (valeur basse) tout en évitant toute pannes.

1.2 Seuil L_r

L_r représente le seuil auquel un capteur va faire une demande de rechargement. De manière similaire avec L_c, nous allons nous intéresser au valeurs extrêmes.

- L_r **proche de** L_C : De manière évidente c'est la même chose que L_c proche de L_r ; voir Section 1.1.
- L_r **proche de** C_i **(capacité maximale d'un capteur)** : Cela va avoir pour effet d'enregistrer un nœud comme nécessitant un rechargement alors que la batterie est encore relativement pleine. A cause de cet effet on risque de créer des tournées où la totalité des capteurs est présente.

  Dans ce cas la notion de « à la demande » n'a plus de sens et revient à créer une tournée planifiée à l'avance et l'exécuter lors de la réception d'un L_c.

  Les batteries prendront surement moins de temps à charger mais d'un autre coté elles peuvent aussi être rechargés si elles se trouvent dans le rayon du chargeur lors d'un rechargement d'un nœud voisin même si notre capteur n'avait pas fait de demande.

Tout comme L_c, il nous faut garder L_r à une valeur distante de C_i tout en laissant de la marge pour ne constituer des tournées qu'avec des capteurs ayant un réel besoin (et ainsi éviter les déplacements inutiles des chargeurs).

1.3 Influence du nombre de MCs

On peut avoir tendance à penser que plus nous augmentons le nombre de MCs plus nous rendons notre système efficace car nous ajoutons de la capacité à traiter les capteurs en parrallèle. Cependant ce n'est pas toujours le cas. Deux raisons expliquent cela :

- En augmentant le nombre de MC, on augmente aussi la consommation globale en énergie. En effet nous nécessitons de plus d'énergie pour déplacer l'ensemble des chargeurs.
- On augmente la complexité des TSPMTWs vis à vis des EMRs. Cela s'explique par le fait que plus l'on a de chargeurs, plus l'on à d'interférences possible entre ces derniers. Il se peut donc qu'avec trop de chargeurs on introduise un nombre conséquent de temps d'attente afin d'éviter des conflits.

Il va donc falloir que nous trouvions un nombre adéquat de chargeurs afin de ne pas surcharger la zone tout en en gardant un nombre suffisant pour pouvoir paralléliser nos rechargements.

1.4 Rayons de rechargement différents

Actuellement l'algorithme proposé ne prend en compte qu'un seul rayon de rechargement pour tous les capteurs. Cependant il pourrait être intéressant de considérer des chargeurs ayant des rayons différents. Cela pourrait peut-être améliorer dans certains cas l'algorithme de clustering.

A terme cela pourrait réduire le temps d'attente en réduisant le nombre de zones en conflit. Cependant cela ajoute une complexité supplémentaire dans la génération de nos clusters.

Il est cependant dans mon opinion pas très intéressant d'explorer cette piste. En effet comme précisé dans l'étude de l'art (Section 2(Chapitre 3)), on suppose que le disque de rechargement M est constant. Une manière simple de travailler avec des rayons différents est de poser $M = \min_i M_i$. Cette approche ne prend cependant pas avantage des rayons différentes.

Une autre approche consisterai en une exploration de différentes solutions où les différents M de chaque cluster prend une valeur parmi les M_i disponibles. On peut imaginer utiliser des algorithmes génétiques ou autres afin de trouver une solution s'approchant de l'optimale. Cependant cette façon de procéder rajoute un nombre conséquent de répétitions de l'algorithme de clustering (avec différents M). L'ajout de cette complexité n'est probablement pas avantageux comparé au gain minime de performance du système que l'on pourrai obtenir.

1.5 Modification de la fonction objectif des TPSMTWs

Un autre point sur lequel nous pouvons influencer est la façon dont sont généré les routes des chargeurs une fois que le premier tracé est défini. Nous utilisons pour le moment une fonction objectif minimisant le temps de parcours. Cependant on peut envisager de modifier cette dernière pour prendre en compte une combinaison du temps et de la distance.

Les paramètre de la combinaison entre les deux variables reste à déterminer expérimentalement. Selon moi cela aura un impact que très minime. En effet le fait de minimiser le temps minimise aussi la distance parcourue d'une certaine manière puisque se déplacer prends du temps.

1.6 Modification de la méthode de clustering

La méthode utilisée pour le clustering est un point central concernant l'aspect « point-to-multipoint » de nos chargeurs. En effet c'est ce dernier qui s'occupe de transformer notre liste de capteurs en un nombre réduit de points d'arrêt qui seront utilisés pour notre tournée. Cela nous permet d'obtenir un point précis d'arrêt ainsi qui de réduire le nombre d'arrêts.

Actuellement la création de ces régions d'arrêt fait en sorte qu'un capteur se retrouve associé à au moins un point d'arrêt. Cependant il peut être aussi faire parti de plusieurs régions à la fois. Cette remarque peut être très intéressante car de cette manière on peut considérer le rechargement d'un capteur depuis plusieurs points d'arrêt.

Cela est utile si l'on a un capteur fortement déchargé entouré de capteurs moyennement déchargés. Ainsi on commence par charger une partie des capteurs, dont celui fortement déchargé. Dès que les capteurs moyennement déchargés sont complètement chargés, on change de zone pour recharger une autre partie moyennement déchargée tout en continuant le rechargement du capteur fortement déchargé (et qui doit donc être moyennement déchargé à ce moment).

Voici une représentation de la situation. On dispose de deux disques (bleu et cyan) qui ont été construits par notre algorithme de clustering. Le centre de chacun d'entre eux représente un point d'arrêt.

Les yeux représentes nos différents capteurs (vert : chargés, orange : moyennement déchargés, rouge : fortement déchargés).

Le véhicule rose représente un chargeur.

On peut imaginer le premier état comme étant le rechargement du disque bleu.

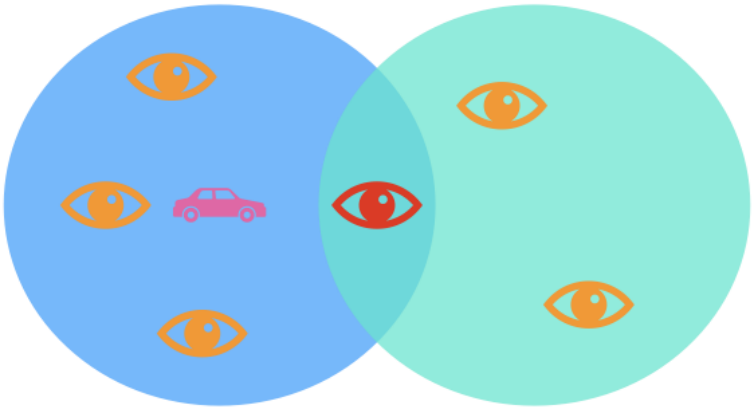

Figure 1 – *Etat A*

Enfin lorsque les capteurs orange du disque bleu sont chargés, le véhicule se déplace dans le disque cyan et effectue le chargement.

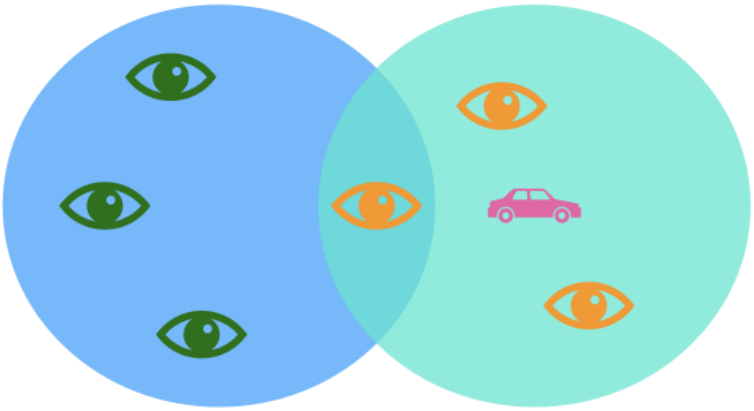

Figure 2 – *Etat B*

Notre nœud originellement rouge a donc été chargé en deux phases, réduisant ainsi le temps de tournée du véhicule.

