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

Tuteur académique
Tifenn Rault

Étudiant
Thomas Couchoud (DI5)

26 mars 2019

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

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 susnommé le tuteur académique.

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 du tuteur académique 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` V2.4.0(2019/03/21)

Table des matières

Table des figures

Liste des tableaux

Liste des codes sources

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'adresser 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 pré-configurée. 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 paralléliser 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 langages Python, C++, C# ou bien Java ce qui permet de ne pas bloquer notre choix du langage.

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 langage 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 personnaliser 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 fonctionnalités 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).

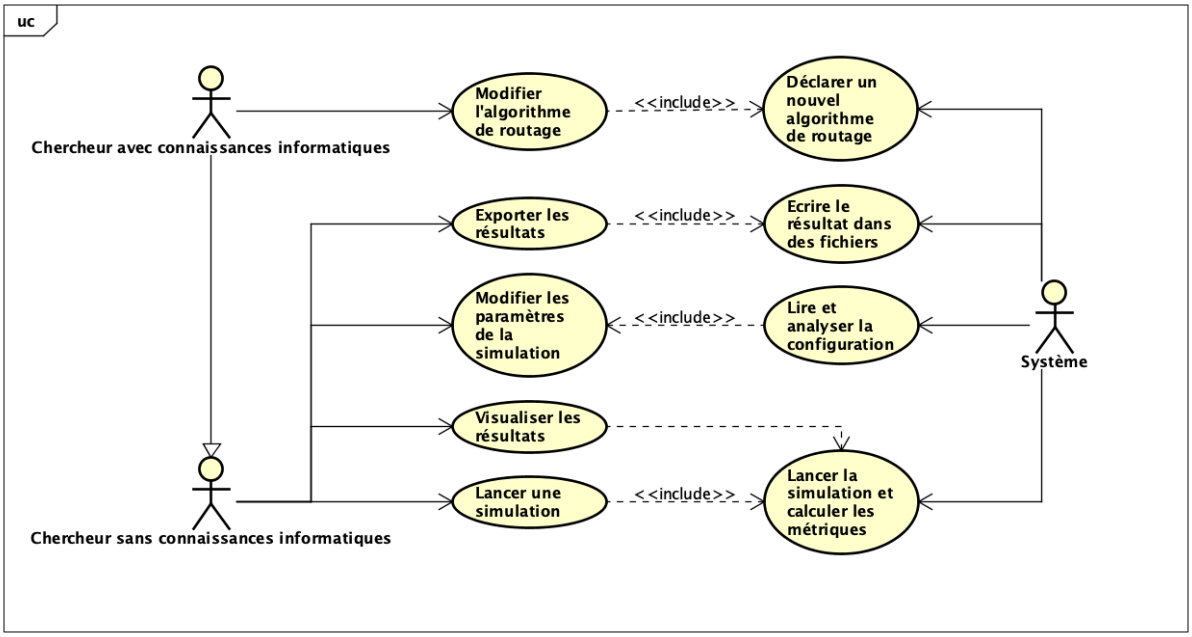


Figure 1 – *Diagramme de cas d'utilisation*

On peut y observer les deux rôles 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 réception.
  - 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 langage 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.

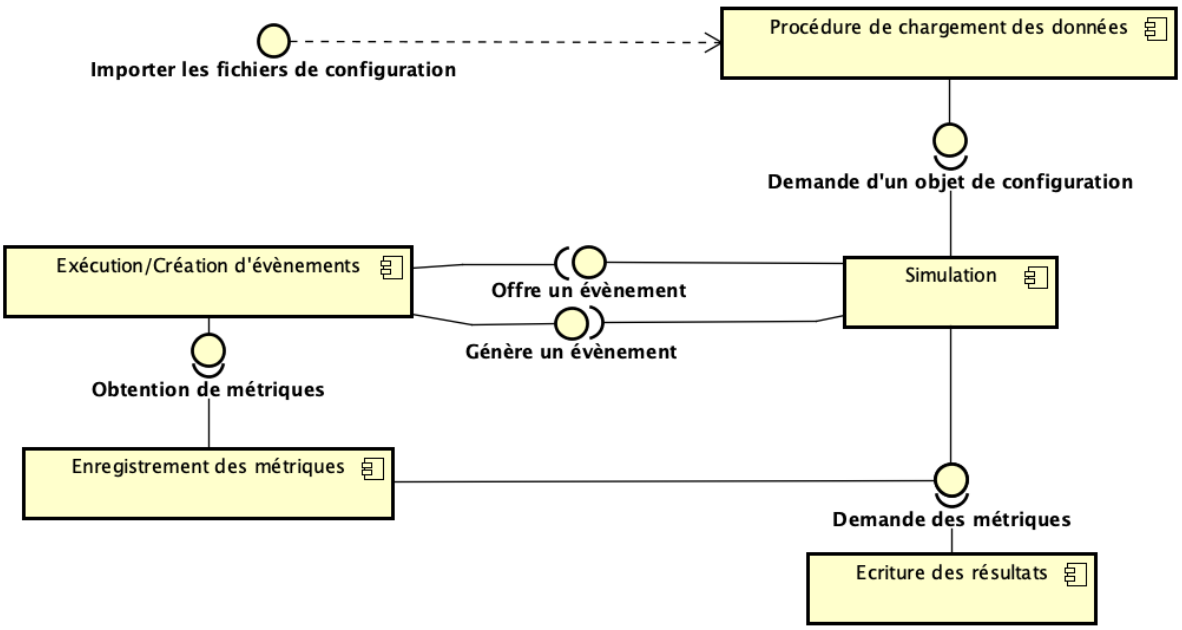


Figure 2 – *Diagramme de composant*

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 occurrence, 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
[15] [7]	non	oui	non	non
[13] [11]	oui	non	non	non
[6]	oui	oui	non	non
[8] [9] [14] [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 :

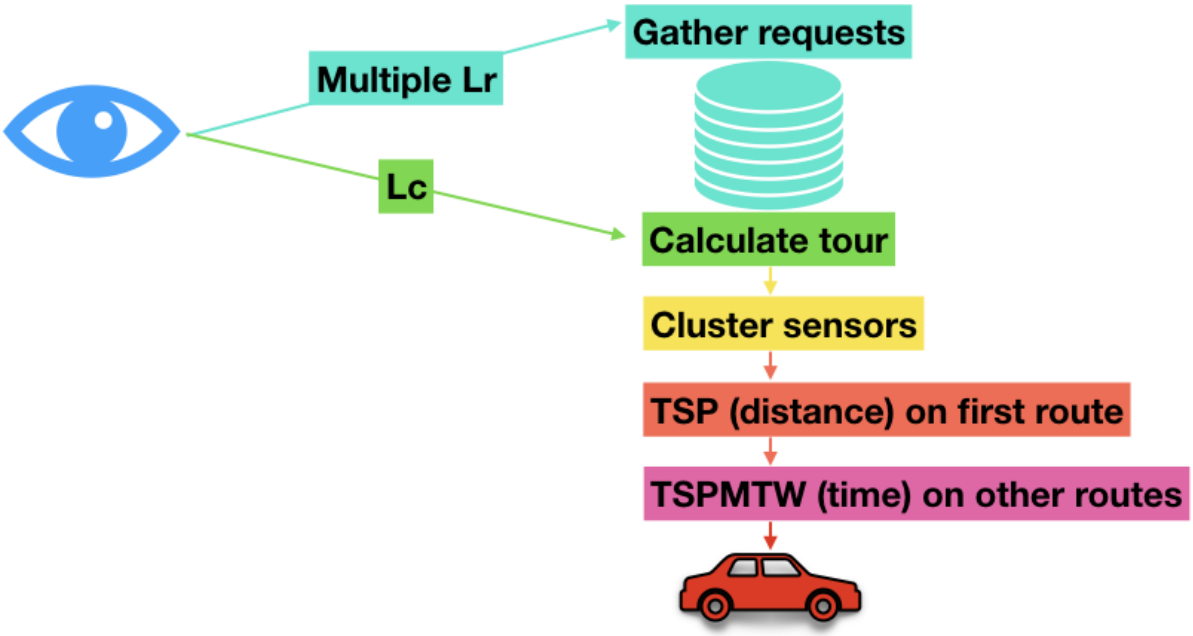


Figure 1 – *Enchainement des étapes de la solution*

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 [WWW1] 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 déterministe 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 modèle 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 doivent 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.

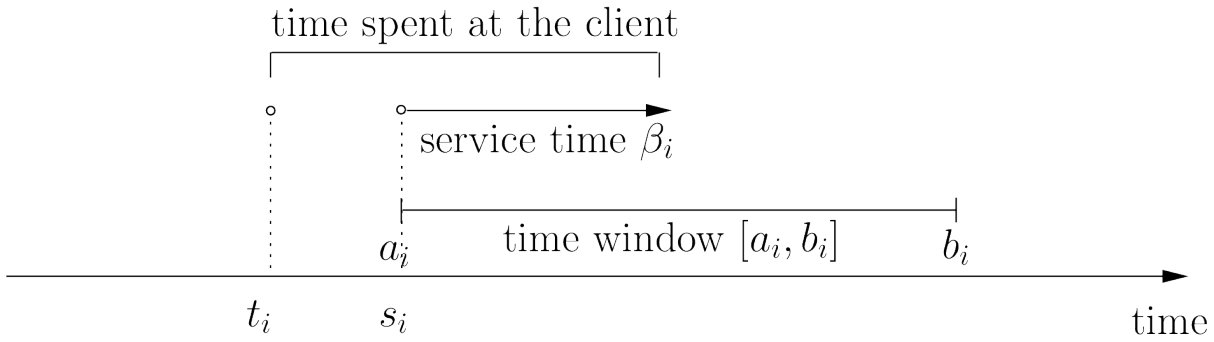


Figure 2 – *Illustration d'une fenêtre de temps (source)*

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'intervalle 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 parallèle. 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.

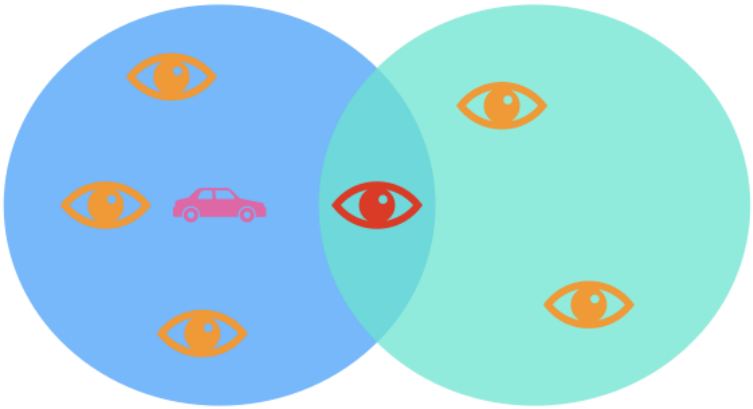


Figure 1 – *Etat A*

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

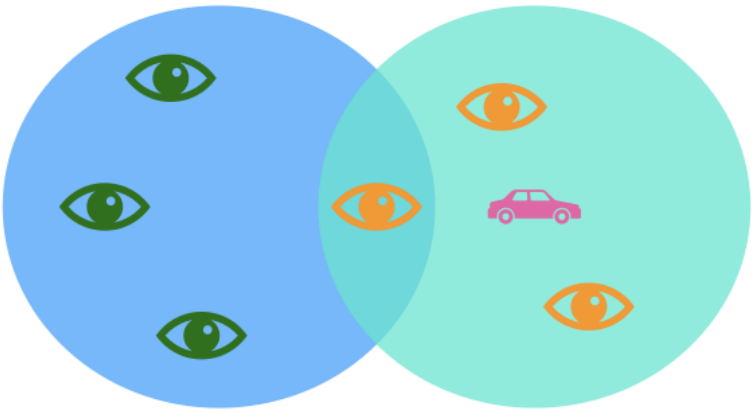


Figure 2 – *Etat B*

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.