Afin de prendre en compte cette appartenance multiple, il nous faut adapter le step 1 du papier de Mme Rault [12] (Fig. 2.). La méthode actuelle consiste à prendre pour chaque région R_j $t_j = \max_{i \in R_j} t_{j,i}$ où $t_{j,i}$ représente le temps de charge d'un capteur i appartenant à la région R_j.

Afin de prendre en compte le principe de rechargement depuis plusieurs zones, on propose de modifier l'algorithme permettant de calculer le temps d'arrêt d'une région :

Algorithm 1 Calcul des temps de recharge à chaque point d'arrêt en prenant en compte les nœuds présents dans plusieurs régions

Input : The set $S = \{1, \ldots, m\}$ of m regions obtained from the 2 firsts steps of the first step. The set $S_j = \{1, \ldots, n_j\}$ of n_j sensors inside a region R_j with their charging time τ_i, $\forall i \in S$. t_i is the charging time required by the sensor i. The set K which for each sensor contains a set containing each regions the sensor is in.

for $j \in S$ **do**
  $P_j \leftarrow \{\}$
  $t_j \leftarrow 0$
  for $i \in S_j$ **do**
    if $\text{card}(K_i) > 1$ **then**
      $P_j \leftarrow P_j \cup \{i\}$
      $K_i \leftarrow K_i \setminus \{j\}$
    else
      $\tau_j \leftarrow \max(\tau_j, t_i)$
    end if
  end for
  for $i \in P_j$ **do**
    $t_i \leftarrow t_i - \tau_j$
  end for
end for

1.7 Méthode de répartition des points d'arrêts

Lors de l'assignement des premiers clusters à chaque chargeur, il peut être intéressant d'utiliser une méthode qui n'est pas aléatoire. En effet si l'on se retrouve avec des chargeurs devant aller dans des régions voisines dès le départ n'est peut être pas optimal. En revanche répartir ces premières destinations au travers de la zone de capteurs peut améliorer la distance parcourue afin de couvrir cette région entièrement.

1.8 Eléments d'analyse

Nous allons dans un premier temps s'intéresser aux influence des paramètres suivants :

- Seuil L_c.
- Seuil L_r.
- Nombre de MCs.
- Modification de la méthode de clustering.
- Modification de la fonction objectif des TSPMTWs.

2 Analyse logicielle

Afin de réaliser la partie logicielle, c'est à dire le simulateur, on propose de suivre un diagramme de classe similaire à celui ci-dessous.

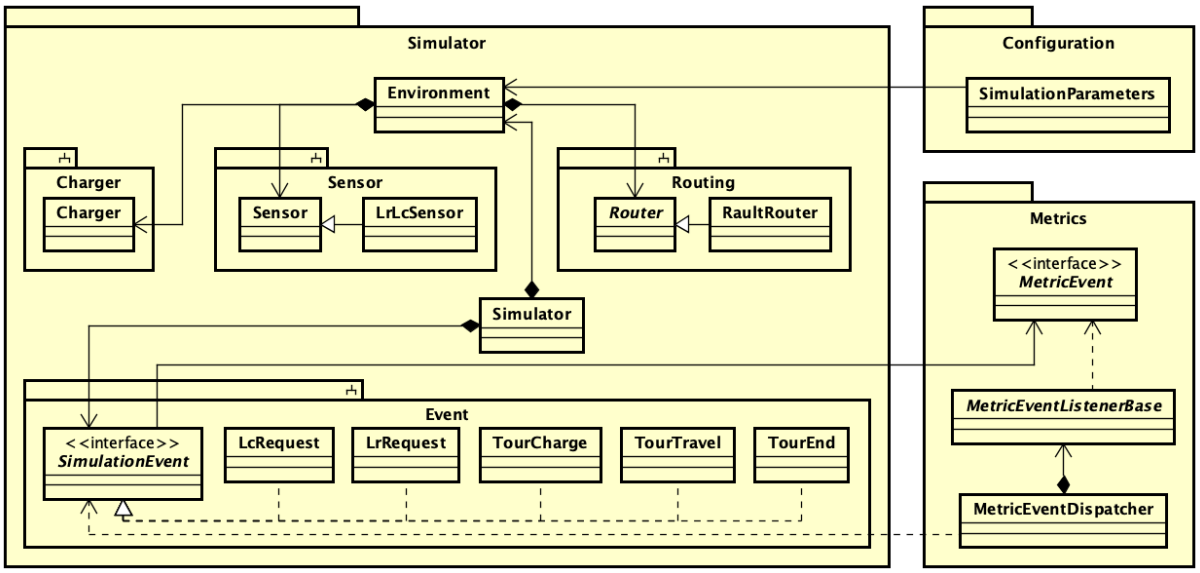

Figure 3 – *Diagramme de classes simplifié*

On y retrouves trois parties principales :

- Configuration : rassemble toutes les classes qui serviront à paramétrer notre simulateur.
- Simulator : contient tout les éléments liés à la simulation cela passe par chargeurs, capteurs, systèmes de routage mais aussi par des évènements qui permettront de faire avancer la simulation.
- Metrics : concerne toute la partie de collecte des informations sur le système afin de les restituer et agréger par la suite.

2.1 Configuration

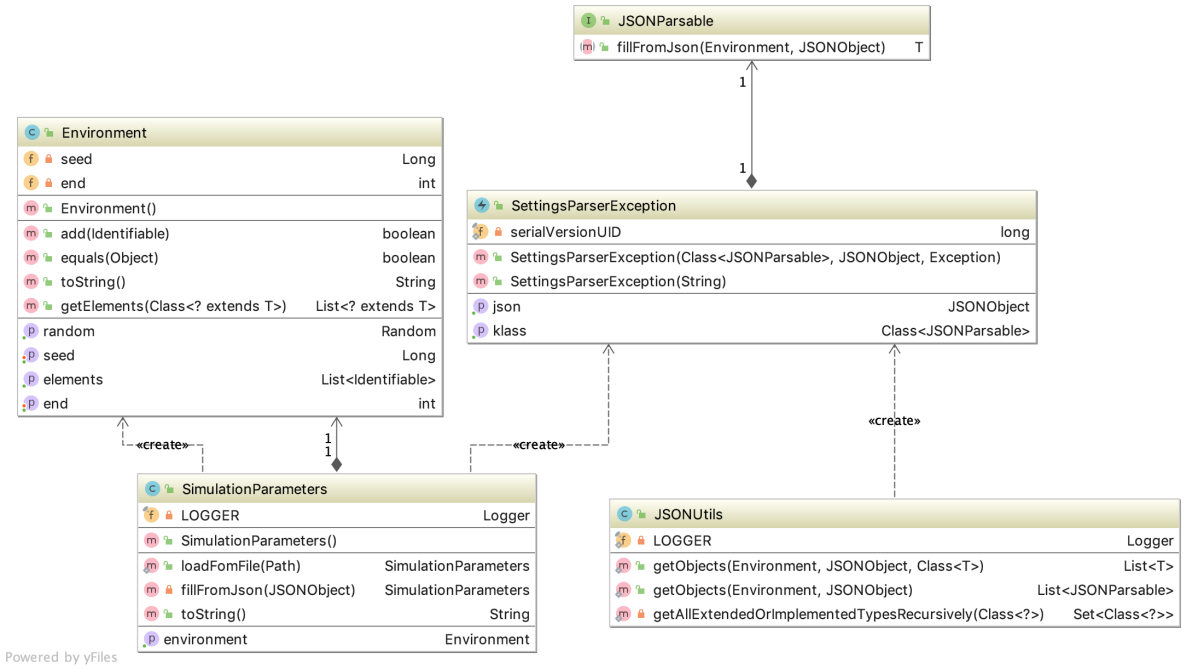

Figure 4 – *Partie configuration*

La partie configuration va comporter une classe principale permettant de lire un fichier JSON qui mettra par la suite un objet Environment disponible au travers d'un getter. Ce chargement de paramètres doit être dynamique en autorisant l'instantiation de classes à partir de leur noms et de paramètres. Pour cela on créera une interface « JSONParsable » permettant à un objet lisible depuis le JSON de récupérer les paramètres dont il a besoin.

Si on problème survient durant la lecture de la configuration, une exception SettingsParserException sera levée en indiquant où se situe le problème.

2.2 Simulator

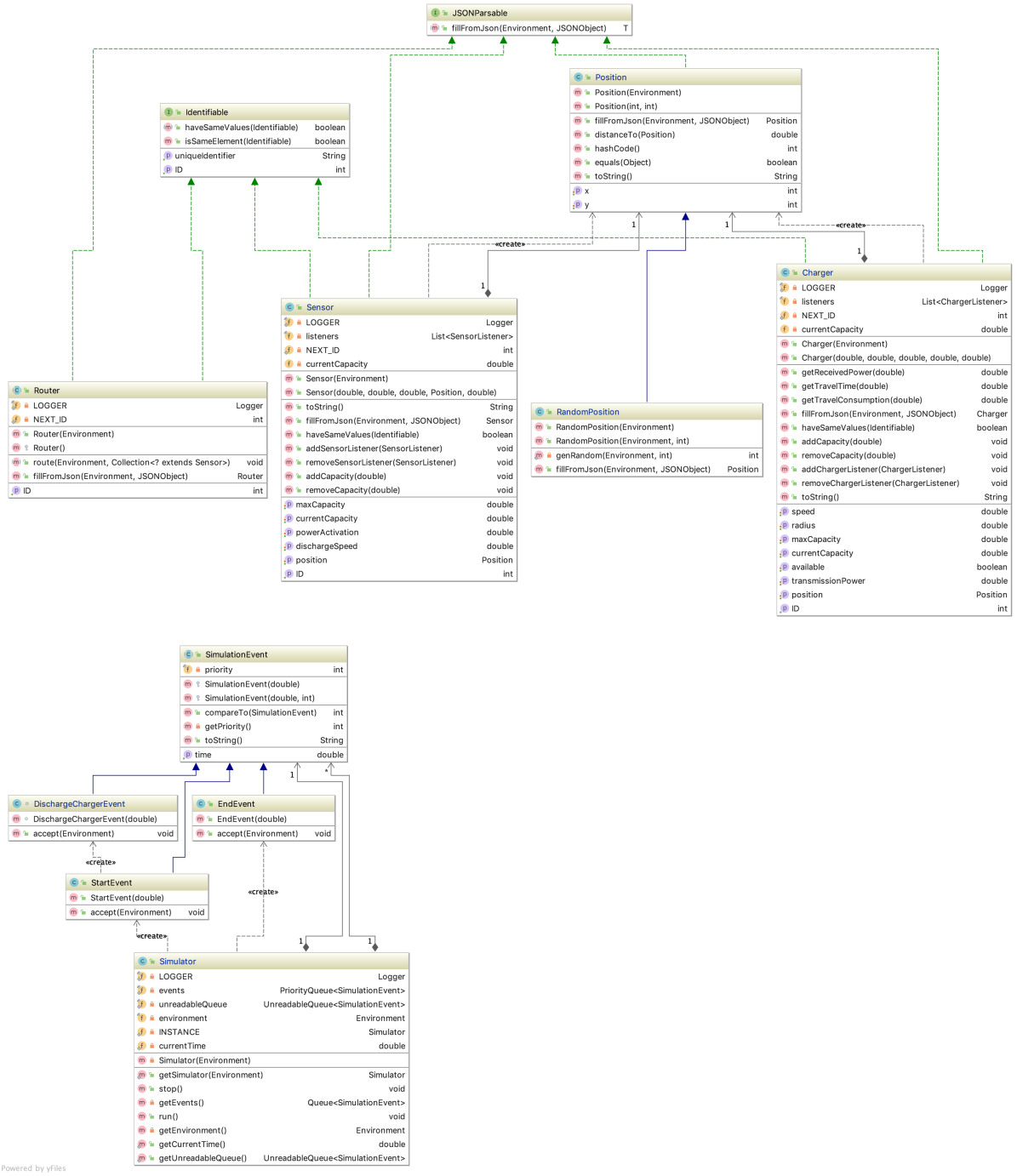

Figure 5 – *Partie simulator*

Le diagramme de classes ci-dessus nous montre la partie simulateur dans son état le plus simple – aucun algorithme de routage n'a été ajouté et les capteur ne font que se décharger.

La partie supérieure concerne le « modèle » avec les différents éléments de notre environnement (router, charger, sensor). Cependant si par la suite nous voulons ajouter des éléments personnalisés, il suffit d'étendre la classe Identifiable et ajouter ces éléments dans l'environnement. Nous permettons donc grâce à cette interface une flexibilité sur les éléments de la simulation.

La partie inférieure se focalise plus sur la simulation en elle même avec notre échéancier et les différents évènements qui l'entourent. De base seul les évènements Start (début), End (Fin) et Discharge (déchargement des capteurs) sont présents. Cependant lors de l'ajout de nouveaux éléments Identifiables, ces derniers pourront créer de nouveaux évènements (SimulationEvent) et le simulateur les traitera aussi. Par exemple on créé une classe LrLcSensor qui extend Sensor, et dès que le niveau de batterie est inférieur à L_r ce dernier va créer un évènement LrEvent.

2.3 Metrics

La collecte des métriques est un point important de la simulation et doit permettre de collecter un grand nombres de données.

Pour cela on propose de suivre un design pattern de producteur consommateur. Les producteurs seront des éléments de la simulation qui généreront des évènements indiquant un changement dans l'état des variables. Les consommateurs vont par la suite consommer ces évènements, ne gardant que ceux qui les intéressent, et traiter ce changement.

Les consommateurs seront aussi responsables d'écrire les résultats dans les différents fichiers de sortie.

2.4 Diagramme complet

Le diagramme de classes complet dans l'état actuel (la partie métriques n'est pas implémentée et l'algorithme de routage non fini) est disponible en Figure 1(Annexe D).

2.5 Limites

La gestion du simulateur sous forme d'évènements impose certaines limites :

- Le temps est discret et non continue.
- Les temps de calculs des routes n'est pas pris en compte. Cette limitation a été est acceptée. Si nécessaire il est possible de la contourner en ajoutant un nouvel intermédiaire qui rendra compte de ce temps de calcul.
- Du fait du temps discret, les chargeurs chargent en 1 coup les différents capteurs puis attendent un certain temps avant de repartir. Cela ne rend pas compte de la réalité qui a un chargement continu. Cependant cette approximation nous est suffisante. De la même manière cette problématique peut être contournée en ajoutant des évènements intermédiaires.

  Des problèmes peuvent survenir si les chargeurs chargent moins vite que la vitesse d'utilisation des batteries par les capteurs. Dans ce cas notre approximation serait fausse. Cependant un tel système n'aurait aucun sens car la finalité serait un réseau dont tous les capteurs sont déchargés.

5 Mise en œuvre

S10

6 Bilan et conclusion

Ce Projet de Recherche et Développement se découpe en deux étapes majeures (S9 et S10). La première est principalement axée sur l'analyse de tout l'écosystème du projet ainsi que la définition du cahier de spécification. La seconde en revanche consiste en le développement de la solution imaginée et l'analyse des résultats obtenus.

1 Bilan S9

Nous allons ici exposer les différentes tâches qui ont été réalisés lors du S9 ainsi que celle qui sont en retard. Nous terminerons ensuite cette partie par un premier aperçu des tâches qu'il reste à faire.

1.1 Réalisation

1.1.1 Recherche – Spécifications

- Compréhension du sujet.
- Analyse de la solution proposée ainsi que des papiers liés.
- Identification des améliorations possibles.
- Modélisation algorithmique.
- Écriture des spécifications.
- Écriture du rapport du S9 & préparation de la soutenance.
- Réalisation d'un diagramme de classe global pour le simulateur.

1.2 Développement

- Codage d'un parser de configuration.

- Ajout de tests pour le parser.
- Codage du simulateur à son niveau le plus bas.
- Ajout de classes/évènements pour modéliser la solution de Mme Rault.
- Début d'ajout de la collecte de métriques.