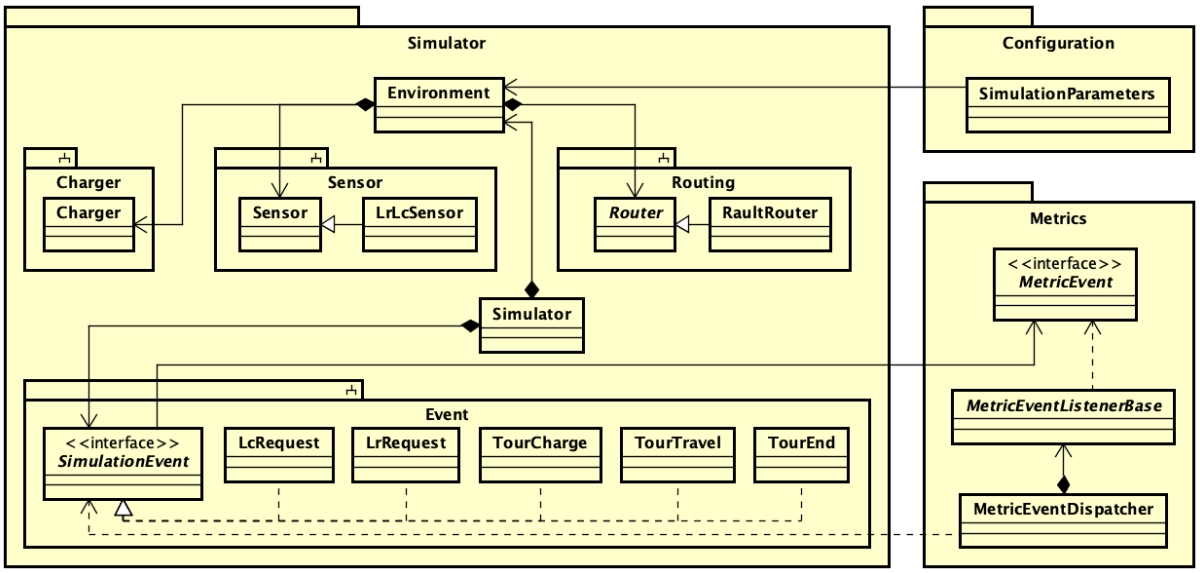


Figure 3 – *Diagramme de classes simplifié*

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

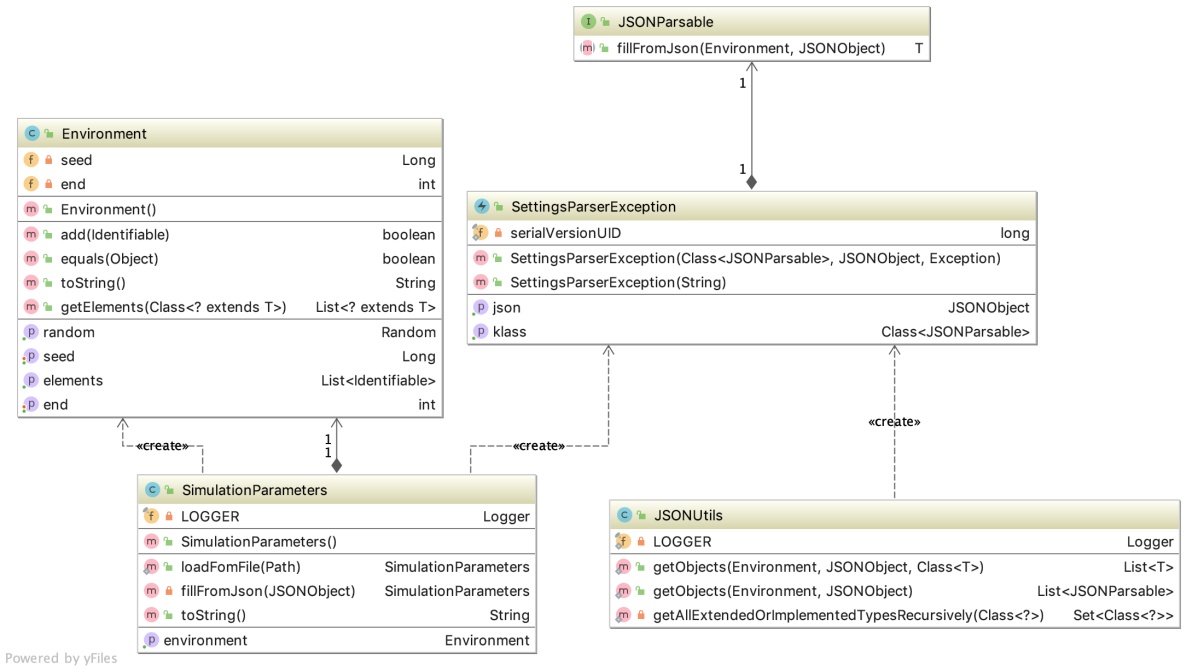


Figure 4 – *Partie configuration*

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

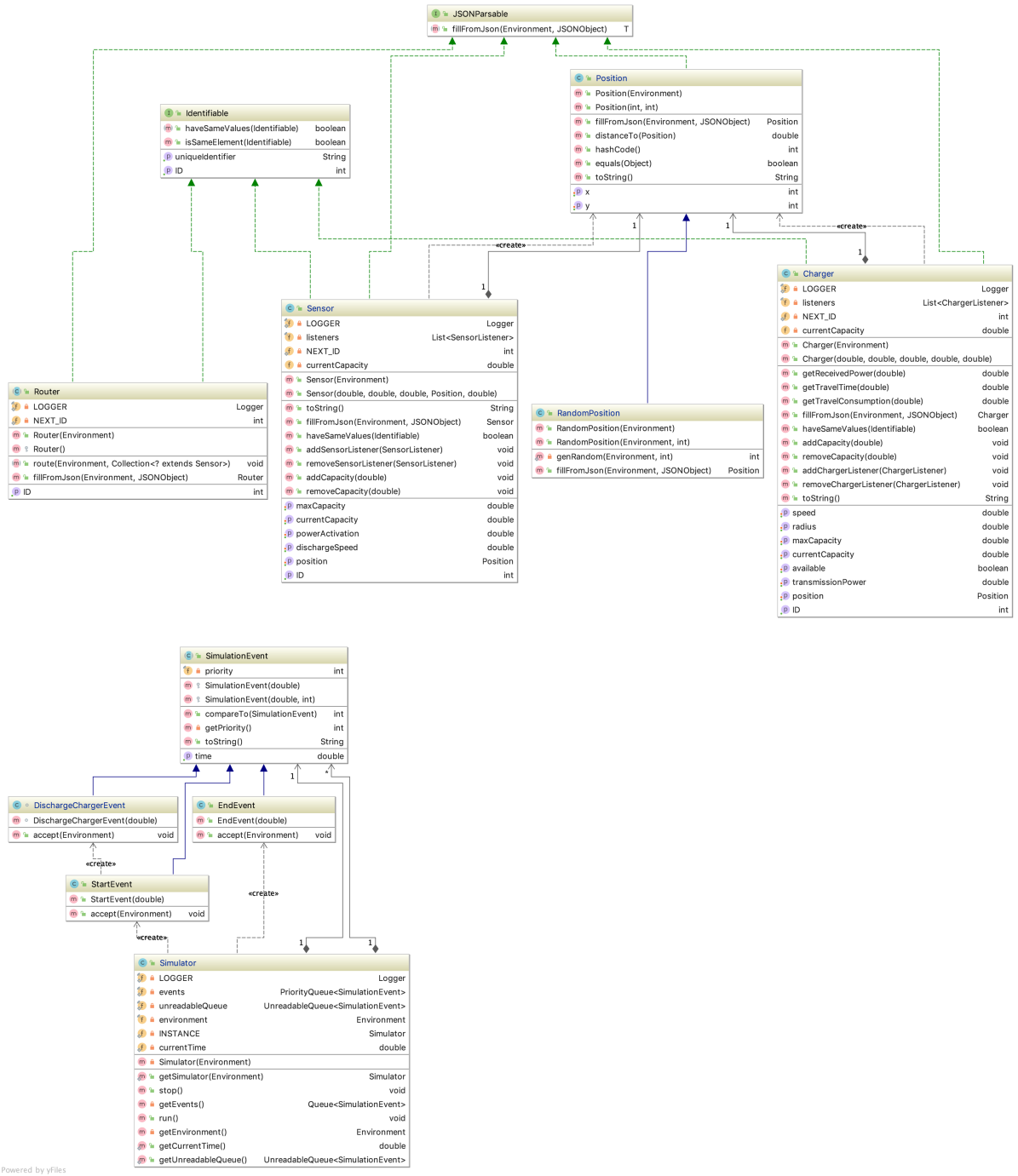


Figure 5 – *Partie simulator*

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 étend 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

Le développement du projet s'est effectué avec la méthode agile. Il n'y a donc pas de Gantt pour cette partie. Tout a été géré au travers d'issues (voir Section 1(Annexe B)) afin de définir les tâches à traiter. Ces dernières ont étés par la suite réparties sur les différents sprints prévus en fonction de leur criticité.

Afin d'améliorer la détection de bugs et la construction de fichiers exécutables, un système d'intégration continue a été mis en place et automatise de nombreuses tâches. Plus d'informations sur cette partie sont disponibles en Annexe J.

Nous allons dans cette partie nous focaliser sur la réalisation du simulateur ainsi que des résultats que nous avons pu obtenir au travers de ce dernier.

1 Simulateur

La réalisation de ce simulateur a été découpée en trois étapes majeures :

- Partie cœur : gestion de la configuration, gestion d'évènements du simulateur, gestion de métriques, ...
- Ajout de l'algorithme de Mme Rault en étendant le cœur (comme si ce dernier était une librairie).
- Implémentation des améliorations proposés à l'algorithme (modification de classes existantes).

1.1 Développement du cœur du simulateur

La partie centrale du simulateur va s'apparenter grandement au diagramme de classes de la Figure 1(Annexe F) (avec quelques implémentations spécifiques en moins).

1.1.1 Lecture de la configuration

Commençons par la lecture du fichier de configuration. En effet ce dernier est notre point d'entré et va définir tout l'environnement de simulation qui sera utilisé par la suite. Les éléments dont nous avons besoin sont les suivants :

- Seed de génération des nombres aléatoires
- Date de fin de la simulation
- Métriques à enregistrer
- Eléments présents dans la simulation

Un des objectifs important de notre configuration est que celle-ci soit capable de décrire des instances possiblement très différentes. De plus elle doit aussi être robuste a de potentiels ajouts d'éléments dans le code.

Pour cela le format JSON a été choisi. Ce dernier permet de définir des structures telles que des listes, objets ou valeurs ce qui est très pratique dans notre cas. Le format de ce fichier coté utilisateur est défini en Section 1(Annexe H). Nous allons ici détailler ce qu'il se passe du point de vue du code.

La lecture des paramètres tels que la seed ou le temps final de la simulation est triviale et ne sera pas décrite. En rechante tous les objets définissants des classes à initialiser méritent une explication. Dans le fichier mentionné plus haut cela concerne notamment les capteurs et les chargeurs.

Le principe est de définir une classe qui sera instanciée avec des paramètres. Pour cela une interface JSONParsable a été crée et définie un élément que nous pouvons parser depuis un objet JSON. Il suffit par la suite d'utiliser la réflexivité afin de rechercher la classe nommée, vérifier qu'elle implémente JSONParseable et appeler la méthode appropriée afin d'initialiser les paramètres de cette instance.

Code source 5.1 – *Création d'un JSONParsable depuis le JSON*

```
1  public static List<JSONParsable> getObjects(final Environment environment, final
       JSONObject elementObj) throws SettingsParserException{
2      final Class<?> elementKlass;
3      try{
4          elementKlass =
                   Class.forName(Optional.of(elementObj.optString("class")).filter(s ->
                   !s.isBlank()).orElseThrow(() -> new IllegalStateException("No class
                   name provided")));
5      }
6      catch(final ClassNotFoundException e){
7          LOGGER.error("Class from {} not found", elementObj, e);
8          throw new IllegalArgumentException("JSON for isn't valid");
9      }
10     if(!getAllExtendedOrImplementedTypesRecursively(elementKlass)
               .contains(JSONParsable.class)){
11         throw new IllegalArgumentException("Element class isn't parsable from JSON");
12     }
13     final var parsableClazz = (Class<JSONParsable>) elementKlass;
14     final var parameters =
               Optional.ofNullable(elementObj.optJSONObject("parameters")).orElse(new
               JSONObject());
15     return IntStream.range(0, Optional.of(elementObj.optInt("count")).filter(i -> i
               > 0).orElse(1)).mapToObj(i -> {
16         try{
17             return (JSONParsable) parsableClazz.getConstructor(Environment.class)
                       .newInstance(environment).fillFromJson(environment, parameters);
18         }
19         catch(final IllegalArgumentException | NoSuchMethodException |
                   IllegalAccessException | InstantiationException |
                   InvocationTargetException e){
20             throw new SettingsParserException(parsableClazz, elementObj, e);
21         }
```

```
22      }).collect(Collectors.toList());
23  }
```

Les lignes 3 à 9 effectuent la recherche de la classe concernée.

Les lignes 10 à 12 vérifient que la classe trouvée implémente bien JSONParseable.

Les lignes 13 à 14 préparent les paramètres qui seront passés à l'instance de la classe.

La ligne 15 va instancier la classe le nombre de fois qui est défini dans le fichier de configuration.

Les lignes 16 à 21 effectuent l'instantiation des classes.

1.2 Eléments de base

Une fois le parser de la configuration réalisé, il a fallu implémenter des objets de base qui seront utilisés lors de la simulation. Dans le cas de notre problème nous retrouvons 3 éléments majeurs :

- Les chargeurs.
- Les capteurs.
- Le système de routage.

Afin de généraliser les éléments cités plus haut, une classe Identifiable a été crée. Son but est de définir un élément de la simulation qui a une importance (contrairement à un objet Position) et donc être capable de l'identifier individuellement. A chaque création de l'un de ces objets, un identifiant unique lui sera attribué et pourra être utilisé dans le code afin de le retrouver.

De plus une généralisation des chargeurs et capteurs a aussi été faite au travers de deux interfaces :

- Rechargeable : indique que l'élément dispose d'une batterie pouvant être chargée ou déchargée.
- Positionable : indique que l'élément dispose d'une position.

Tous les éléments peuvent par la suite étendus par d'autres classes afin d'y rajouter des fonctionnalités. Nous allons ici détailler les implémentations de bases qui ont été réalisées afin d'avoir le minimum nécessaire pour le simulateur.

1.2.1 Capteurs

Les capteurs implémentent toutes les interfaces précédemment citées : Identifiable, Positionable et JSONParseable.

Ils ont de plus une vitesse de décharge ainsi un seuil d'activation pour être rechargé.

1.2.2 Chargeurs

Les chargeurs implémentent toutes les interfaces précédemment citées : Identifiable, Positionable et JSONParseable.

Ils ont de plus un rayon de recharge, une intensité d'émission des ondes EMRs et une vitesse.

1.2.3 Routeur

Un router est juste Identifiable et JSONParseable et définit une méthode route qui sera à implémenter. Il est à noter que le nombre de router par instance est limité à 1 et qu'aucune implémentation de base n'est fournie.

1.3 Simulateur & évènements

Maintenant que nous avons les différent éléments permettant de définir un environnement (instance), il nous faut les utiliser de manière logique de manière à rendre compte de ce qu'il se passerai dans la réalité.

1.3.1 Evènements

Le principe du simulateur repose sur une gestion ordonnée d'évènements. Chaque évènement pourra par la suite modifier l'état des différents Identifiables de la simulation.

Du point de vue du code un évènement contient :

- Une date d'exécution
- Une priorité
- Un code à exécuter

Par défaut 3 évènements sont implémentés :

- Start : initialise des variables et ajoute des évènements décharge et fin dans la file du simulateur.
- Fin : vide la file du simulateur et nettoie le simulateur.
- Décharge : décharge tous les capteurs et ajoute un évènement décharge à $t+1$ dans la file du simulateur.

1.3.2 Simulateur

Le simulateur en lui même effectue toujours les mêmes opérations :

- Initialise son temps d'exécution à 0
- Ajoute un évènement Start dans la file au temps 0
- Tant que des éléments sont présents dans la queue, extraire l'évènement avec le temps le plus faible et si égalité celui avec la priorité la plus faible (en terme de valeur) :
  - Modification du temps du simulateur comme étant celui de l'évènement
  - Exécution de l'évènement
  - Exécution des potentiels évènements de métriques non distribués
- Fin

Les évènements seront ajoutés au fur et à mesure que d'autres sont exécutés (par exemple l'évènement décharge qui ajoute une autre décharge en queue). La fin du simulateur se situe, d'après la manière d'opérer, lorsque la file du simulateur est vide. C'est pour cela que l'évènement fin vide cette file afin de permettre au simulateur de s'arrêter.

Il est à noter qu'un évènement ayant un temps inférieur au temps actuel du simulateur ne peut être ajouté dans la file. En effet, cela pourrait mener a des états incohérents.

1.4 Premiers lancements

En ayant implémenté les parties précédentes notre simulateur peut déjà être lancé et effectuer les opérations qui lui sont demandées. Dans notre cas cela revient à avoir des capteurs qui vont se décharger de manière constante tout au long de la simulation.

```
14:15:14.494 INFO  [fr.mrcraftcod.simulator.Main] - Main.main:47 - Starting simulator version 7.0.0
14:15:14.654 INFO  [fr.mrcraftcod.simulator.Main] - Main.main:82 - Replication 1/1
14:15:14.709 INFO  [fr.mrcraftcod.simulator.Environment] - Environment.setSeed:116 - The seed to be used for random generation is: 1552482914708
14:15:14.771 TRACE [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.setCurrentCapacity:194 - Set sensor fr.mrcraftcod.simulator.sensors.Sensor[1] current capacity from 0.0 to 0.0
14:15:14.778 DEBUG [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.<init>:66 - New sensor created: fr.mrcraftcod.simulator.sensors.Sensor[1]
14:15:14.788 TRACE [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.setCurrentCapacity:194 - Set sensor fr.mrcraftcod.simulator.sensors.Sensor[1] current capacity from 0.0 to 33.473502142456454
14:15:14.812 TRACE [fr.mrcraftcod.simulator.Main] - Main.loadParameters:164 - Params: fr.mrcraftcod.simulator.SimulationParameters@3d1848cc[environment=fr.mrcraftcod.simulator.Environment@7dda48d9[elements_count=1,s
14:15:14.814 INFO  [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:121 - Starting simulator
14:15:14.821 DEBUG [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:132 - Executing event StartEvent at time 0.0
14:15:14.828 DEBUG [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:132 - Executing event DischargeSensorEvent at time 1.0
14:15:14.833 TRACE [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.setCurrentCapacity:194 - Set sensor fr.mrcraftcod.simulator.sensors.Sensor[1] current capacity from 33.473502142456454 to 33.37350214245645
14:15:14.845 DEBUG [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:132 - Executing event DischargeSensorEvent at time 2.0
14:15:14.846 TRACE [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.setCurrentCapacity:194 - Set sensor fr.mrcraftcod.simulator.sensors.Sensor[1] current capacity from 33.37350214245645 to 33.27350214245645
14:15:14.847 DEBUG [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:132 - Executing event DischargeSensorEvent at time 3.0
14:15:14.848 TRACE [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.setCurrentCapacity:194 - Set sensor fr.mrcraftcod.simulator.sensors.Sensor[1] current capacity from 33.27350214245645 to 33.17350214245645
14:15:14.849 DEBUG [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:132 - Executing event DischargeSensorEvent at time 4.0
14:15:14.850 TRACE [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.setCurrentCapacity:194 - Set sensor fr.mrcraftcod.simulator.sensors.Sensor[1] current capacity from 33.17350214245645 to 33.07350214245645
14:15:14.850 DEBUG [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:132 - Executing event DischargeSensorEvent at time 5.0
14:15:14.851 TRACE [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.setCurrentCapacity:194 - Set sensor fr.mrcraftcod.simulator.sensors.Sensor[1] current capacity from 33.07350214245645 to 32.97350214245645
14:15:14.852 DEBUG [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:132 - Executing event DischargeSensorEvent at time 6.0
14:15:14.853 TRACE [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.setCurrentCapacity:194 - Set sensor fr.mrcraftcod.simulator.sensors.Sensor[1] current capacity from 32.97350214245645 to 32.873502142456445
14:15:14.856 DEBUG [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:132 - Executing event DischargeSensorEvent at time 7.0
14:15:14.857 TRACE [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.setCurrentCapacity:194 - Set sensor fr.mrcraftcod.simulator.sensors.Sensor[1] current capacity from 32.873502142456445 to 32.773502142456444
14:15:14.858 DEBUG [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:132 - Executing event DischargeSensorEvent at time 8.0
14:15:14.873 TRACE [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.setCurrentCapacity:194 - Set sensor fr.mrcraftcod.simulator.sensors.Sensor[1] current capacity from 32.773502142456444 to 32.67350214245644
14:15:14.874 DEBUG [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:132 - Executing event DischargeSensorEvent at time 9.0
14:15:14.876 TRACE [fr.mrcraftcod.simulator.sensors.Sensor] - Sensor.setCurrentCapacity:194 - Set sensor fr.mrcraftcod.simulator.sensors.Sensor[1] current capacity from 32.67350214245644 to 32.57350214245644
14:15:14.877 DEBUG [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:132 - Executing event EndEvent at time 10.0
14:15:14.878 INFO  [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.stop:76 - Stopping, clearing queue
14:15:14.879 INFO  [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.run:149 - Simulation ended
14:15:14.880 INFO  [fr.mrcraftcod.simulator.simulation.Simulator] - Simulator.stop:76 - Stopping, clearing queue
14:15:14.882 INFO  [fr.mrcraftcod.simulator.Main] - Main.main:89 - Replication 1/1 done
```

Figure 1 – *Première simulation*

On peut ici observer un chargeur se faisant décharger de 0.1 (définit dans la configuration) à chaque unité de temps du simulateur. Cela est effectué jusqu'au temps 10 étant définit comme la fin de la simulation.