1.3 En retard

Aucun retard sur les tâches n'est encore survenu. En réalité il y a plutôt une avance dans la réalisation du projet avec la partie codage du simulateur qui est actuellement à un stage déjà quasi fonctionnel.

1.4 Planification S10

Le déroulé du S10 repose principalement sur la réalisation du simulateur ainsi que l'étude des résultats obtenus.

Pour cela il sera nécessaire de faire les tâches suivantes :

- Finalisation du simulateur :
  - Compléter l'implémentation de la solution de Mme Rault :
    - Ajout des calculs des TSPMTWs.
    - Ajout de l'algorithme calculant les points d'arrêts.
  - Capturer plus de métriques.
  - Exportation des résultats sous forme graphique.
  - Implémentation des amélioration proposées.
  - Ajout de tests au simulateur.
- Création des jeux de données pour le simulateur.
- Analyse des résultats
- Rédaction du rapport du S10.

2 Bilan S10

S10

Annexes

A Découverte du sujet

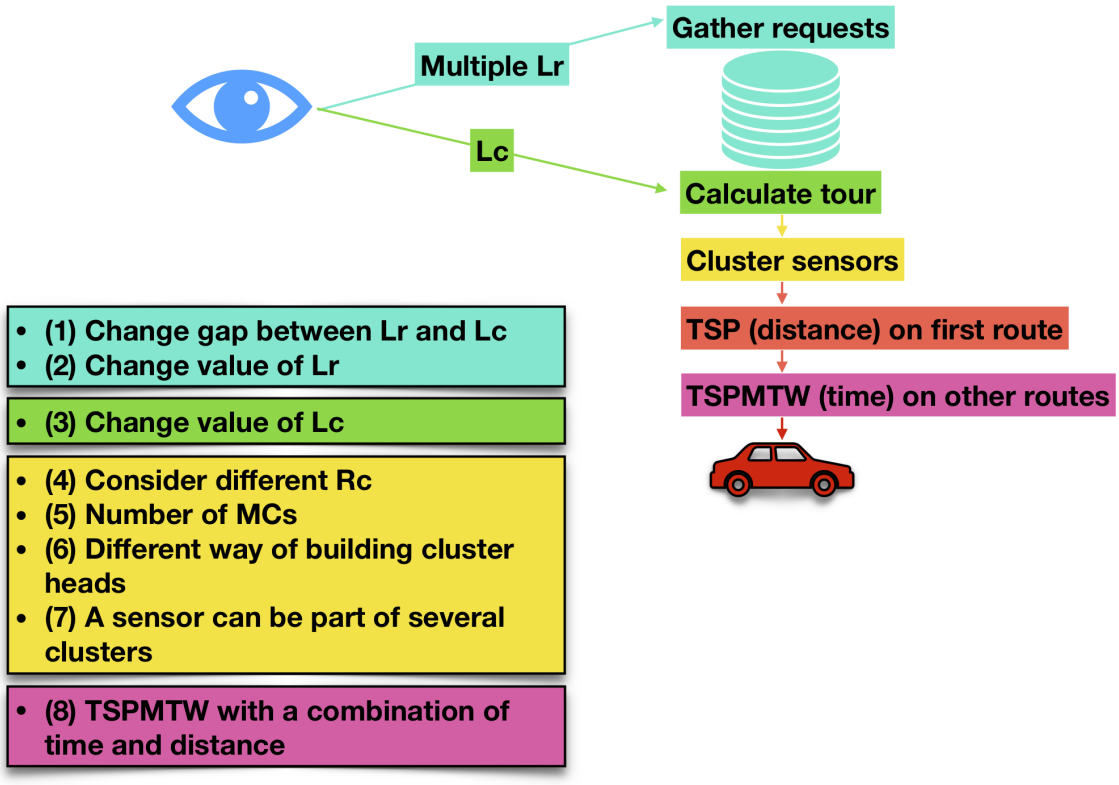

Figure 1 – *Première compréhension du sujet*

B Planification

Toute la planification du projet va se faire au travers d'un système de cartes pouvant se retrouver dans 3 états différents :

- Non commencée
- En cours
- Terminée

Ce procédé peut sembler très différent de l'utilisation d'un diagramme de Gantt mais est en faite assez similaire si l'on oubli la partie présentation. En effet, on découpe notre projet en une suite de tâches qui devront être réalisées dans un ordre donné. Chacune d'entre elles va comporter une date de fin qui sera à respecter et peut éventuellement contenir des sous-tâches plus précises.

Toutes ces tâches peuvent elles-même être regroupées dans une « milestone » permettant d'avoir une vue plus globale sur les tâches à réaliser pour une partie donnée.

Ce choix me parait être intéressant car on retrouve toujours la notion du temps tout comme dans un Gantt mais il est plus facile d'insérer de nouvelles tâches au besoin du fait que ce système met l'accent sur les dates de fin au lieu de dates de début et fin. De plus de tels outils s'intègrent parfaitement dans les outils VCS modernes tels que GitLab ou GitHub.

Nous allons donc détailler l'outil qui a été utilisé (GitLab) puis décrire les différentes tâches identifiées.

1 Outils

Commençons tout d'abord par présenter le système de cartes qui a été utilisé. Ce dernier est proposé par GitLab, un site web mettant à disposition des dépôts Git permettant de versionner son code pour garder les traces de toutes modifications.

Le plus souvent ces entreprises mettent aussi à disposition un système de gestion de « tickets ».

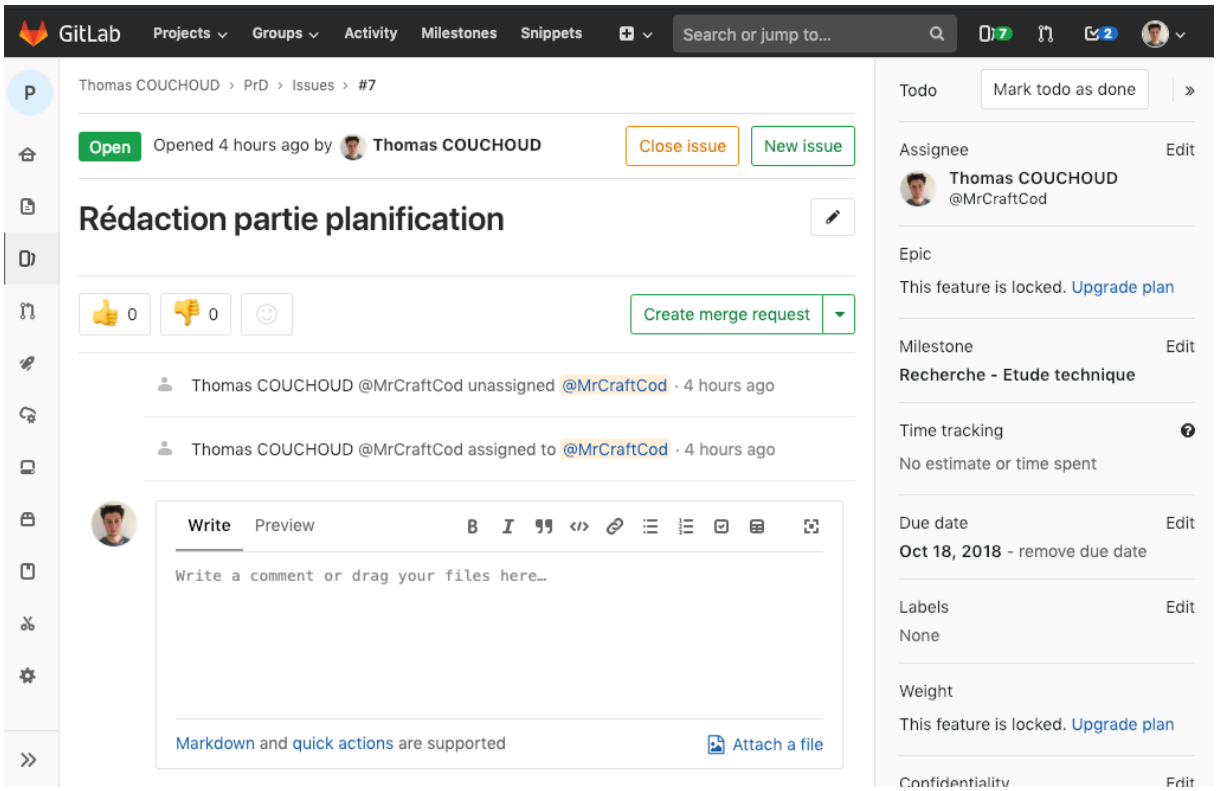

Figure 1 – *Exemple d'une issue*

Un ticket représente une tâche à effectuer et comporte plusieurs éléments la définissant :