1.5 Métriques

Maintenant que notre simulateur est capable d'exécuter nos évènements, il est nécessaire que nous soyons en mesure de récupérer des informations sur ce qu'il se passe afin d'en extraire des métriques. Une première solution serait que le simulateur notifie des listeners à chaque évènement traité. Cependant cette approche est très restrictive.

En effet si par exemple un évènement modifie la capacité d'un chargeur et sa position en même temps il sera très couteux de maintenir le code pour que les listeners s'adaptent au code de l'évènement. De plus si par la suite on ajoute un autre évènement qui décharge des capteurs, nos listeners précédemment créés ne gérerons pas ce cas et nous aurons une dissonance entre le simulateur et ce qui est indiqué dans les sorties des métriques.

Afin de contourner ce problème une autre solution a été mise en place. Cette dernière repose sur un système de producteurs/consommateurs. Chaque évènement du simulateur va produire des évènements métriques (MetricEvent) correspondant aux opérations qu'il effectue (décharge de capteur, changement de position, ...). Cet évènement métrique est ensuite envoyé à un dispatcher qui va le redistribuer aux différents consommateurs.

De cette manière le code des listeners de métriques n'est plus dépendant du code d'un évènement. Cela permet ainsi d'écouter des évènement plus ciblés et si un nouvel évènement est créé dans le code par la suite, il pourra lui aussi utiliser ces évènements métriques qui seront écoutées par les listeners approprié.

Dans le cas de notre mini-exemple, deux évènements métriques sont présents :

- Décharge d'un capteur : Permet d'indiquer qu'un chargeur en particulier a eu sa capacité de changé.
- Décharge de capteurs : Permet d'indiquer qu'au moins un des capteurs a eu sa capacité de changé.

Ces évènements sont créés par l'évènement de décharge :

Code source 5.2 – *Evènement de décharge*

```
1  class DischargeSensorEvent extends SimulationEvent{
2      DischargeSensorEvent(final double time){ super(time); }
3
4      @Override
5      public void accept(final Environment environment){
6          environment.getElements(Sensor.class).forEach(s -> {
7              s.removeCapacity(s.getDischargeSpeed());
8              environment.getSimulator().getMetricEventDispatcher().dispatchEvent(new
                       SensorCapacityMetricEvent(environment, getTime(), s,
                       s::getCurrentCapacity));
9          });
10         environment.getSimulator().getMetricEventDispatcher().dispatchEvent(new
                   SensorsCapacityMetricEvent(environment, getTime()));
11         environment.getSimulator().getUnreadableQueue().add(new
                   DischargeSensorEvent(getTime() + 1));
12     }
13 }
```

On peut observer à la ligne 8 la création et envoi de l'évènement métrique décharge d'un capteur. De manière similaire la ligne 10 effectue le même traitement lorsque au moins un des capteur a changé de capacité.

Du coté des listeners, ces derniers les reçoivent tous. A eux de trier ceux qu'ils veulent garder et les traiter.

Voici un exemple d'un listener qui sauvegarde sous format CSV la capacité des capteurs tout au long de la simulation :

Code source 5.3 – *Exemple de MetricEventListener*

```
1  public class SensorCapacityMetricEventListener implements MetricEventListener{
2      private final PrintWriter outputFile;
3
4      public SensorCapacityMetricEventListener(final Environment environment) throws
               FileNotFoundException{
5          final var path =
                   MetricEvent.getMetricSaveFolder(environment).resolve("sensor")
                   .resolve("capacity.csv");
6          outputFile = new PrintWriter(new FileOutputStream(path.toFile()));
7          outputFile.print("time");
8          outputFile.print(CSV_SEPARATOR);
9          outputFile.println(environment.getElements(Sensor.class).stream()
                   .map(Identifiable::getUniqueIdentifier).sorted()
                   .collect(Collectors.joining(CSV_SEPARATOR)));
10         outputFile.flush();
11     }
12
13     @Override
14     public void onEvent(final MetricEvent event){
15         if(event instanceof SensorsCapacityMetricEvent){
16             final var evt = (SensorsCapacityMetricEvent) event;
17             outputFile.print(evt.getTime());
18             outputFile.print(CSV_SEPARATOR);
19             outputFile.println(event.getEnvironment().getElements(Sensor.class)
                       .stream().sorted(Comparator
                       .comparing(Identifiable::getUniqueIdentifier)).map(s -> "" +
                       s.getCurrentCapacity())
                       .collect(Collectors.joining(CSV_SEPARATOR)));
20             outputFile.flush();
21         }
```

```
22      }
23
24      @Override
25      public void close(){
26          outputFile.close();
27      }
28  }
```

Sortie :

Code source 5.4 – *Exemple de sortie de métrique*

```
1  time,fr.mrcraftcod.simulator.sensors.Sensor[1]
2  1.0,30.696919155275385
3  2.0,30.596919155275383
4  3.0,30.496919155275382
5  4.0,30.39691915527538
6  5.0,30.29691915527538
7  6.0,30.196919155275378
8  7.0,30.096919155275376
9  8.0,29.996919155275375
10 9.0,29.896919155275373
```

1.6 Implémentation de l'algorithme de Mme Rault

Maintenant que nous avons un simulateur fonctionnel, il ne reste plus qu'à l'étendre comme si ce dernier était une librairie. On va en fait réaliser ce que tout le monde peut faire avec le simulateur du fait de sa grande adaptation.

Nous allons donc devoir implémenter :

- Nos capteurs L_r / L_c
- Notre système de routage :
  - Logique du routage
  - TSP
  - TSPMTW
- Les évènements associés (requête L_c, requête L_r, début d'une tournée, ...)
- Les évènements métrique associés

Les parties suivantes vont détailler les deux premiers points et inclurons les 2 derniers au fur et a mesure.

1.6.1 Capteur *LrLc*

Notre capteur LrLcSensor va repartir de l'implémentation basique Sensor et y ajouter les éléments propre à ce type de capteur :

- Valeur de L_c.
- Valeur de L_r.

Rien de plus concernant le capteur en lui même. Cependant il est nécessaire de créer nos évènements de requêtage au près du serveur de routage. Pour cela on attache un SensorListener qui va effectuer des actions lorsque le niveau de batterie est modifié :

- Si une requête L_r n'a pas déjà été faite :
  - Si on est en dessous du seuil L_r :
    - Création d'un évènement LrRequestEvent.
    - Enregistrement de la demande.
- Si une requête L_c n'a pas déjà été faite :
  - Si on est en dessous du seuil L_c :
    - Création d'un évènement LcRequestEvent.
    - Enregistrement de la demande.
- Si une requête L_r a déjà été faite :
  - Si on est au dessus du seuil L_r :
    - Réinitialisation des demandes.

L'évènement LrRequestEvent se contente juste d'enregistrer les capteurs ayant fait des requêtes dans une liste et génère aussi un évènement de métrique LrRequestMetricEvent.

L'évènement LcRequestEvent va quant à lui générer un évènement de métrique LcRequestMetricEvent puis lancer le routing avec le Router définit dans l'environnement. Si aucun Router n'est définit, aucun routing ne sera réalisé.

Il est à noter que si une tournée est déjà en cours de réalisation, il est nécessaire d'attendre que tous les chargeurs soient revenus à la base. Dans ce cas l'évènement LcRequestEvent est reporté à la date $t+1$.

1.6.2 Routage

Maintenant que nos capteurs effectuent leur requêtes et qu'un routeur est appelé dès que nécessaire, il nous faut maintenant en définir un qui va respecter l'algorithme vu dans les parties précédentes. Ce dernier est RaultRouter et surcharge la méthode « route » de sa classe parent Router.

Nous allons détailler de manière assez séquentielle ce que le router réalise. Il est à noter que certains points diffèrent de la partie théorique et ces derniers seront mentionnés car ils auront une influence assez forte sur les résultats obtenus.

Le routeur commence tout d'abord par obtenir la liste de tous les chargeurs et vérifie que tous sont disponibles. Si l'un d'entre eux est toujours occupé cela signifie qu'une autre tournée est toujours en cours de traitement et nous devons donc attendre la fin de cette dernière. Dans ce cas le routage n'est pas effectué.

Création des points d'arrêt

Si en revanche ils sont disponibles, nous allons créer toutes les structures nécessaires afin de pouvoir effectuer la tournée. Cela commence par la création des points d'arrêt. En effet l'algorithme ne route pas des capteurs directement mais des zones où le capteur s'arrêtera pour recharger un ou plusieurs capteurs. Un point d'arrêt est représenté par StopLocation et est composé une

coordonnée dans l'environnement ainsi qu'une liste de capteurs présents dans la zone autour de ce point.

Une première différence est ici présente. La théorie repose sur une solution décrite par [6] et résout un problème de clique minimale. Etant donné que cette solution est relativement complexe, cela n'a pas été une priorité pour nous et l'implémentation actuelle est très différente. Nous allons considérer que le rayon de recharge est le rayon le plus petit de tous les chargeurs. Puis pour chaque capteur à recharger, il définit lui même un point d'arrêt. Cela a pour conséquence que le nombre de points d'arrêt n'est pas minimal et certaines zones peuvent être dupliquées, créant aussi des zones de conflict inutiles.

Voici une représentation des différences. Chaque œuil représente un capteur devant être chargé, les cercles les rayon d'action et chaque centre de ces derniers représentent les points d'arrêt.

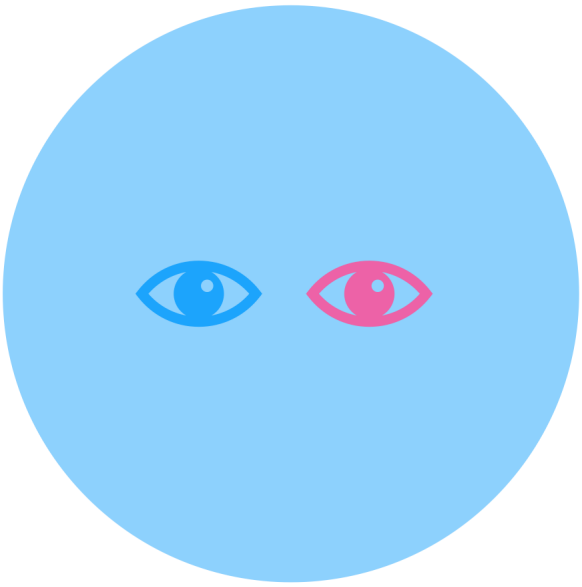


Figure 2 – *Un point d'arrêt qui aurait été généré par l'algorithme théorique*

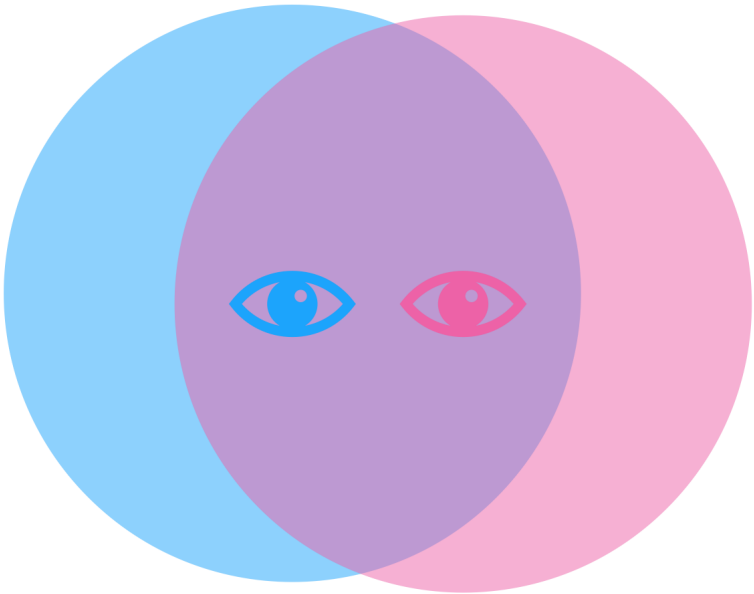


Figure 3 – *Points d'arrêts générés par notre implémentation*

Création des points de recharge

Une fois nos points d'arrêt créés, nous allons les faire évoluer en ChargerStop qui va représenter un point pour le système de routage. Ce dernier contient :

- Un StopLocation comme vu précédemment.
- Un temps de recharge nécessaire.
- Une liste de temps d'indisponibilité.
- Une liste de zones en conflict.
- Une date d'arrivée du chargeur.
- Un chargeur attribué pour ce point.

Lors de la création de ces objets nous faisons encore des hypothèses qui vont influencer sur la qualité de nos résultats. En effet pour calculer le temps de recharge nécessaire pour un ChargingStop, nous prenons le temps maximum de recharge de tous les capteurs dans la zone. Cela a pour effet de ne pas prendre en compte les capteurs étant dans plusieurs zones (et donc chargés deux fois), mais en plus de cela, le temps de recharge d'un capteur est calculé à partir la puissance minimale de recharge des chargeurs. Or il est possible que le chargeur qui sera attribué à cette zone ait une puissance plus élevée.

A ce moment, notre objet contient seulement la StopLocation ainsi qu'une estimation du temps de recharge nécessaire pour cette zone.

Création des routes

Maintenant que nous avons des objets plus précis sur ce qu'est un point d'arrêt, il faut leur assigner un chargeur qui devra s'occuper de cette zone. Pour cela nous reprenons exactement la partie théorique. Chaque capteur a une zone associé aléatoirement, puis pour les points restant, nous ajoutons au ChargingStop ayant le moins de temps accumulé la zone la plus proche.

Tout cela est consigné dans un ChargerTour contenant un chargeur et une liste de ChargerStop représentant ainsi la route d'un chargeur pour une tournée.

Enfin, les routes sans points d'arrêt (cas où il y a plus de chargeurs que de points d'arrêt à desservir) sont détruits.

Création des zones de conflict

Pour chaque ChargingStop nous mettons ensuite à jour les zones avec lesquelles il est en conflict. A ce moment nous savons quel chargeur va s'occuper de chaque zone, nous utilisons donc le vrai rayon et nous évite ainsi de faire des hypothèses.

Ordonnancement des tours

A ce point nous avons chaque chargeur avec la liste de points qu'il devra recharger. Il nous faut à présent les arranger dans l'ordre dans lequel nous voulons qu'ils soient visités.

TSP

Dans le cas du premier tour que nous traitons, nous effectuons un simple TSP. En effet il n'y a aucune contrainte de fenêtre de temps.

Pour cela nous utilisons la librairie OR-Tools. Nous allons expliquer de manière synthétique comment cette dernière est utilisée.

Code source 5.5 – *TSP*

```
1 public Optional<Pair<List<Integer>, List<Double>>> solve(){
2   //TODO: See https://github.com/google/or-tools/issues/885 should be fixed in ↩
          ortools 7.0
3   @SuppressWarnings("MismatchedQueryAndUpdateOfCollection") final ↩
          List<NodeEvaluator2> callbacks = new ArrayList<>();
```

```
4
5   final var routing = new RoutingModel(getTour().getStops().size() + 1, 1, 0);
6
7   //Setup distances
8   final var distanceCallback = new ↩
            DistanceCallback(getTour().getCharger().getPosition(), getTour().getStops());
9   routing.setArcCostEvaluatorOfAllVehicles(distanceCallback);
10
11  //Setup charger
12  final var totalTimeCallback = new TotalTimeCallback(getTour().getCharger(), ↩
            getTour().getStops());
13  routing.addDimension(totalTimeCallback, MAX_DIMENSION_RANGE, MAX_DIMENSION_RANGE, ↩
            true, "time");
14
15  final var search_parameters = RoutingSearchParameters.newBuilder() ↩
            .mergeFrom(RoutingModel.defaultSearchParameters()) ↩
            .setFirstSolutionStrategy(FirstSolutionStrategy.Value.PATH_CHEAPEST_ARC) ↩
            .setTimeLimitMs(getTimeout() * 1000L).build();
16
17  callbacks.add(distanceCallback);
18  callbacks.add(totalTimeCallback);
19
20  LOGGER.info("Starting TSP");
21  final var solution = routing.solveWithParameters(search_parameters);
22  final var arrivalTimes = new ArrayList<Double>();
23  if(solution != null){
24    LOGGER.debug("TSP cost for tour of {}: {}", ↩
              getTour().getCharger().getUniqueIdentifier(), solution.objectiveValue());
25    final var newOrder = new ArrayList<Integer>();
26    for(var node = routing.start(0); !routing.isEnd(node); node = ↩
              solution.value(routing.nextVar(node))){
27      final var time = routing.cumulVar(node, "time");
28      if(node > 0){
29        newOrder.add((int) (node - 1));
30        arrivalTimes.add(getEnvironment().getSimulator().getCurrentTime() + ↩
                  (solution.min(time) / Callbacks.COST_MULTIPLICAND) - ↩
                  getTour().getStops().get((int) node - 1).getChargingTime());
31      }
32    }
33    return Optional.of(MutablePair.of(newOrder, arrivalTimes));
34  }
35  else{
36    LOGGER.warn("TSP found no solution for {}, keeping old order", getTour());
37  }
38  return Optional.empty();
39 }
```

La ligne 5 permet de définir un model de routing. Les paramètres sont les suivants :

- Le nombre de points : dans notre cas le nombre de points d'arrêt plus un point pour la base.
- Le nombre de véhicule : dans notre cas 1 car nous routons chaque chargeur un par un.
- L'index du point de dépôt : dans notre cas la base.

Les lignes 8 et 9 permettent de définir les distances entre les différents points et l'assigner à notre model. La classe DistanceCallBack contient notamment une méthode qui à partir de deux indices de points va renvoyer une distance. Il est à noter que dans la librairie cette distance est un entier, or notre programme utilise des nombres réels. Cela a pour conséquence de perdre des informations. Par exemple si nous prenons un carré de coté 1 entre 4 points, toutes les distances

seront de 1 (même les diagonales). Pour effacer un peu ce problème, la distance sera multipliée par une constante « Callbacks.COST_MULTIPLICAND ». Actuellement sa valeur est 10 et nous permet donc de garder une précision d'un chiffre après la virgule.

Les lignes 12 et 13 ajoutent une nouvelle dimension « time » au problème en lui affectant TotalTimeCallback comme source pour les valeurs. De manière similaire, à partir de deux indices, une valeur sera retournée. Sauf que cette fois-ci cette dernière représentera le temps nécessaire pour se rendre à un point et recharger ce dernier.

La ligne 15 définit les paramètres de notre model. On définit notamment le fait que nous voulons calculer l'arc le moins cher et lui définissons un temps maximal de calcul.

La ligne 21 résout notre modèle.

Les lignes 22 à 28 permettent d'extraire les résultats qu'OR-Tools a trouvé. Ces derniers sont répartis en deux listes :

- Liste des indices des points à visiter dans l'ordre. Par exemple « $\{2,1,0\}$ » inversera l'ordre actuel de la liste dans le cas de 3 zones d'arrêt.
- Liste des temps d'arrivé sur chaque point définit dans la première liste.

Les deux listes sont ensuite utilisées pour réorganiser la liste dans ChargerTour, définir les temps estimés d'arrivé à une zone et mettre à jour les fenêtre de temps interdites dans les zones en conflict.

TSPMTW

Le TSPMTW est utilisé pour les autres ChargeurTour et est très similaire au code du TSP.

Seul le code suivant est inséré à la ligne 14 du code précédent (Code source 5.5).

Code source 5.6 – *TSPMTW*

```
1  //Setup sensors
2  final var timeDimension = routing.getMutableDimension("time");
3  timeDimension.setGlobalSpanCostCoefficient(1);
4  for(var i = 0; i < this.getTour().getStops().size(); i++){
5      final var stopIndex = routing.nodeToIndex(i);
6      final var timeWindowCumulVar = timeDimension.cumulVar(stopIndex);
7      timeWindowCumulVar.setRange(0, MAX_DIMENSION_RANGE);
8      for(final var occupation : getTour().getStops().get(i).getForbiddenTimes()){
9          timeWindowCumulVar.removeInterval((long) occupation.getLeft().doubleValue(),
                   (long) occupation.getRight().doubleValue());
10         LOGGER.info("Node {} forbidden from {}, to {}", stopIndex,
                   occupation.getLeft(), occupation.getRight());
11     }
12 }
```

La ligne 2 permet d'obtenir la contrainte nommé « time ».

La ligne 4 itère pour chaque nœud les opérations suivantes :

- Ligne 5 et 6 : récupère la contrainte du temps pour un noeud donné.
- Ligne 7 : définit la plage dans laquelle le noeud peut être rechargé.
- Ligne 8 et 9 : pour chaque fenêtre de temps d'interdiction, on interdit le rechargement dans notre model.

Limites

La résolution de ce problème est, de par sa nature, non résoluble en temps polynomiale. Nous avons donc des approximations de la solution optimale. De plus du fait des hypothèses précédentes, ce qui est calculé dans le modèle ne reflète pas à 100% ce qu'il se passera dans la simulation.

1.6.3 Evènements

La partie théorique étant résolue et ayant notre tournée de prête, il nous reste plus qu'à la jouer dans la simulation. Pour cela des évènements supplémentaires ont été créés. Nous ne décrirons cependant pas les évènements de métrique (chaque évènement déclenche un évènement de métrique associé).

Le point de départ est TourStartEvent créé par le routeur et permet le début de la tournée. Ce dernier va créer des évènements TourTravelEvent pour chaque chargeur qui va représenter le trajet de la position actuel au point à charger.

TourTravelEvent a deux cas possibles :

- Si plus aucun point n'est à charger, un évènement TourEndEvent est généré à $t + t_b$ où t_b est le temps de trajet pour retourner à la base.
- Sinon un évènement TourChargeEvent est généré à $t + t_b$ où t_b est le temps de trajet pour aller au point.

TourChargeEvent a deux cas possibles lui aussi :

- Si le point à charger est actuellement en conflict, le chargement sera réessayé à $t + 1$.
- Sinon on recharge le point. Un évènement TourTravelEvent est créé à $t + t_c$ où t_c représente le temps de recharge de la zone. Il est à noter que le temps de charge est recalculé en fonction de l'état actuel des capteurs et du chargeur utilisé. Cela permet de jouer une simulation réaliste en gommant les approximations faites lors du calcul des TSPs ou TSPMTWs. Cependant cela implique un fort décalage entre réalité et théorie. Cela à pour implication que certains points peuvent introduire de fort temps d'attente qui n'étaient pas prévus à l'avance.

1.7 Ajout d'améliorations

le but de ce PRD étant de proposer des améliorations, une nouvelle classe a été crée pour cela : RaultRouterModified. Cette dernière effectue les mêmes opérations que RaultRouter mais essaie de prendre en compte une approche où les capteurs sont chargés en plusieurs phases. Cependant du fait que notre modèle OR-Tools est assez simplisme nous effectuons encore des hypothèses. En effet nous ne savons pas si le chargeur fera 3 puis 2 ou bien 2 puis 3. Cela pourrait avoir une influence sur la répartition des temps de recharge, et donc sur les fenêtres de temps interdites. A cause de cela nous imposons à l'avance le temps de recharge de chaque zone de manière aléatoire. **Il se peut donc que lors de la simulation l'ordre soit inversé et donc des temps d'attente non prévus soient ajoutés**.

Le reste des améliorations proposés peuvent s'effectuer principalement dans le fichier de configuration JSON en modifiant les paramètres de L_r ou L_c.

1.8 Interface graphique

A ce point la simulation est jouée dans la console. Cela est suffisant afin obtenir des résultats mais assez peu pratique pour se rendre compte de ce qu'il se passe dans la simulation.

Pour cela nous proposons une interface graphique qui va afficher différentes représentations de l'état actuel de la simulation. Cette dernière se présente principalement comme listener des évènements de métrique et agit donc vraiment comme une vue sur l'état et non pas comme un moyen d'interagir avec cette dernière. De ce fait nous avons un modèle MVC est simplifié où le contrôleur est très réduit. De plus, l'utilisation de JavaFX permet d'automatiser beaucoup d'actualisations des éléments graphiques.

1.8.1 Contrôles de la simulation

Comme dit précédemment le but de la vue n'est pas d'avoir un contrôle sur la simulation. Cependant certains moyens d'interagir sont quand même disponible.

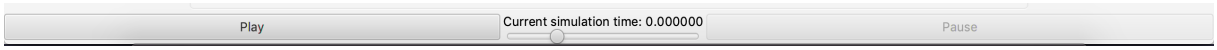


Figure 4 – *Boutons de contrôle de la simulation*

Nous y retrouvons deux boutons afin de mettre en pause ou reprendre la simulation. Ces derniers vont modifier une variable dans le Simulator.

Nous pouvons aussi observer le temps actuel de la simulation. Cette dernière est mise à jour automatiquement par JavaFX. En effet ce dernier est stocké dans une Property, élément de JavaFX qui implémente un pattern Observer. Cela a donc pour conséquence de mettre à jour chaque élément graphique reposant sur cette variable.

Enfin un slider est présent et permet d'ajuster la vitesse de la simulation. Ce ressenti est réalisé en ajoutant un temps de pause entre chaque itération du simulateur. Le slider influe sur ce temps de pause. Encore une fois cela est fait avec une Property, dès que le slider change, la variable associée est modifiée automatiquement.

1.8.2 Etat des batteries

La première visualisation qui a été implémenté consiste à afficher les différentes capacité des chargeurs au cours du temps. Le moyen le plus simple de réaliser cela est sous forme d'un graphique avec en abscisse le temps et en ordonnée la capacité.

Cette représentation est très simpliste mais ne permet pas de tout voir. En effet on peut repérer les moments de recharge mais rien de plus. De plus avec un grand nombre de capteurs le graphique devient assez vite illisible.

Contrairement à la vue précédente, ici nous ne faisons que lire les données du simulateur. Pour cela le graphique implémente simplement MetricEventListener et construit sa fenêtre basé sur les évènements de métrique.

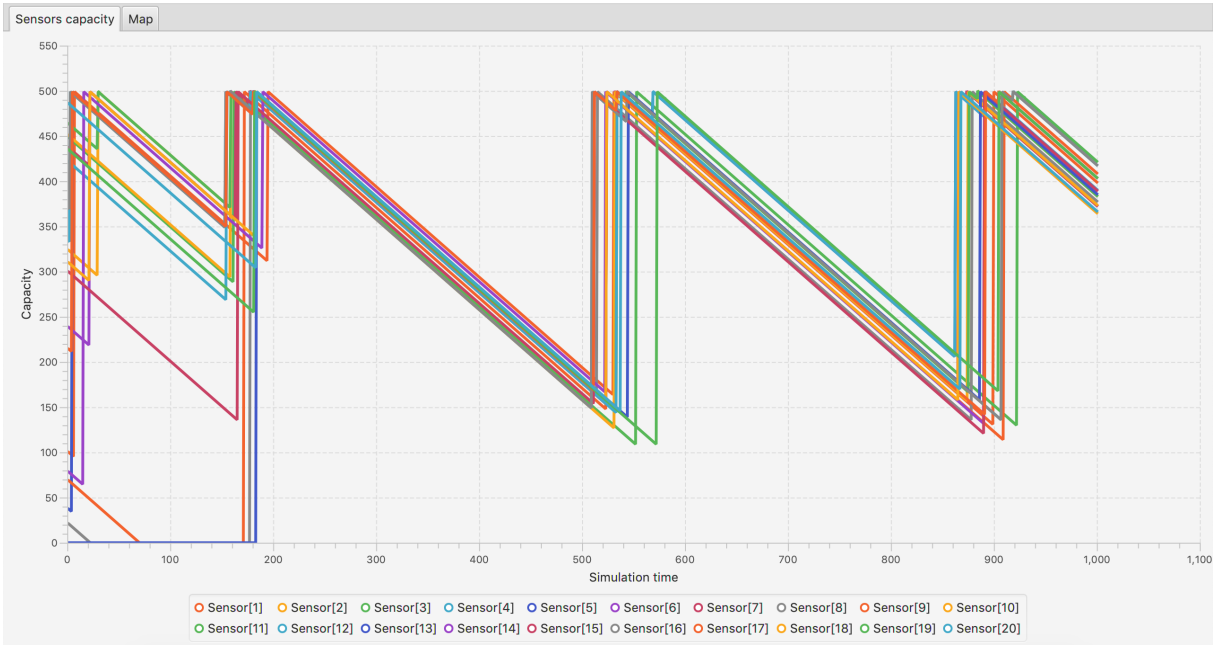


Figure 5 – *Visualisation de la capacité des capteurs*

1.8.3 Carte des éléments

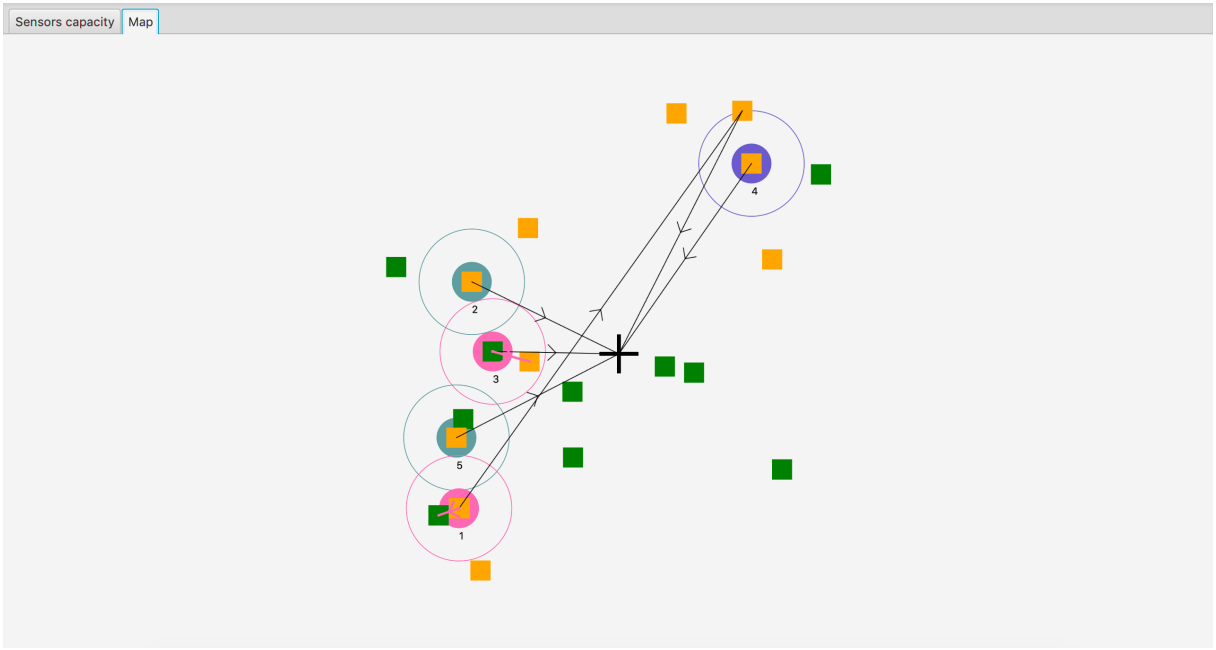


Figure 6 – *Visualisation de la simulation sous forme spatiale*

Une autre approche implémentée est de visualiser la simulation sous forme spatiale en plaçant les différents éléments sur une « carte » et les faire évoluer au cours du temps. Cette approche permet de visualiser plus simplement ce qu'il se passe dans la simulation mais a le désavantage de ne pas conserver cette évolution (on ne peut pas revenir en arrière dans le temps).