- Titre.
- Description : Permet de décrire plus précisément ce qui est attendu.
- Assignee : La personne assignée sur la tâche. Dans notre cas cela n'a pas trop de sens puisque je suis tout seul mais nous utiliserons ce système pour différencier une tâche non commencée (non assignée) d'une tâche en cours (assignée).
- Milestone : Représente l'étape du projet qui englobe la tâche. Nous verrons par la suite le tableau de gestion des tâches par milestone.
- Time tracking : Permet d'indiquer le temps que l'on a passé sur une tâche. Cela peut être utile pour estimer les prochaines tâches ou tout simplement vérifier que le temps passé sur une tâche ne devient pas excessif.
- Due date : La date due.
- Label : Représente une liste de labels que l'on peut appliquer afin de catégoriser la tâche.

De cette manière on peut organiser nos tâches tout en laissant une flexibilité sur l'ordre dans lesquels elles sont réalisées (sauf si une tâche en requiert une autre), du temps que les tâches sont réalisées avant la date de fin.

De plus le regroupement de nos tâches en milestones nous permet d'avoir une vue plus focalisé sur le travail que nous avons à faire à un moment donné. La vue ressemble à l'image ci-dessous :

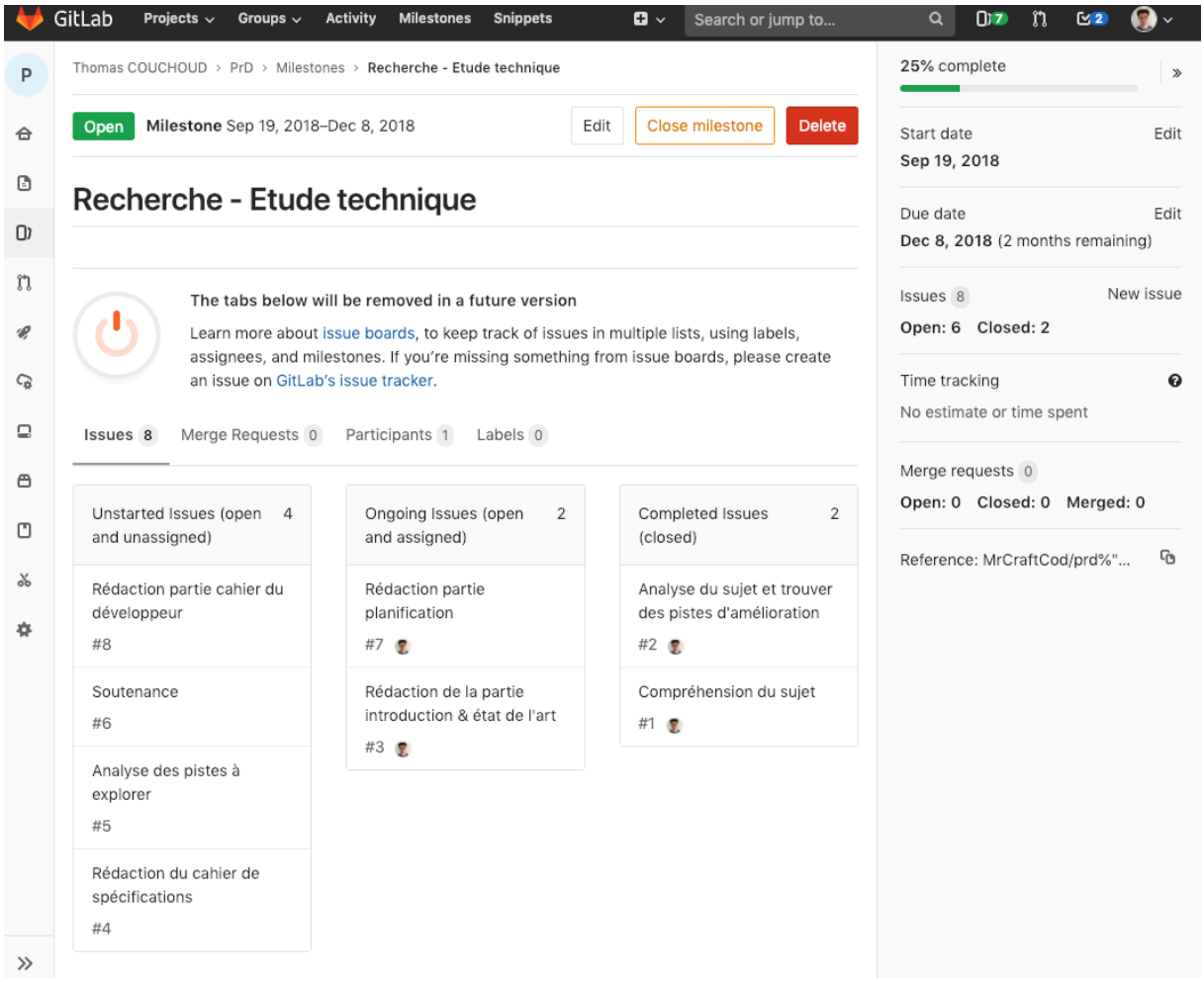

Figure 2 – *Vue milestone*

Comme on peut le voir, la plus grosse partie représente les différentes tâches d'une étape. Cela est très pratique pour avoir une vue générale de ce qu'il reste à faire.

2 Découpage des tâches

2.1 Gestion de projet et version

- **Description de la tâche** :

  Cette tâche à pour but de déterminer la méthode et les outils qui seront utilisés pour la gestion de projet.

- **Estimation de charge** :

  1 jour/homme.

2.2 Compréhension des objectifs et du contexte

- **Description de la tâche** :

  Cette tâche à pour but de prendre en main le sujet en découvrant le papier déjà réalisé et prendre connaissance des objectifs.

- **Estimation de charge** :

  4 jour/homme.

2.3 Rédaction du cahier de spécification

- **Description de la tâche** :

  Cette tâche à pour but de rédiger la partie spécification du rapport. Ce dernier sera amélioré au cours du projet.

- **Estimation de charge** :

  4 jour/homme.

2.4 Etudier l'existant

- **Description de la tâche** :

  Cette tâche à pour but de rentrer plus en détail dans le papier de Mme Rault tout en analysant les resources utilisées afin d'envisager des premières pistes d'amélioration.

- **Estimation de charge** :

  3 jour/homme.

2.5 Etat de l'art

- **Description de la tâche** :

  Cette tâche à pour but de rédiger l'état de l'art afin de rendre compte des méthodes utilisés dans le papier.

- **Estimation de charge** :

  3 jour/homme.

2.6 Analyse des améliorations

- **Description de la tâche** :

  Cette tâche à pour but de présenter une analyse des améliorations possibles de l'algorithme de génération des tournées.

- **Estimation de charge** :

  10 jour/homme.

2.7 Analyse du simulateur

- **Description de la tâche** :

  Cette tâche à pour but de présenter une première analyse de la conception du simulateur.

- **Estimation de charge** :

  5 jour/homme.

2.8 Finalisation du rapport S9

- **Description de la tâche** :
  Cette tâche à pour but de finaliser le rapport pour le S9.
- **Estimation de charge** :
  5 jour/homme.
- **Livrable** :
  Rapport final du S9.
- **Date limite** :
  10/12/2018

2.9 Préparation de la soutenance S9

- **Description de la tâche** :
  Cette tâche à pour but de préparer la soutenance du S9.
- **Estimation de charge** :
  2 jour/homme.
- **Livrable** :
  Exposé PowerPoint.
- **Date limite** :
  12/12/2018

2.10 Réalisation du simulateur

- **Description de la tâche** :
  Cette tâche à pour but de réaliser le simulateur. Elle sera effectuée en plusieurs sprints.
- **Estimation de charge** :
  16 jour/homme.
- **Livrable** :
  Plusieurs versions de l'application.