Ici nous représentons uniquement les capteurs et chargeurs ainsi que quelques informations visuelles. Les carrés représentent les capteurs. Ces derniers peuvent voir plusieurs états :

- Vert : Le capteur est dans un état normal (capacité au dessus de L_r).
- Orange : Le capteur a sa capacité en dessous de L_r et a effectué une requête.

- Rouge : Le capteur est en dessous de L_c et a effectué une requête.

On a plus de détail précis sur la capacité de ces derniers mais avons tout de même une indication sur les étapes majeures de l'évolution de la batterie.

Vient ensuite les chargeurs. Ils sont représentés par des disques avec un cercle autour d'eux représentant leur rayon d'action. De la même manière plusieurs états sont possibles :

- Cyan : Le chargeur est au repos ou en attente à cause d'une zone en conflict.
- Violet : Le chargeur est en cours de déplacement vers l'emplacement où il se trouve.
- Rose : Il est en train de recharger des capteurs dans son rayon d'action. A ce moment des petites flèches supplémentaires rose vont apparaître montrant plus précisément quels capteurs sont rechargés.

En plus de ces éléments, nous retrouvons une croix épaisse au centre de la zone dénotant les coordonnées $(0;0)$ étant la base des chargeurs. On y observe aussi des flèches plus minces permettant de visualiser le trajet que chaque chargeur va réaliser pour effectuer une tournée.

Cette vue est particulièrement efficace pour repérer les interactions entre les chargeurs et entre les chargeurs et capteurs. On y identifie rapidement les moments où des chargeurs sont en conflicts.

Afin de faciliter la navigation dans cette vue, il est possible de déplacer la zone en restant cliqué dans celle-ci et glissant la souris. Nous pouvons aussi zoomer grâce à la molette.

A titre indicatif pour montrer que les évènements de métriques permettent un identification poussée de la scène, la vue repose totalement sur les évènements suivants : LrRequestMetricEvent, LcRequestMetricEvent, SensorChargedMetricEvent, TourStartMetricEvent, TourTravelMetricEvent, TourTravelBaseMetricEvent, TourTravelEndMetricEvent, TourEndMetricEvent, TourChargeMetricEvent et TourChargeEndMetricEvent.

2 Analyse des résultats

Afin de savoir si l'ajout de nos modifications dans l'algorithme a une pertinence, nous allons tester les deux versions grâce au simulateur. L'instance utilisée va essayer de se rapprocher le plus possible de ceux utilisés dans le papier de Mme Rault :

Code source 5.7 – *"Lr*

```
{
  "end": 5000,
  "seed": 983416832,
  "environment": [
    {
      "class": "fr.mrcraftcod.simulator.rault.sensors.LrLcSensor",
      "count": 50,
      "parameters": {
        "powerActivation": 0.1,
        "position": {
          "class": "fr.mrcraftcod.simulator.positions.RandomPosition",
          "parameters": {
            "minX": -12.5,
            "maxX": 12.5,
            "minY": -12.5,
            "maxY": 12.5
          }
        },
        "dischargeSpeed": 0.1,
        "maxCapacity": 50,
```

```
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.RandomCapacity",
          "parameters": {
            "max": 50
          }
        },
        "lc": 3,
        "lr": 18
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.chargers.Charger",
      "count": 5,
      "parameters": {
        "transmissionPower": 5,
        "radius": 2.7,
        "maxCapacity": 1000000,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 1000000
          }
        },
        "speed": 2
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.rault.routing.RaultRouter",
      "count": 1,
      "parameters": {
      }
    }
  ],
  "metrics": [
    "fr.mrcraftcod.simulator.metrics.listeners.ReplicationTotalDepletionMetricEventListener",
    "fr.mrcraftcod.simulator.metrics.listeners.ReplicationTotalChargerInactiveTimeMetricEventListener",
    "fr.mrcraftcod.simulator.metrics.listeners.ReplicationChargerCapacityUsedMetricEventListener"
  ]
}
```

Cette configuration a été utilisée pour réaliser 100 réplications. Vous trouverez ci-dessous des histogrammes montrant la répartition de la différence entre les deux méthodes sur trois critères :

- Le temps de panne des capteurs.
- Le temps d'inactivité des chargeurs.
- La capacité utilisé par les chargeurs.

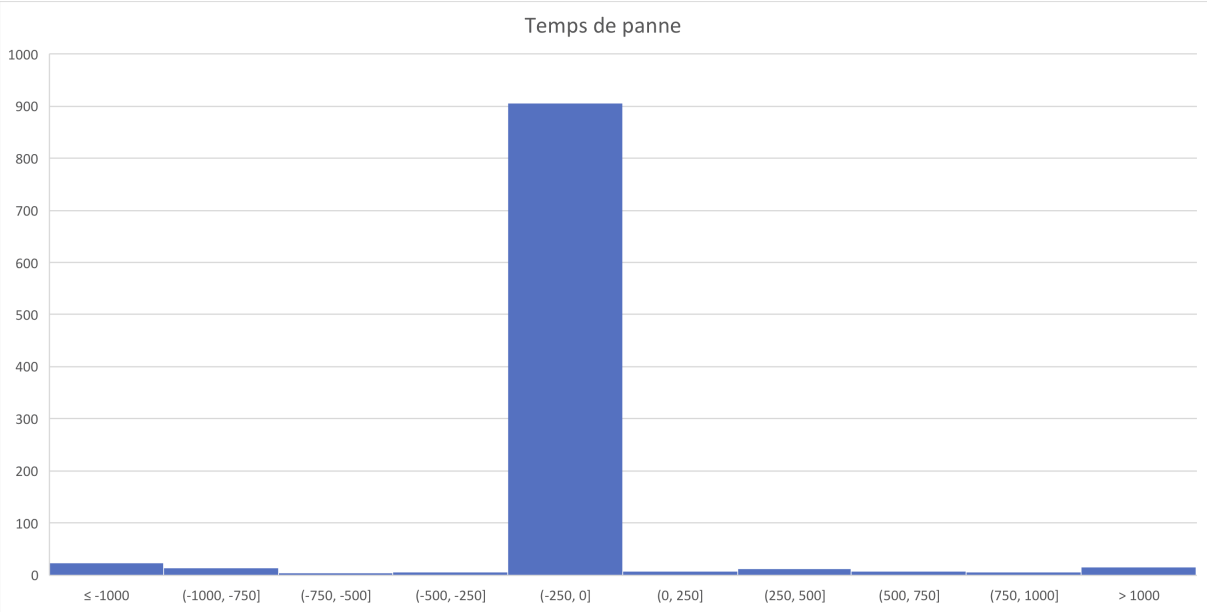


Figure 7 – *Différence du temps de panne entre les deux méthodes (les scores positifs sont meilleurs)*

Nous pouvons observer que dans la majorité des simulations le temps de panne est légèrement détérioré. Cependant cela est aussi contrasté par certaines simulations où ce dernier est soit très diminué, soit très augmenté.

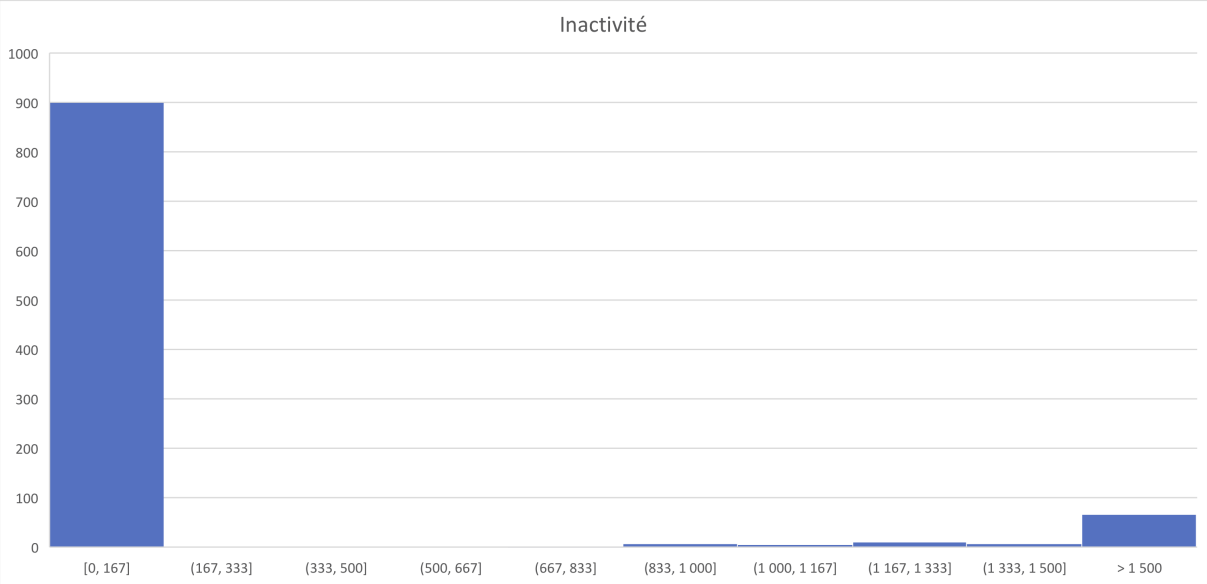


Figure 8 – *Différence du temps d'inactivité des chargeurs entre les deux méthodes (les scores positifs sont meilleurs)*

En revanche le temps d'inactivité des chargeurs voit un gain important. Une amélioration faible dans la majorité des cas, mais qui s'étendent aussi à des valeurs plus grandes dans certains cas. Dans aucune simulation il a été détérioré.

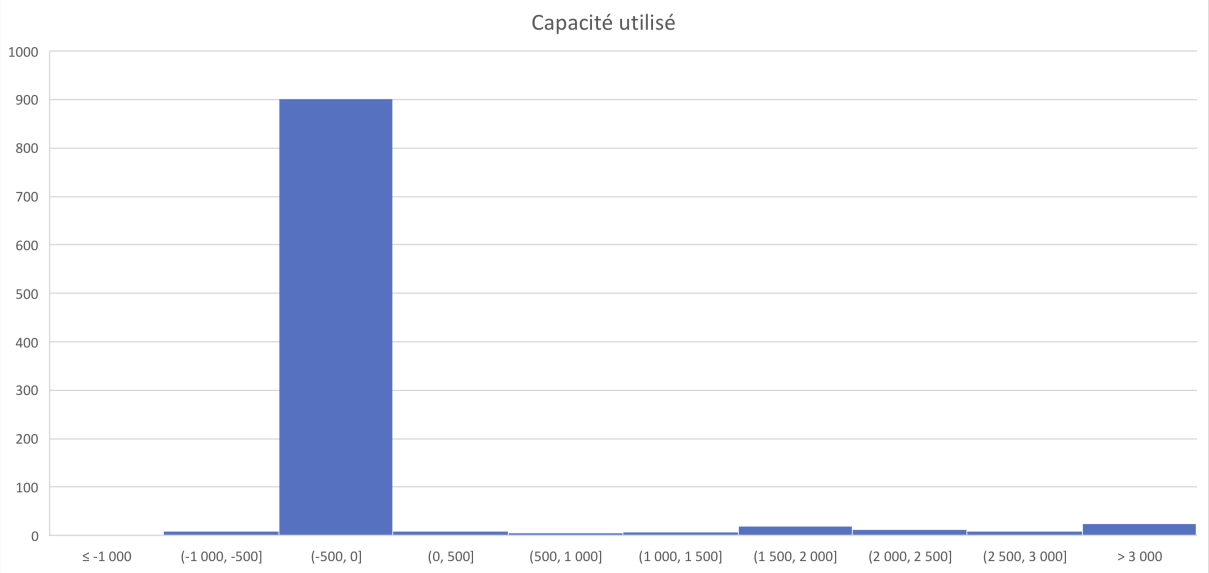


Figure 9 – *Différence du temps d'inactivité des chargeurs entre les deux méthodes (les scores positifs sont meilleurs)*

Concernant La capacité utilisée cela reste assez stable même si on a une majorité des cas où un peu plus est utilisé. Cependant on observe un minorité où cette dernière est grandement moins utilisé.

Il faut aussi noter que la majorité des cas représente une différence très minime puisque les valeurs varient autours de 22000 (un changement de 500 représente 2%).

Voici dans le tableau ci-dessous un résumé des moyennes et écart-types de ces différences. On remarque notamment le fait que les valeurs sont très étalées grâce à l'écart-type très fort.

	Temps de panne	Temps d'inactivité	Capacité utilisé
Moyenne	-17	186.378	196.853
Ecart-type	344.465	597.892	784.709

De manière générale on semble observer que la version modifiée dégrade légèrement les performances (temps de panne & capacité utilisée) mais diminue le temps d'inactivité. Il est nécessaire de garder en tête le fait que le simulateur est loin d'être parfait. En effet toutes les approximations faites durant la phase de résolution du problème entrainent de forts décalages avec ce qu'il se passe dans la simulation. Cela peut notamment entrainer de nombreux conflicts entre les chargeurs et donc décaler tous les temps prévus de recharge ayant ainsi un impact sur le temps de panne des capteurs qui doivent attendre plus longtemps que prévu.

2.1 Autres résultats

Etant donné que les résultats sont très liés aux instances utilisées, nous allons essayer d'étudier l'impact des différents paramètres sur les résultats. Pour cela nous allons faire varier un paramètre à chaque fois et comparer la moyenne des réplications (50) entre la version normale et version modifiée pour chaque métrique.

2.1.1 Nombre de capteurs

Ici nous gardons la zone de capteurs comme étant fixe ($25m^2$) mais changeons le nombre de ces derniers.

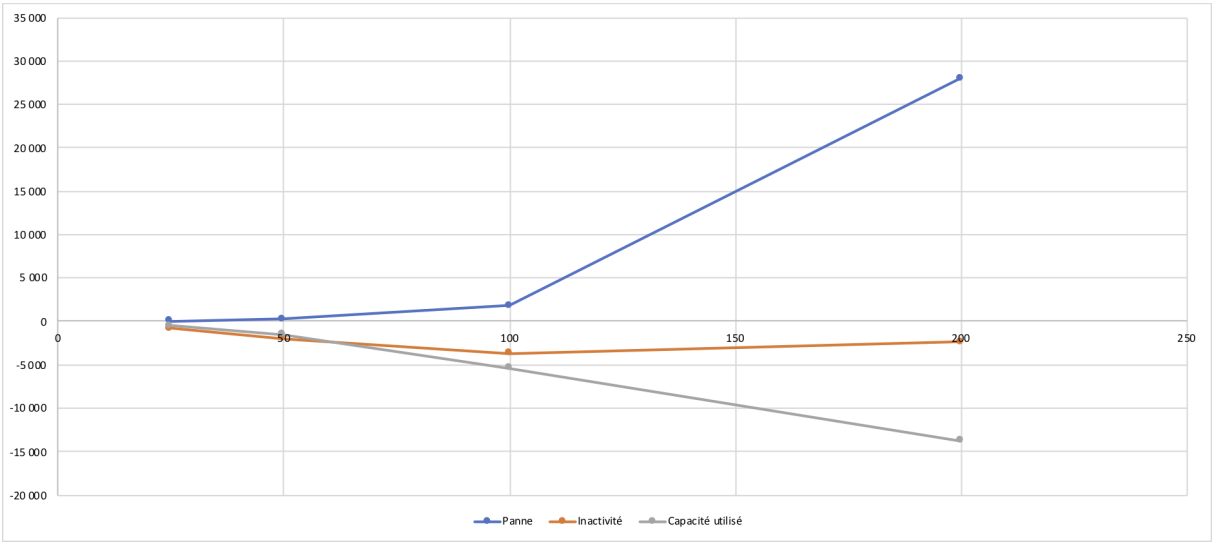


Figure 10 – *Evolution en fonction du nombre de capteurs (bleu : panne, orange : inactivité, gris : capacité utilisé)*

Le même phénomène semble se généraliser. Le temps de panne n'est pas amélioré mais la capacité utilisé et l'inactivité des chargeurs l'est.

2.1.2 Nombre de chargeurs

Comme la simulation précédente, nous gardons les même paramètres, avec un nombre fixe de capteurs de 100 et un nombre variable de chargeurs.

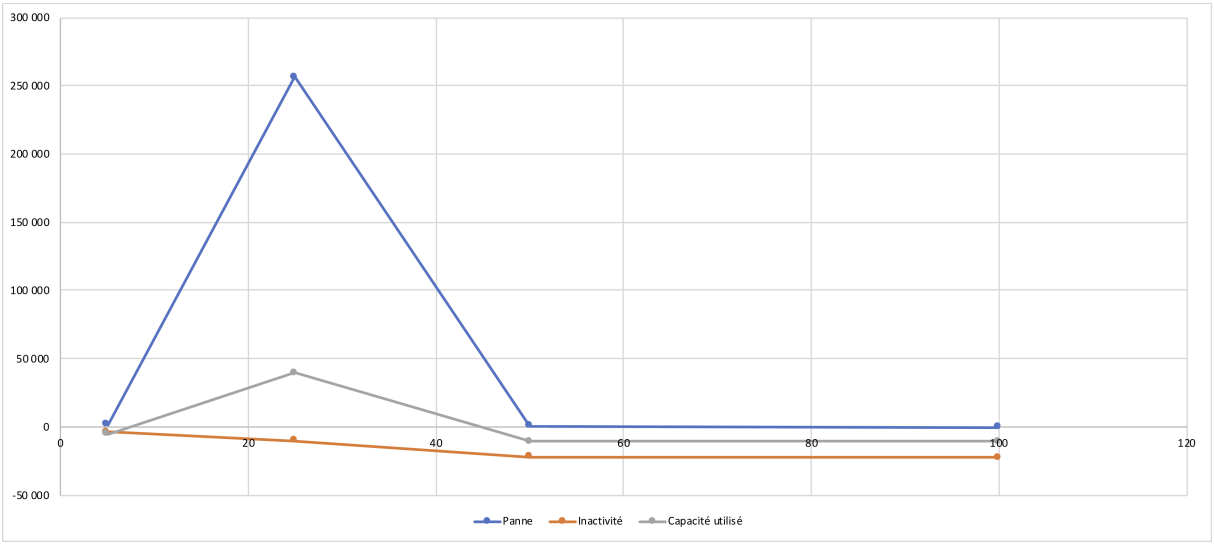


Figure 11 – *Evolution en fonction du nombre de chargeurs (bleu : panne, orange : inactivité, gris : capacité utilisé)*

Encore une fois nous observons la même chose.

3 Conclusion

Après les différents tests effectués, assez peu variés par manque de temps, nous pouvons conclure que la méthode proposée n'améliore pas les performances de l'algorithme en général. Certes certaines métriques semblent être améliorées comme le temps d'inactivité ou la capacité utilisé, cependant la métrique principale, le temps de panne, est détériorée.

Cela est cependant à mettre en parallèle avec la fiabilité du simulateur. En effet comme mentionné précédemment un décalage non négligeable entre les résultats d'OR-Tools et la simulation est présent. De plus la librairie ne donne parfois pas de solutions et donc l'algorithme n'est pas appliqué correctement.

Il serait intéressant de réitérer les simulations une fois que les points cités précédemment soient corrigés. Il est de mon point de vue fort possible d'obtenir des résultats positifs si le décalage est moins important et donc que les interférences entre chargeurs soient réduits.

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 celles 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

Nous allons ici exposer les différentes tâches qui ont été réalisés lors du S10 ainsi que celles qui reste à faire ou les potentielles améliorations envisageable.

2.1 Avancée

- Finalisation du simulateur :
  - Algorithme de Mme Rault.
  - TSP.

  - TSPMTW.
- Extraction de métriques.
- Analyse des résultats

Grâce au simulateur il a été possible d'extraire des résultats afin d'y effectuer des analyses. L'objectif principal du PRD est donc atteint. Cependant la solution proposée n'est pas parfaite et les résultats obtenus peuvent encore beaucoup changer avec l'avancée du simulateur.

2.2 Améliorations envisageables

- Amélioration de l'utilisation d'OR-Tools :
  - Modification du code pour que la librairie renvoie une solution plus souvent.
  - Mise à jour de la librairie pour éviter des crash de la JVM.
- Amélioration de la méthode de clustering des capteurs à recharger.
- Ajout de nouvelles visualisations dans l'UI afin de suivre plus facilement le déroulement de la simulation et repérer de potentiels problèmes.

2.3 Bilan qualité

La qualité de la base du simulateur semble solide. En effet il est très modulable et peut être étendu très facilement pour y implémenter un nouvel algorithme ou éléments. Les différents documents en annexes devraient faciliter la prise en main du code afin que ce dernier puisse être repris de manière efficace.

Concernant l'implémentation de l'algorithme de routage il est à revoir. En effet de nombreux points remettent en cause les résultats finaux car la simulation n'est pas fidèle à la théorie. Cela passe par la phase de clustering qui est très différente et OR-Tools donnant des solutions pas toujours proche de la simulation.

Enfin de manière générale le code comprends des tests unitaires qui sont joués automatiquement en intégration continue. Cela aide grandement à maintenir un code valide. Cependant la couverture de ces derniers n'est que de 30%. Bien que tester le simulateur en lui-même est difficile, il est possible d'étendre cette couverture en testant plus profondément le modèle représentant nos éléments de la simulation. Le simulateur pourrait être testé avec des tests d'intégrations avec Cucumber par exemple.

2.4 Bilan auto-critique

Le PRD a été une bonne occasion de mettre en œuvre les différentes compétences que nous avons apprises tout au long de notre formation. Le fait d'être en solitaire sur le projet nous pousse vraiment à réfléchir par nous-même et nous permet de nous rendre compte de nos acquis dans un environnement mélangeant plusieurs disciplines.

Je pense que mon projet s'est bien passé. Une grande autonomie m'a permis d'avancer malgré un premier semestre où les rencontres étaient difficiles. Cependant il aurait peut-être été meilleur si j'aurai fait preuve d'un peu plus d'initiatives pour aller à la rencontre de l'encadrant.

Les résultats sur l'avancé du projet ont toujours étés positifs. Certains problèmes sont survenus mais des alternatives ont étés trouvées.

De plus ce projet a été l'occasion pour moi de mettre en place de l'intégration continue avec Gitlab ainsi que d'autres outils pour garder un œuil sur la qualité du code (jacoco, pmd).

Annexes

A Découverte du sujet

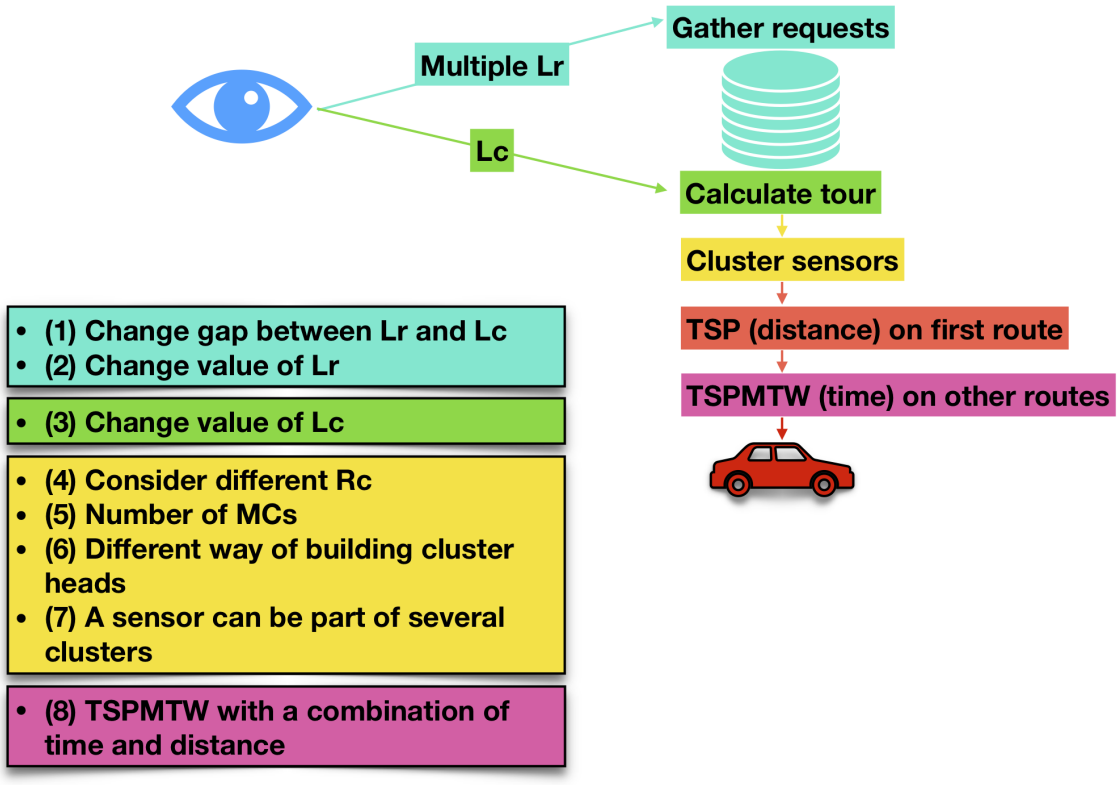


Figure 1 – *Première compréhension du sujet*

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 ».

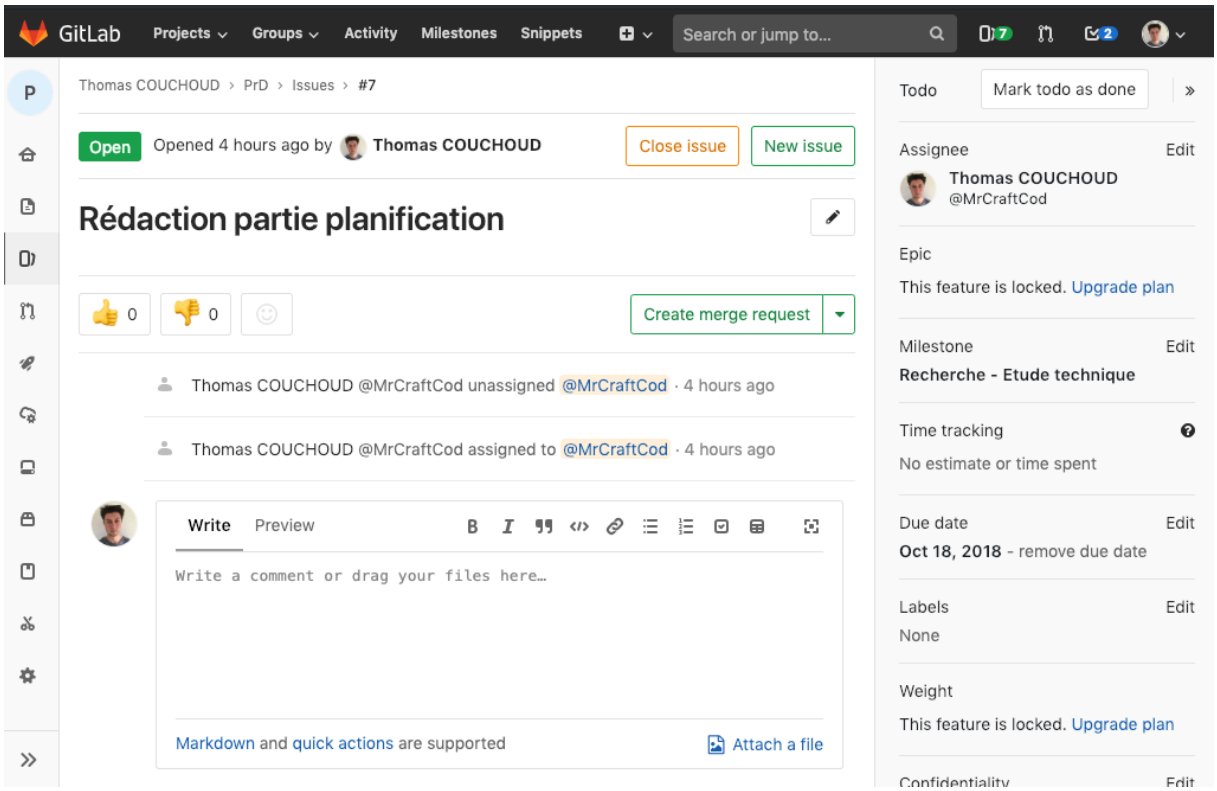


Figure 1 – *Exemple d'une issue*

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 :

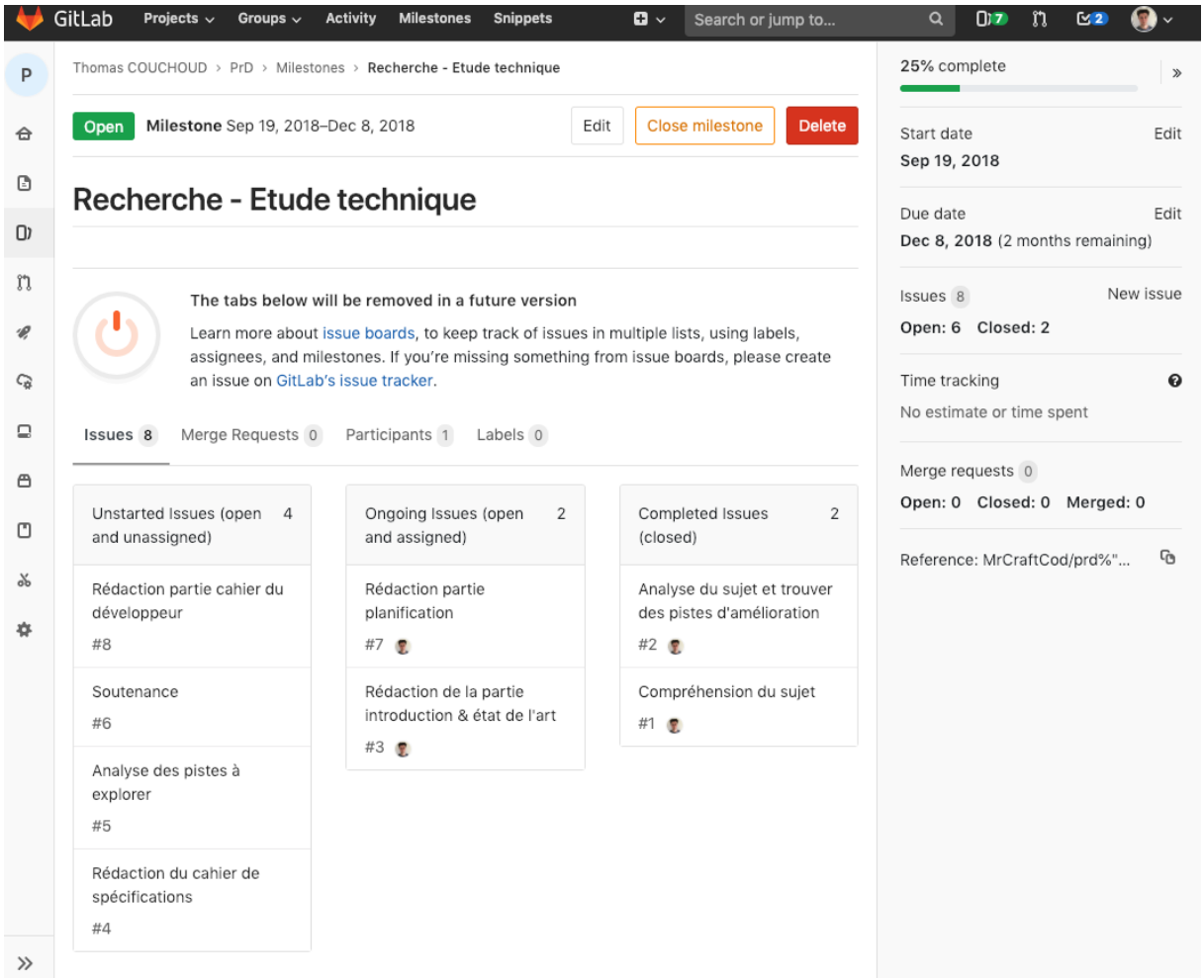


Figure 2 – *Vue milestone*

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** :

  2 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 ressources 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** :

  5 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** :
  2 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 Questionnaire gestion de projet

- **Description de la tâche** :

  Fournir les documents pour l'évaluation de la gestion de projet.