2.11 Expérimentation des solutions proposées

- **Description de la tâche** :
  Cette tâche à pour but de réaliser les tests sur les améliorations proposées précédemment.
- **Estimation de charge** :
  4 jour/homme.

2.12 Interprétation des résultats expérimentaux

- **Description de la tâche** :

  Cette tâche à pour but d'analyser les résultats obtenus et les mettre en parallèle avec la solution d'origine.

- **Estimation de charge** :

  4 jour/homme.

2.13 Finalisation du rapport S10

- **Description de la tâche** :

  Cette tâche à pour but de finaliser le rapport du S10.

- **Estimation de charge** :

  6 jour/homme.

- **Livrable** :

  Rapport final du S10.

- **Date limite** :

  ? ?/04/2018

  Change date when known

2.14 Préparation de la soutenance du S10

- **Description de la tâche** :

  Cette tâche à pour but de préparer la soutenance du S10.

- **Estimation de charge** :

  3 jour/homme.

- **Livrable** :

  Exposé PowerPoint.

- **Date limite** :

  ? ?/04/2019

  Change date when known

C Spécifications fonctionnelles

Nous allons dans cette partie introduire les fonctions principales qui peuvent être utilisées dans le simulateur.

1 Importation des paramètres

1. **Identification** :

   Cette fonction permet de lire le fichier d'entrée contenant les paramètres de la simulation.

2. **Présentation** :

   - Nom : SimulationParameters.fromJSON

   - Priorité : Primordiale

3. **Description** :

   Afin de pouvoir personnaliser la simulation on laisse l'utilisateur fournir un fichier de configuration. Ce dernier permet de préciser le nombre, type et comportement de différents éléments de la simulation. Plus de détails sur le format du fichier en Section 1(Annexe H).

4. **Description précisée** :

   - Entrée : Le fichier à lire.

   - Sortie : Un objet SimulationParameters initialisé.

   - Exception : Si le fichier n'existe pas, est illisible ou contient une configuration invalide.

2 Module de simulation

1. **Identification** :

   Cette fonction permet de lancer une simulation afin d'obtenir des métriques.

2. **Présentation** :

   - Nom : Simulator.run

   - Priorité : Primordiale

3. **Description** :

   La simulation aura pour but de simuler un environnement de capteurs avec les différents chargeurs mobiles. Cette partie devra prendre en compte la configuration afin de lancer la simulation voulue. De plus les implémentations personnalisées (capteurs, chargeurs, méthode de routage) devront être prises en compte.

4. **Description précisée** :

   - Entrée : Un objet SimulationParameters.
   - Sortie : $\emptyset$.
   - Exception : $\emptyset$.

3 Visualisation des résultats

1. **Identification** :

   Cette partie permet de générer et afficher les résultats sous forme de graphiques dans une fenêtre ou dans un fichier image.

2. **Présentation** :

   - Nom : -
   - Priorité : Modéré

3. **Description** :

   Afin de pouvoir observer les résultats on envisage d'afficher dans un premier temps d'exporter les valeurs observées dans un fichier CSV puis traiter ces données pour en obtenir des graphiques.

4. **Description précisée** :

   - Entrée : Les données de la métrique enregistré.
   - Sortie : Un graphique / CSV.
   - Exception : $\emptyset$.

4 Collecte de métriques

1. **Identification** :

   Cette fonction permet d'être à l'écoute des changements dans le système afin de pouvoir les enregistrer.

2. **Présentation** :

   - Nom : MetricEventDispatcher.dispatch
   - Priorité : Primordiale

3. **Description** :

   Afin d'être flexible, on propose de faire un système d'écoute de changement des valeurs du système. Par la suite des listeners viendront écouter ces changement pour en tirer des interprétations ou faire de l'archivage.

4. **Description précisée** :

- Entrée : Un évènement de changement de l'état du système.
- Sortie : $\emptyset$.
- Exception : $\emptyset$

D Diagramme de classes

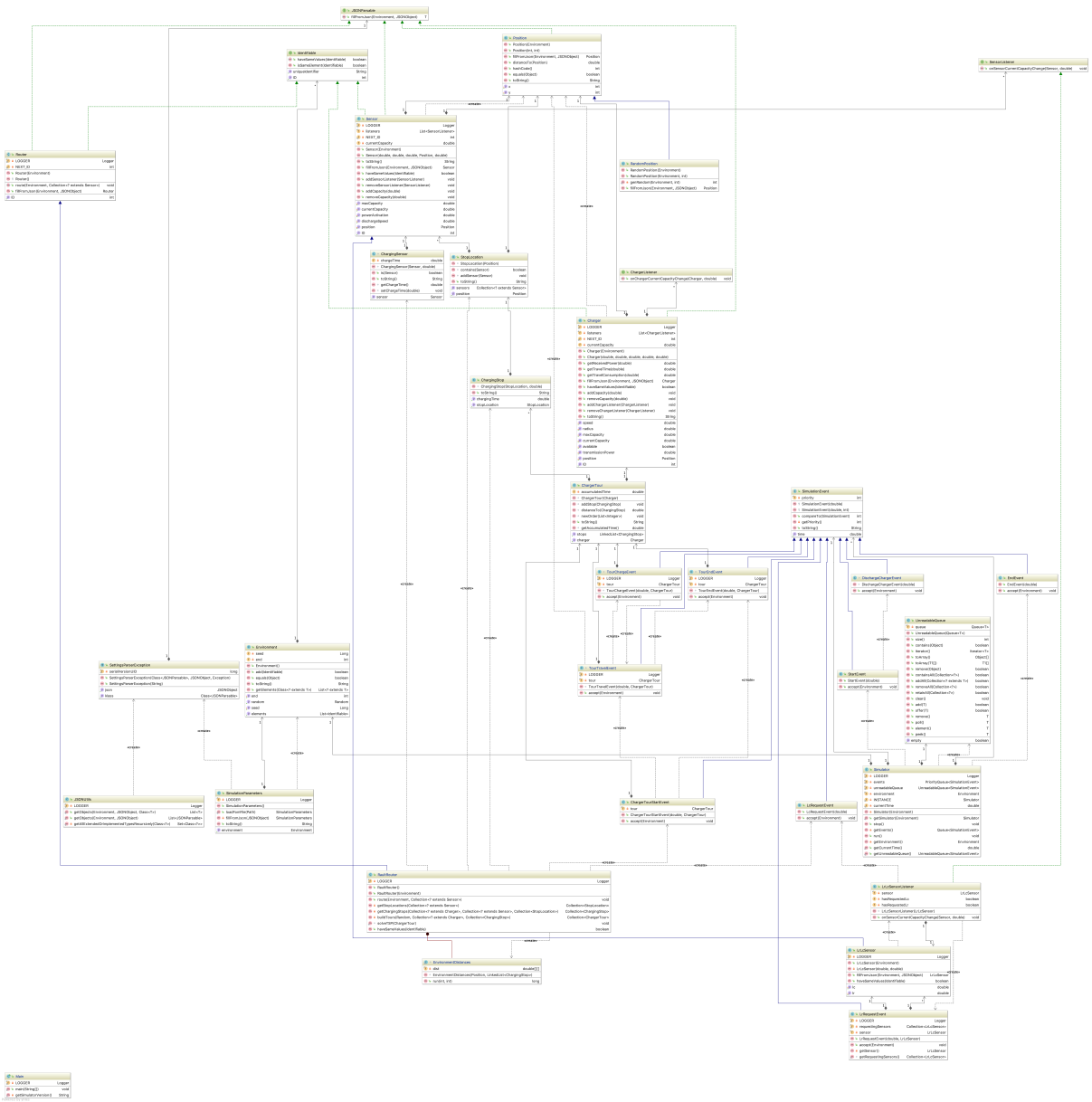

Figure 1 – *Diagramme de classes complet*

E Spécifications non fonctionnelles

1 Analyse de l'algorithme

D'un point de vue algorithmique, des améliorations de l'algorithme écrit par Mme Rault [12] devront être proposés et détaillés.