- **Estimation de charge** :

  0.5 jour/homme.

- **Livrable** :

  Questionnaire gestion de projet.

- **Date limite** :

  27/03/2018

2.14 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.

  Poster du projet.

- **Date limite** :

  04/04/2018

2.15 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** :
  27/04/2019

3 Gantt

Si l'on essaie de représenter les tâches du S9 dans un Gantt, on obtient un diagramme théorique suivant :

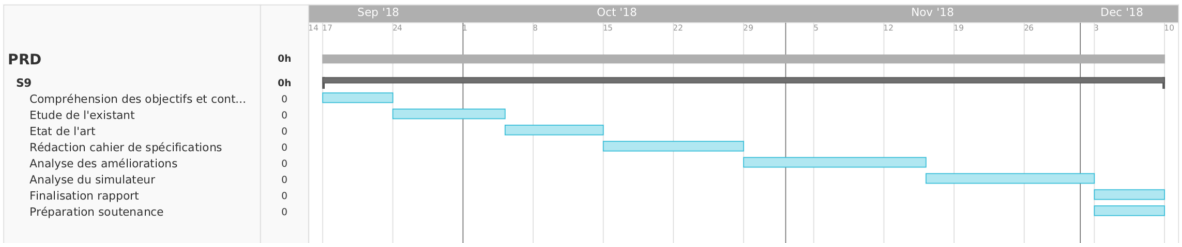


Figure 3 – *Diagramme de Gantt*

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

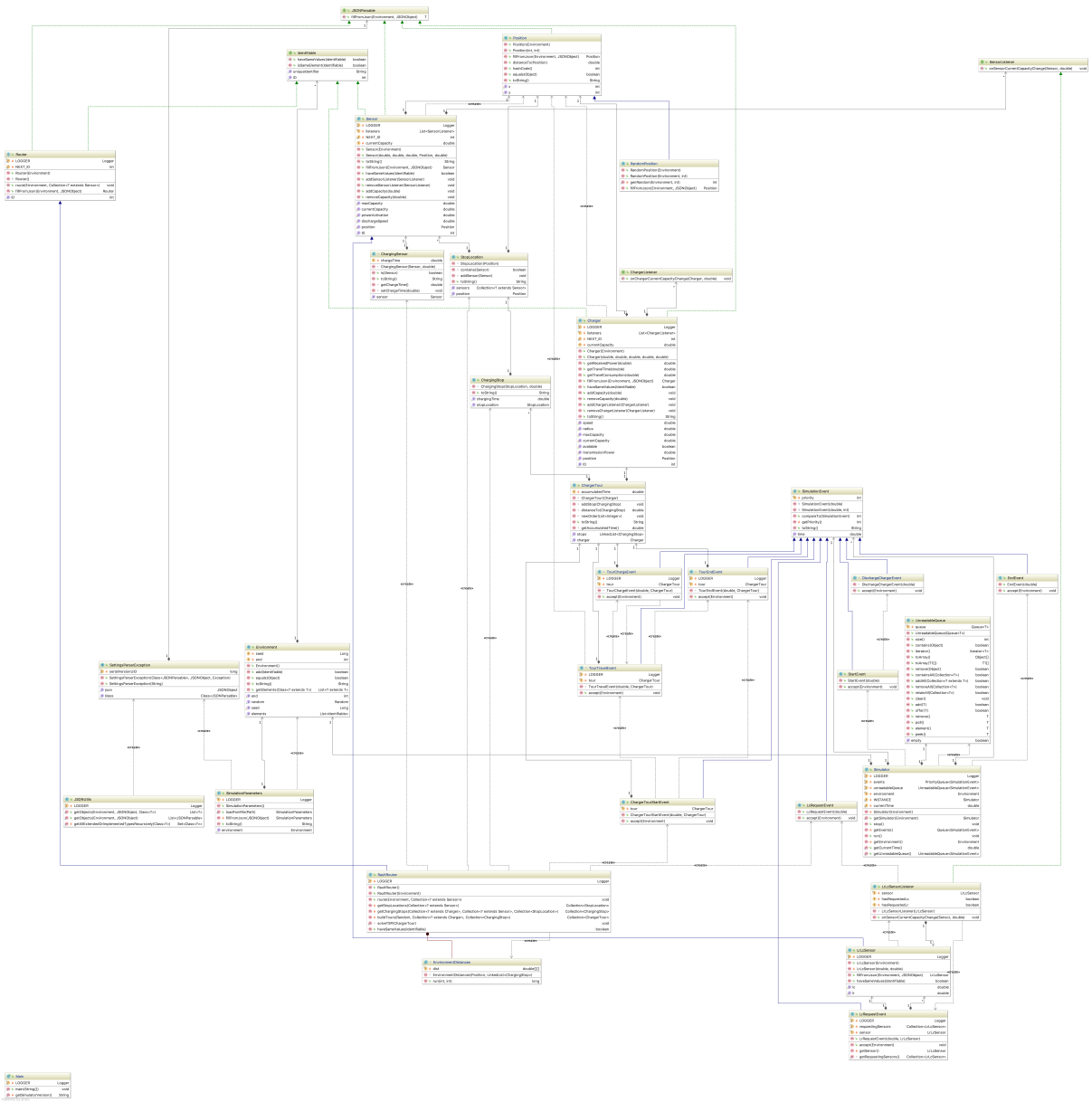


Figure 1 – *Diagramme de classes complet*

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 intervalle. 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

- Langage : Java 12+
- Plateformes cibles : Windows, macOSX, 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

1 Description des classes

Nous allons dans cette section décrire les différentes classes majeures du programme. Commençons tout d'abord par un diagramme représentant les liens entre-elles ainsi que leur attributs/méthodes principales.

1.1 Diagramme de classes

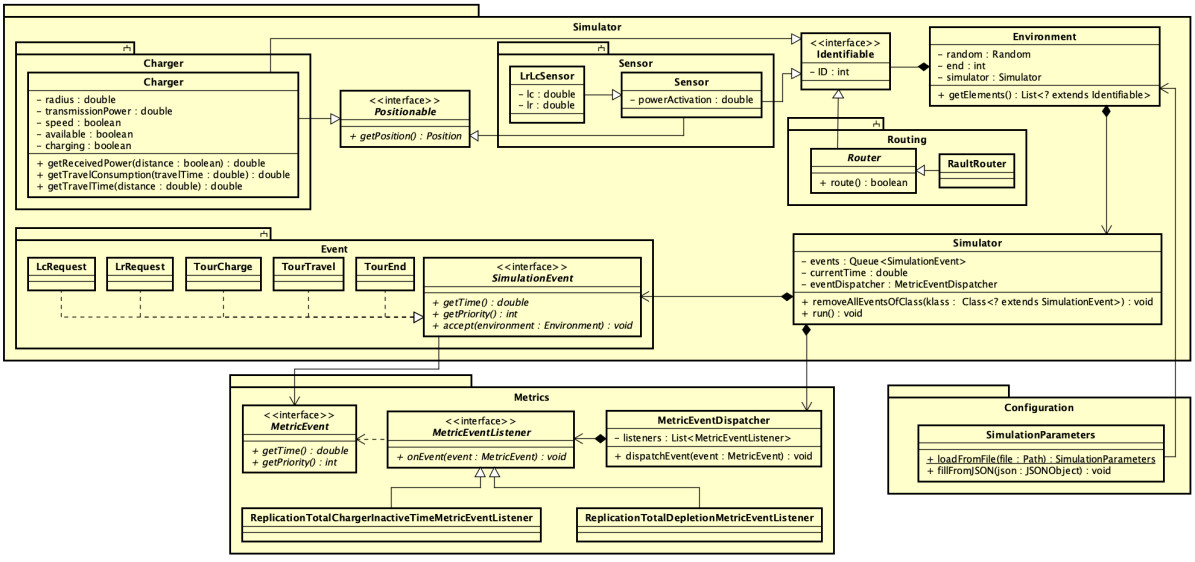


Figure 1 – *Diagramme de classes un peu plus détaille*

1.2 Packages et classes

Toutes les classes sont organisées sous forme de packages ayant chacun un but particulier :

fr.mrcraftcod.simulator

Package principal de l'application contenant tous les autres packages ainsi que :

- **CLIParameters** : Définit les paramètres de la ligne de commande.
- **Environment** : Représente une instance de simulation, contenant les différents « Identifiable », une instance d'un « Simulator » et une instance d'un Random utilisé pour les générations aléatoires de cette instance.
- **Main** : Point d'entré du programme. Charge les paramètres et démarre la simulation ou l'interface graphique.
- **SimulationParameters** : Chargeur de la configuration.

fr.mrcraftcod.simulator.capacity

Package contenant les classes chargées d'obtenir une capacité depuis le fichier de configuration.

- **AbstractCapacity** : Implémente les bases pour stocker une capacité et implémente JSONParsable.
- **Capacity** : Hérite de AbstractCapacity et permet d'obtenir une capacité depuis une valeur.
- **RandomCapacity** : Hérite de AbstractCapacity et génère une capacité dans un intervalle donné.

fr.mrcraftcod.simulator.chargers

Package contenant les classes d'un chargeur.

- **Charger** : Implémente JSONParsable, Identifiable, Positionable et Rechargeable et définit un chargeur dans la simulation.
- **ChargerListener** : Listener permettant pour le moment d'écouter les changements de capacité d'un Charger.

fr.mrcraftcod.simulator.exceptions

Package contenant les exceptions personnalisées.

- **SettingsParserException** : Exception utilisée lors des erreurs durant la lecture de la configuration.

fr.mrcraftcod.simulator.jfx

Package contenant la partie interface graphique.

- **MainApplication** : Fenêtre principale de l'UI.

fr.mrcraftcod.simulator.jfx.tabs

Contient les différents onglets de l'interface graphique.

fr.mrcraftcod.simulator.jfx.utils

Package contenant des utilitaires liés à l'UI.

fr.mrcraftcod.simulator.metrics

Package contenant toute la partie gestion des métriques.

- **IdentifiableMetricEvent** : Etend ValueMetricEvent permettant de représenter un évènement avec une valeur tout en étant associé à une instance d'un Identifiable.
- **MetricEvent** : Représente un évènement de métrique.
- **MetricEventDispatcher** : Classe chargée de distribuer les évènement de métrique aux différents MetricEventListeners.
- **MetricEventListener** : Listener recevant les évènements de métrique.
- **ValueMetricEvent** : Etend MetricEvent associant une valeur à un évènement de métrique.

fr.mrcraftcod.simulator.metrics.events

Package contenant des évènements de métriques.

- **SensorCapacityMetricEvent** : Etend IdentifiableMetricEvent et est déclenché lorsque la capacité d'un capteur est modifiée par le simulateur.
- **SensorsCapacityMetricEvent** : Etend MetricEvent et est déclenchée lorsque au moins une capacité d'un capteur est modifié.

fr.mrcraftcod.simulator.metrics.listeners

Package contenant des implémentations de listeners de métriques.

- **DepletionMetricEventListener** : Etend MetricEventListener permettant d'obtenir un CSV avec pour chaque date le taux de panne cumulé par capteur depuis le début.
- **ReplicationChargerCapacityUsedMetricEventListener** : Etend MetricEventListener permettant d'obtenir un CSV avec pour chaque réplication la capacité totale utilisée par les chargeurs.
- **ReplicationTotalChargerInactiveTimeMetricEventListener** : Etend MetricEventListener permettant d'obtenir un CSV avec pour chaque réplication le temps d'inactivité totale des chargeurs.
- **ReplicationTotalDepletionMetricEventListener** : Etend MetricEventListener permettant d'obtenir un CSV avec pour chaque réplication le temps totale de panne des capteurs.
- **SensorCapacityMetricEventListener** : Etend MetricEventListener permettant d'obtenir un CSV avec pour chaque date la capacité d'un capteur.

fr.mrcraftcod.simulator.positions

Package contenant les classes définissant une position.

- **Position** : Implémente les bases pour stocker une position et calculer des distances. Implémente JSONParsable afin permet d'obtenir une position à partir de valeurs x et y.
- **RandomPosition** : Hérite de Position et génère une position dans des intervalles donnés.

fr.mrcraftcod.simulator.rault

Package contenant les implémentations liées au modèle proposé par Mme Rault.

fr.mrcraftcod.simulator.rault.events

Package contenant les implémentations de SimulationEvents.

fr.mrcraftcod.simulator.rault.metrics.events

Package contenant les implémentations de MetricEvents.

fr.mrcraftcod.simulator.rault.routing

Package contenant le routeur.

- **ChargerTour** : Représente la tournée d'un chargeur.
- **ChargingSensor** : Représente un Sensor qui devra être chargé en lui associant le temps de charge nécessaire.
- **ChargingStop** : Définit un point d'arrêt contenant une StopLocation ainsi qu'une liste de temps d'indisponibilité.
- **RaultRouter** : Etend Router et implémente l'algorithme de Mme Rault.
- **RaultRouterModified** : Etend RaultRouter et définit un modèle où un capteur peut être chargé en plusieurs étapes.
- **StopLocation** : Définit un point d'arrêt contenant une liste de Sensors rechargeable depuis ce point.

fr.mrcraftcod.simulator.rault.sensors

Package contenant les implémentations des capteurs.

- **LrLcSensor** : Etend Sensor et définit un capteur avec un système à valeurs lr et lc.
- **LrLcSensorListener** : Etend SensorListener et génère les évènements lr et lc.

fr.mrcraftcod.simulator.rault.utils

Package contenant des utilitaires pour le routing.

- **TourSolver** : Définit la base d'un moyen d'ordonnancer une tournée de rechargement.
- **TSP** : Etend TourSolver en utilisant une méthode TSP.
- **TSPMTW** : Etend TourSolver en utilisant une méthode TSPMTW.

fr.mrcraftcod.simulator.rault.utils.callbacks

Package contenant des des NodeEvaluator2 pour les modèles d'OR-Tools.