Ces dernières seront par la suite testées avec le simulateur et conduiront à une conclusion quant à l'efficacité de ces dernières.

On considérera qu'un algorithme améliore une solution si le temps de rechargement est diminuée sans altérer grandement les autres métriques. Cette amélioration sera à noter par instance étant donné que selon la configuration de départ, le déroulement des tournées peut grandement changer.

Les instances à utiliser seront celles décrites dans le papier de Mme Rault [12]. Certaines d'entre elles pourront comporter des valeurs aléatoires qui seront définies dans un interval. Dans ce cas la simulation sera jouée plusieurs fois et nous garderons la moyenne des résultats comme étant notre résultat final.

2 Contraintes de développement et conception

- Language : Java 11
- Plateformes cibles : Windows, OSX, Linux

3 Contraintes de fonctionnement et d'exploitation

3.1 Performances

- Du point de vue environnement, le programme doit être utilisable dans la majorité des systèmes d'exploitation. Grâce à la JVM cela devrait être transparent au niveau du codage.
- D'un point de vue utilisateur il faut que le programme réalise la simulation dans un temps non infini. Il faut donc un système permettant de limiter le nombre d'itération du programme. On propose une méthode les limitant dans le temps en définissant une date d'arrêt.

3.2 Capacité

- Afin de pouvoir obtenir une vision détaillé du processus de la simulation, on met à disposition les logs du programme au travers d'un fichier. Pour éviter de saturer l'espace mémoire, on limite les fichiers de logs à 10Mb et ne gardons que les 10 derniers logs.
- De plus, le grand nombre d'itérations entraine un grand nombre de données accumulées. Il sera donc nécessaire de ne pas tout garder en mémoire RAM afin d'éviter des dépassements.

3.3 Contrôlabilité

- La simulation doit produire les même résultats pour la même configuration. Dans le cas d'un algorithme incluant un choix aléatoire on proposera à l'utilisateur de configurer la graine de génération de ces nombres. Cela permet à l'utilisateur de reproduire une simulation et analyser les logs si les résultats paraissent aberrants.

F Cahier du développeur

S10

G Document d'installation du projet

S10

1 Utilisation simple

2 Utilisation développeur

H Document d'utilisation du projet

1 Fichier de configuration

Le fichier d'entrée du simulateur doit être sous le format JSON. Nous aurons donc une structure semblable à celle-ci :

```
{
  "seed":  42,
  "end": 20,
  "environment": [
    {
      "class": "fr.mrcraftcod.simulator.sensors.Sensor",
      "count": 1,
      "parameters": {
        "powerActivation": 3,
        "position": {
          "class": "fr.mrcraftcod.simulator.positions.RandomPosition",
          "parameters": {
            "max": 5
          }
        },
        "maxCapacity": 40,
        "currentCapacity": 23
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.chargers.Charger",
      "count": 3,
      "parameters": {
        "transmissionPower": 3,
        "radius": 40.5,
        "maxCapacity": 500,
        "currentCapacity": 450,
        "speed": 12
      }
    }
  ]
}
```

La racine du fichier est un objet JSON contenant plusieurs éléments :

- **seed** : Correspond à la graine utilisée pour la génération de nombres aléatoires. Si cette dernière n'est pas précisée, une graine basée sur le temps actuel sera utilisée.
- **end** : La date de fin de la simulation.
- **environment** : Une liste d'objets définissant les objets présents dans notre simulation. Ces derniers seront décrits plus en détail en Section 1.1.

1.1 Environment

Les objets présents dans cette liste peuvent être très variés. De base l'application propose quelques capteurs et quelques chargeurs mais il est aussi possible d'utiliser certains objets qui ont étés créés par la suite afin de traiter le problème avec un algorithme différent. Cependant tous ces objets suivent la même forme :

- **class** : La classe de l'objet dans le code JAVA.
- **count** : Le nombre d'objets de ce type à créer avec les paramètres donnés (si les paramètres doivent être différents pour chaque éléments, il faut que la classe JAVA gère la génération d'un nombre aléatoire dans un intervalle (qui sera passé en paramètres) ou il faut déclarer plusieurs objets dans le JSON).
- **parameters** : Les paramètres pour initialiser l'objet. Ces derniers sont différents pour chaque éléments que nous voulons utiliser (ceux fournis de base sont décris en Section 1.2).

1.2 Paramètres

1.2.1 Chargers

fr.mrcraftcod.simulator.chargers.Charger

- **radius** : Nombre réel, le rayon de rechargement du chargeur.
- **transmissionPower** : Nombre réel, la puissance de transmission.
- **maxCapacity** : Nombre réel, la capacité maximale de la batterie.
- **currentCapacity** : Nombre réel, la capacité de départ de la batterie.
- **speed** : La vitesse de déplacement.

1.2.2 Capteurs

fr.mrcraftcod.simulator.sensors.Sensor

- **powerActivation** : Nombre réel, la puissance minimale pour activer le rechargement.
- **position** : objet JSON (Section 1.2.4), la position.
- **maxCapacity** : Nombre réel, la capacité maximale de la batterie.
- **currentCapacity** : Nombre réel, la capacité de départ de la batterie.
- **dischargeSpeed** : La vitesse de déchargement.

fr.mrcraftcod.simulator.rault.sensors.LrLcSensor

- Pareil que fr.mrcraftcod.simulator.sensors.Sensor (Section 1.2.2).
- **lc** : Nombre réel, la valeur de lc.
- **lr** : Nombre réel, la valeur de lr.

1.2.3 Routing

fr.mrcraftcod.simulator.rault.routing.RaultRouting

- $\emptyset$

1.2.4 Position

fr.mrcraftcod.simulator.positions.Position

- **x** : Nombre entier, la position x.
- **y** : Nombre entier, la position y.

fr.mrcraftcod.simulator.positions.RandomPosition

- **max** : Nombre entier, la borne maximum pour la génération aléatoire. Les valeurs x et y seront telles que $x, y \in [-max, max[$.

2 Fichier de sortie

Nous retenons deux types de fichiers de sortie. Les premiers sont des images représentant des graphes de l'évolution de différentes métriques. Ces dernières pourront êtres définies par l'utilisateur dans le code mais certaines seront fournies de base :

- Pourcentage de batterie des capteurs.
- Nombre de capteurs déchargés.
- Nombre de chargeurs utilisés.
- Pourcentage d'utilisation des chargeurs.
- Nombre de requêtes Lr en attente.
- Temps de calcul de la tournée.

De plus ces données seront aussi fournies sous un format CSV afin d'avoir un accès aux valeurs précises et pouvoir les exploiter dans un autre programme ou bien un tableur.

I Cahier des tests

S10

J Comptes rendus hebdomadaires

Compte rendu n°1 du 23/09/2018

- Reception et lecture du papier de Mme Rault [12].
- Echange avec Mme Rault sur le sujet.
- Lecture de certaines références.
- Réalisation d'un petit résumé de la compréhension ainsi que des premières améliorations possibles (Figure 1(Annexe A)).

Compte rendu n°2 du 30/09/2018

Avancement:

- Mise en place de Biber pour le rapport.
- Début du rapport avec notamment la partie contexte & état de l'art.

Questions:

- Découpage des différentes parties dans le rapport étant donné qu'il est plus axé recherche.

Compte rendu n°3 du 07/10/2018

Avancement:

- Ajout des parties manquantes dans l'introduction.
- Ecriture de la partie « Description générale ».
- Début du rapport avec notamment la partie contexte & état de l'art.

Questions:

- Découpage des différentes parties dans le rapport étant donné qu'il est plus axé recherche.

Compte rendu n°4 du 14/10/2018

Avancement:

- Avancement sur la partie contexte & état de l'art.

Questions:

- Language de codage imposé?

Compte rendu n°5 du 21/10/2018

Avancement:

- Finalisation contexte et état de l'art.
- Début d'un peu de code (parsing fichier entrée) pour se changer les idées.
- Rédaction des spécifications.

Compte rendu n°6 du 28/10/2018

Avancement:

- Avancement sur la partie analyse.

Compte rendu n°7 du 11/10/2018

Avancement:

- Ecriture de l'algorithme Algorithme 1(Chapitre 4).
- Test de la librairie OR-Tools avec un TSP.
- Début d'implémentation de l'algorithme de mme Rault [12] dans le simulateur.

Compte rendu n°8 du 18/10/2018

Avancement:

- Analyse logicielle.
- Avancement dans le code.

Compte rendu n°9 du 25/10/2018

Avancement:

- Continuation de l'implémentation de l'algorithme de Mme Rault [12] dans le simulateur.

Compte rendu n°10 du 02/11/2018

Avancement:

- Adaptation du rapport suite à une échange avec Mme Rault.
- Début d'ajout d'une métrique dans le simulateur.

Compte rendu n°11 du 09/11/2018

Avancement:

- Finalisation du rapport pour le S9.
- Création du support de présentation pour la soutenance.

K Bibliographie

[1] H. Dai, Y. Liu, G. Chen, X. Wu et T. He. « SCAPE : Safe Charging with Adjustable Power ». In : *2014 IEEE 34th International Conference on Distributed Computing Systems*. Juin 2014, p. 439-448. DOI : 10.1109/ICDCS.2014.52.

[2] H. Dai, Y. Liu, G. Chen, X. Wu, T. He, A. X. Liu et H. Ma. « Safe Charging for Wireless Power Transfer ». In : *IEEE/ACM Transactions on Networking* 25.6 (déc. 2017), p. 3531-3544. ISSN : 1063-6692. DOI : 10.1109/TNET.2017.2750323.

[3] H. Dai, Y. Liu, A. X. Liu, L. Kong, G. Chen et T. He. « Radiation constrained wireless charger placement ». In : *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. Avr. 2016, p. 1-9. DOI : 10.1109/INFOCOM.2016.7524385.

[4] G. Jiang, S. Lam, Y. Sun, L. Tu et J. Wu. « Joint Charging Tour Planning and Depot Positioning for Wireless Sensor Networks Using Mobile Chargers ». In : *IEEE/ACM Transactions on Networking* 25.4 (2017), p. 2250-2266. ISSN : 1063-6692. DOI : 10.1109/TNET.2017.2684159.

[5] Lintong Jiang, Xiaobing Wu, Guihai Chen et Yuling Li. « Effective On-Demand Mobile Charger Scheduling for Maximizing Coverage in Wireless Rechargeable Sensor Networks ». In : *Mobile Networks and Applications* 19.4 (août 2014), p. 543-551. ISSN : 1572-8153. DOI : 10.1007/s11036-014-0522-y. URL : https://doi.org/10.1007/s11036-014-0522-y.

[6] Lyes Khelladi, Djamel Djenouri, Michele Rossi et Nadjib Badache. « Efficient on-demand multi-node charging techniques for wireless sensor networks ». In : *Computer Communications* 101 (2017), p. 44-56. ISSN : 0140-3664. DOI : https://doi.org/10.1016/j.comcom.2016.10.005. URL : http://www.sciencedirect.com/science/article/pii/S0140366416304182.

[7] Chi Lin, Guowei Wu, Mohammad S. Obaidat et Chang Wu Yu. « Clustering and splitting charging algorithms for large scaled wireless rechargeable sensor networks ». In : *Journal of Systems and Software* 113 (2016), p. 381-394. ISSN : 0164-1212. DOI : https://doi.org/10.1016/j.jss.2015.12.017. URL : http://www.sciencedirect.com/science/article/pii/S0164121215002836.

[8] Chi Lin, Youkun Wu, Zhicheng Liu, Mohammad S. Obaidat, Chang Wu Yu et Guowei Wu. « GTCharge : A game theoretical collaborative charging scheme for wireless rechargeable sensor networks ». In : *Journal of Systems and Software* 121 (2016), p. 88-104. ISSN : 0164-1212. DOI : https://doi.org/10.1016/j.jss.2016.08.046. URL : http://www.sciencedirect.com/science/article/pii/S0164121216301480.

[9] Adelina Madhja, Sotiris Nikoletseas et Theofanis P Raptis. « Distributed wireless power transfer in sensor networks with multiple mobile chargers ». In : *Computer Networks* 80 (2015), p. 89-108.

[10] Sotiris Nikoletseas, Theofanis P. Raptis et Christoforos Raptopoulos. « Radiation-constrained algorithms for Wireless Energy Transfer in Ad hoc Networks ». In : *Computer Networks* 124 (2017), p. 1-10. ISSN : 1389-1286. DOI : https://doi.org/10.1016/j.comnet.2017.05.025. URL : http://www.sciencedirect.com/science/article/pii/S1389128617302323.

[11] Gilles Pesant, Michel Gendreau, Jean-Yves Potvin et Jean-Marc Rousseau. « On the flexibility of constraint programming models : From single to multiple time windows for the traveling salesman problem ». In : *European Journal of Operational Research* 117.2 (1999), p. 253-263. ISSN : 0377-2217. DOI : https://doi.org/10.1016/S0377-2217(98)00248-3. URL : http://www.sciencedirect.com/science/article/pii/S0377221798002483.

[12] Tifenn Rault. « Avoiding radiation of on-demand multi-node energy charging with multiple mobile chargers ».

[13] X. Ren, W. Liang et W. Xu. « Maximizing charging throughput in rechargeable sensor networks ». In : *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*. Août 2014, p. 1-8. DOI : 10.1109/ICCCN.2014.6911792.

[14] *Travelling salesman problem*. URL : https://en.wikipedia.org/wiki/Travelling_salesman_problemhttps://en.wikipedia.org/wiki/Travelling_salesman_problem.

[15] C. Wang, J. Li, F. Ye et Y. Yang. « Multi-vehicle Coordination for Wireless Energy Replenishment in Sensor Networks ». In : *2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS 2013)(IPDPS)*. T. 00. Mai 2013, p. 1101-1111. DOI : 10.1109/IPDPS.2013.22. URL : https://doi.ieeecomputersociety.org/10.1109/IPDPS.2013.22.

[16] L. Xie, Y. Shi, Y. T. Hou, W. Lou, H. D. Sherali et S. F. Midkiff. « On renewable sensor networks with wireless energy transfer : The multi-node case ». In : *2012 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*. Juin 2012, p. 10-18. DOI : 10.1109/SECON.2012.6275766. URL : https://ieeexplore.ieee.org/abstract/document/6275766/metrics#metrics.

L Acronymes

EMR electromagnetic radiation. 2, 8, 9, 12, 14

ILP Integer Linear Programming. 11

MC mobile charger. 2, 10, 11, 14, 17

TSP travelling salesman problem. 4, 9, 11, 12
TSPMTW travelling salesman problem with multiple time windows. 4, 10, 12, 14, 17, 23

WET wireless energy transfer. 1

Amélioration d'un protocole de rechargement de capteurs

Résumé

Projet de fin d'études visant à améliorer un algorithme de recharge de capteurs. Ce dernier devra prendre en compte le rechargement à la demande point-à-multipoint et plusieurs chargeurs tout en évitant les problèmes liés à l'agrégation des champs électromagnétiques. Les améliorations proposées seront ensuite testées et étudiées dans un simulateur qui sera conçu.

Mots-clés

radiation électromagnétique, programmation en nombres entiers, chargeur mobile, capteur, problème du voyageur de commerce, voyageur de commerce à fenêtre multiples, transfer d'énergie sans contact, PRD

Abstract

Final year project aiming to improve an algorithm to charge sensors. It should take into account the on point-to-multipoint demand charging and the problematic of the aggregation of electromagnetic radiations. Those improvements will be tested in a simulator that will be created.

Keywords

electromagnetic radiations, integer linear programming, mobile charger, sensor, travelling salesman problem, travelling salesman problem with multiple time windows, wireless energy transfer, PRD

Tuteurs académiques
Tifenn Rault
Ronan Bocquillon

Étudiant
Thomas Couchoud (DI5)