- **Callbacks** : Définit un NodeEvaluator2 avec un poids pour les valeurs.
- **ChargingTimeCallback** : Etend Callbacks et représente le temps de recharge.
- **DistanceCallback** : Etend Callbacks et représente les distances entre les différents points.
- **TotalTimeCallback** : Etend Callbacks et représente la somme de ChargingTimeCallback et TravelTimeCallback.
- **TravelTimeCallback** : Etend Callbacks et représente le temps de voyage entre les différents points.

fr.mrcraftcod.simulator.routing

Package contenant les classes nécessaires à la définition de modèles de routing.

- **Router** : Implémente Identifiable et JSONParsable. Définit les méthodes nécessaires pour que le simulateur puisse effectuer des routings.

fr.mrcraftcod.simulator.sensors

Package contenant les classes d'un capteur.

- **Sensor** : Implémente JSONParsable, Identifiable, Positionable et Rechargeable et définit un capteur dans la simulation.
- **SensorListener** : Listener permettant pour le moment d'écouter les changements de capacité d'un Sensor.

fr.mrcraftcod.simulator.simulation

Package contenant le cœur du simulateur.

- **SimulationEvent** : Représente un évènement de la simulation.
- **Simulator** : Exécute la simulation consistant en une suite de SimulationEvent pouvant interagir avec les éléments de l'Environment.

fr.mrcraftcod.simulator.simulation.events

Package contenant les évènements de base du simulateur.

- **DischargeSensorEvent** : Etend SimulationEvent permettant de décharger les capteurs.
- **EndEvent** : Etend SimulationEvent permettant de marquer la fin de la simulation et arrêter le simulateur.
- **StartEvent** : Etend SimulationEvent permettant de marquer le début de la simulation et initialiser de potentiels paramètres.

fr.mrcraftcod.simulator.utils

Package contenant des utilitaires.

- **Identifiable** : Permet de définir un élément qui est identifiable dans la simulation.
- **JSONParsable** : Définit une classe qui à partir d'un objet JSON va instancier un objet.
- **JSONUtils** : Suite d'utilitaires pour lire un format JSON.
- **Positionable** : Définit une classe ayant une position.
- **Rechargeable** : Définit une classe pouvant être rechargé (ajouter ou retirer de la capacité).
- **UnreadableQueue** : Définit une Queue où seuls les actions d'ajout sont possibles.

2 Description des fichiers I/O

2.1 Fichiers de configuration

Le fichier de configuration est la seule entrée que le programme accepte en entrée. Tout le reste de la configuration est réalisé avec des arguments passés au programme. Le fichier mentionné

précédemment est décrit en Section 1(Annexe H) et a pour but de définir l'instance qui sera utilisée.

Afin de lire ce fichier nous utilisons la classe SimulationParameters nous renvoyant un objet de la même classe contenant un Environment qui nous servira tout au long d'une simulation.

Ce fichier a pour but d'être très extensible. En effet la majorité des paramètres définies référencent une classe Java (JSONParsable) qui sera initialisée. Peu de modifications concernant le parser sont à prévoir, il suffit uniquement de créer les nouvelles classes voulues dans le code.

2.2 Logs

Les logs sont un point essentiel afin de pouvoir retracer ce qu'il s'est produit dans la simulation, surtout en cas de problèmes. Pour cela nous utilisons la librairie slf4j conjointement avec log4j2.

slf4j a pour but de généraliser l'utilisation de loggers en offrant un grand nombre d'interfaces. Parmi ces dernières on y retrouve par exemple Logger qui définit un logger avec lequel on peut effectuer des opérations comme « info », « warn », « error ».

log4j2 quant à lui propose une implémentation de ces interfaces afin d'avoir un logger fonctionnel. Il est configuré dans un fichier « log4j2.xml ». Ce dernier définit deux sorties :

- Console : écrit dans la sortie standard.
- RollingFile : écrit dans un fichier
  - Change de fichier dès que ce dernier dépasse 10Mb.
  - Garde seulement les 10 derniers fichiers.

Ensuite les différents loggers sont assignés aux sorties précédemment cités avec un niveau de log minimal requis afin de filtrer les messages inutiles dans le cas d'une exécution normale.

2.3 Fichiers de résultats

En plus des logs, le simulateur produit des fichiers de sorties contenant des métriques. Ces derniers sont organisés de na manière suivante :

```
.
└── metrics
    ├── <Stimestamp> ..... Dossier d'un lancement du programme
    │   ├── config.json ..... Copie de la configuration utilisée pour ce lancement
    │   ├── <outputAll> ..... Sortie de métrique offrant une valeur par réplication
    │   ├── <outputAll>
    │   ├── ⋮
    │   ├── <Rtimestamp> ..... Dossier contenant les sorties propres à une réplication
    │   │   ├── <outputRepl> ..... Contient une sortie de métrique sur une réplication
    │   │   ├── <outputRepl>
    │   │   └── ⋮
    │   ├── <Rtimestamp>
    │   └── ⋮
    ├── <Stimestamp>
    └── ⋮
```

3 Structure du projet

Le projet suit principalement la norme de Maven avec quelques dossiers en plus :

- `libs` ..... Dossier contenant la dépendance OR-Tools pour Maven (repository local)
- `log` ..... Dossier de sortie des logs
- `metrics` ..... Dossier de sortie des métriques, voir Section 2.3
- `native` ..... Dossier contenant les dll/dylib/so pour OR-Tools
  - `<numéro de version>`
    - `windows`
    - `osx`
    - `linux`
  - `<numéro de version>`
- `src` ..... Sources du programme
  - `main` ..... Sources de production
    - `java` ..... Sources java, voir Section 1.1
    - `resources` ..... Ressources pour la production (images pour l'UI, log4j, ...)
  - `test` ..... Sources de test
    - `java` ..... Sources java pour les tests
    - `resources` ..... Ressources pour les tests (fichiers de configuration à parser
- `target` ..... Sorties de Maven (compilation, Javadoc, rapports de couverture, ...)
- `pom.xml` ..... Fichier de configuration Maven, à ouvrir avec un IDE en tant que projet

4 Mise à jour des dépendances

La plus part des dépendances sont gérées avec Maven. Afin de les mettre à jour il suffit de changer le numéro de version de ces dernières dans le « pom.xml ». Il est aussi possible d'obtenir la liste de celles non à jour grâce à la commande « mvn versions :display-dependency-updates ».

En revanche OR-Tools n'est pas disponible sur Maven. Actuellement il se trouve dans une repository local (dossier libs) afin de pouvoir l'inclure avec Maven. Si une nouvelle version veut être déployée, il faut suivre les étapes suivantes :

- Télécharger OR-Tools pour un OS quelconque. Le ficher est sous la forme « or-tools_os _vxxxx ».
- Extraire le zip, se rendre dans le dossier « lib » et garder de coté les deux fichiers jar (protobuf et com.google.ortools).
- Déplacer les fichier jar. xxxx représentera la version d'OR-Tools :
  - Placer le fichier protobuf.jar dans « Simulator/libs/com/google/protobuf/xxxx/protobuf-xxxx.jar » et renommez-le comme dans le chemin.
  - Placer le fichier com.google.ortools.jar dans « Simulator/libs/com/google/protobuf/xxxx/ortools-xxxx.jar » et renommez-le comme dans le chemin.
- La version xxxx est maintenant disponible dans le « pom.xml »>.

4.1 OR-Tools

Dans la version 6 d'OR-Tools, un problème fait que la JVM plante de manière aléatoire. Cela semble être du à un accès mémoire incohérent. Les issues suivantes peuvent être liées au problème :

- https://github.com/google/or-tools/issues/686
- https://github.com/google/or-tools/issues/621
- https://github.com/google/or-tools/issues/1091

La dernière notamment semble avoir été fixée dans la version 7. Il pourrai donc être intéressant d'envisager d'effectuer une migration vers cette dernière lorsqu'elle est disponible.

G Document d'installation du projet

1 Utilisation simple

1.1 Java

Afin de pouvoir utiliser l'application il est nécessaire d'avoir Java 12 ou supérieur d'installé sur l'ordinateur. Ce dernier peut être téléchargée ici : OpenJDK downloads (ou ici pour la version Oracle : Oracle downloads)(une version d'OpenJDK est distribuée par AdoptOpenJDK et propose un installateur afin de faciliter l'intallation de Java). Il suffit de prendre la version adaptée à l'architecture cible et suivre les étapes d'installation (peut différer selon vos besoins, néanmoins une possibilité est expliquée ici).

1.2 ORTools

Un autre pré-requis est la librairie OR-Tools. Le téléchargement se trouve ici : OR-Tools (dans le cas de Windows ou macOSX, les fichiers nécessaires sont déjà présents dans le dossier « native »). Le fichier a télécharger est nommé « or-tools_xxxxx_v6.10.6025 » où xxxx représente le système d'exploitation cible. Une fois l'archive téléchargée, il faut l'extraire et conserver le dossier « libs » nous sera utile par la suite (le dossier libs correspond aussi à « native/version/os »).

2 Utilisation développeur

Afin de pouvoir faire évoluer le code, il est nécessaire d'avoir un environnement pour coder en Java 12+. Pour cela la partie Java vue en Section 1.1 doit être réalisée. Il faut ensuite installer l'utilitaire Maven disponible ici.

Le projet est ensuite utilisable simplement comme un projet Maven (il suffit d'ouvrir le pom.xml avec par exemple IntelliJ) ce qui doit être supporté par la majorité des IDEs.

Si toutefois aucun IDE n'est utilisé, il est possible de compiler le projet avec la commande « mvn package ». Suite à cela le fichier jar se trouvera au chemin suivant : target/Simulator.jar. La commande « mvn dependency :resolve » permet quant à elle de télécharger les dépendances nécessaires (placées dans « ~/.m2 »).

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 :

Code source H.1 – *Fichier de configuration*

```
{
  "seed": 42,
  "end": 20,
  "environment": [
    {
      "class": "fr.mrcraftcod.simulator.sensors.Sensor",
      "count": 1,
      "parameters": {
        "powerActivation": 3,
        "position": {
          "class": "fr.mrcraftcod.simulator.positions.RandomPosition",
          "parameters": {
            "minX": -5,
            "maxX": 5,
            "minY": -5,
            "maxY": 5
          }
        },
        "maxCapacity": 40,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 23
          }
        }
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.chargers.Charger",
      "count": 3,
      "parameters": {
        "transmissionPower": 3,
```

```
        "radius": 40.5,
        "maxCapacity": 500,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 450
          }
        },
        "speed": 12
      }
    }
  ],
  "metrics": [
    "fr.mrcraftcod.simulator.metrics.listeners.ReplicationTotalDepletionMetricEventListener",
    "fr.mrcraftcod.simulator.metrics.listeners.ReplicationTotalChargerInactiveTimeMetricEventListener",
    "fr.mrcraftcod.simulator.metrics.listeners.ReplicationChargerCapacityUsedMetricEventListener"
  ]
}
```

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.
- **metrics** : Une liste de listeners de métriques à utiliser.

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** : objet JSON (Section 1.2.5), la capacité.
- **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** : objet JSON (Section 1.2.5), la capacité.
- **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.RaultRouter

- $\emptyset$

fr.mrcraftcod.simulator.rault.routing.RaultRouterModified

- $\emptyset$

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 $x, y \in [-max, max[$.

1.2.5 Capacité

fr.mrcraftcod.simulator.capacity.Capacity

- **value** : La valeur de la capacité.

fr.mrcraftcod.simulator.capacity.RandomCapacity

- **max** : Nombre entier, la borne maximum pour la génération aléatoire. La capacité sera telle que $capacite \in [0, max[$.

2 Lancement

Afin de lancer le simulateur en lui-même, il suffit d'ouvrir un terminal et se rendre dans le dossier contenant le fichier jar et lancer ce dernier avec « java -Djava.library.path=yyyy -jar Simulator.jar -c zzzz» où yyy est le chemin vers de dossier libs précédemment téléchargé et zzz le chemin vers le fichier de configuration (Section 1). Cette commande lancera une interface graphique avec comme instance celle définie dans le fichier zzz.

Il est aussi possible de rajouter le paramètre « –cli » afin de lancer la simulation directement dans la console sans UI. De plus dans le mode cli nous pouvons préciser le nombre de réplications que nous voulons effectuer. Par exemple en ajoutant « –replication 54 » le programme réalisera 54 simulations à la suite.

3 Interface

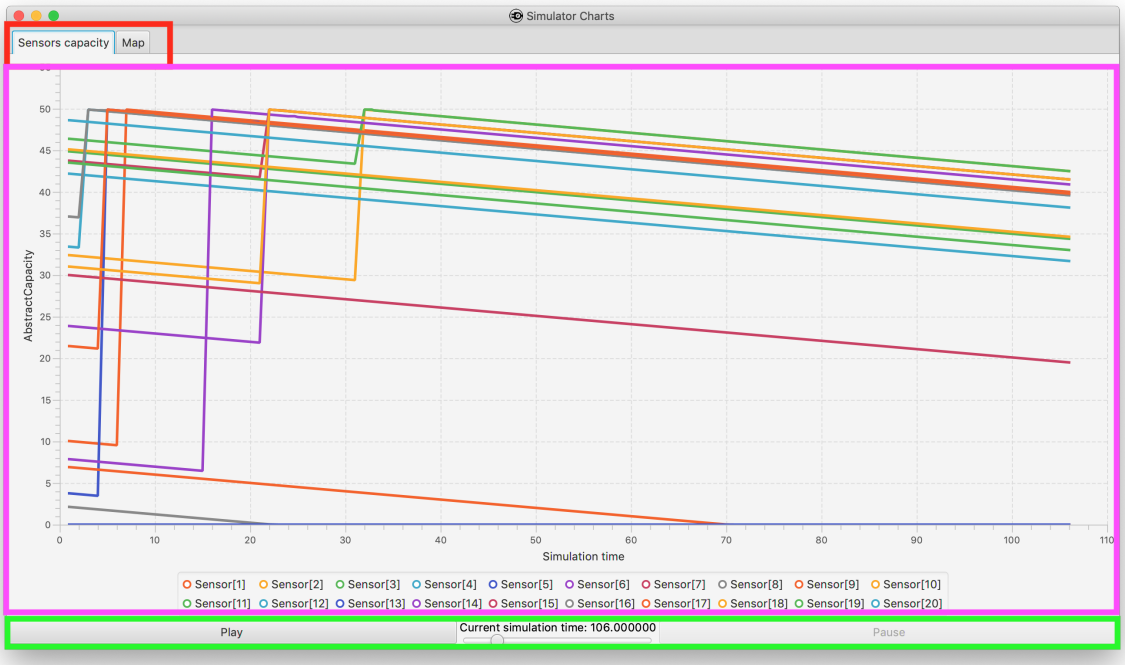


Figure 1 – *Page de démarrage de l'interface graphique*

L'image ci-dessus montre la première vue sur laquelle l'utilisateur arrivera une fois le programme lancé. Nous pouvons y observer 3 parties majeures :

- La zone rouge : nous y retrouvons des onglets qui vont afficher différents éléments chacun.
- La zone verte : cette partie regroupe des contrôles concernant la simulation :
  - Bouton « Play » pour lancer la simulation.
  - Bouton « Pause » pour mettre en pause la simulation.
  - Une indication du temps actuel de la simulation.
  - Un slider permettant d'altérer la vitesse du simulateur (un délais sera ajouté à chaque étape de la simulation). Le slider totalement à gauche n'induit aucun temps de délais, cependant plus le slider est glissé vers la droite plus le délais est important.
- La zone rose qui contient le contenu de notre onglet.

3.1 Onglet « Sensors capacity »

L'onglet sensors capacity (voir Figure 1) a pour but d'afficher une représentation sous forme de graphique de l'évolution des différentes capacités des capteurs au cours du temps. Chaque capteur à sa propre ligne représentant sa capacité. Une légende est disponible dans la partie inférieure.

3.2 Onglet « Map »

Dans cette partie de l'interface nous donnons une représentation géospatiale des différents éléments en les plaçant sur un plan.

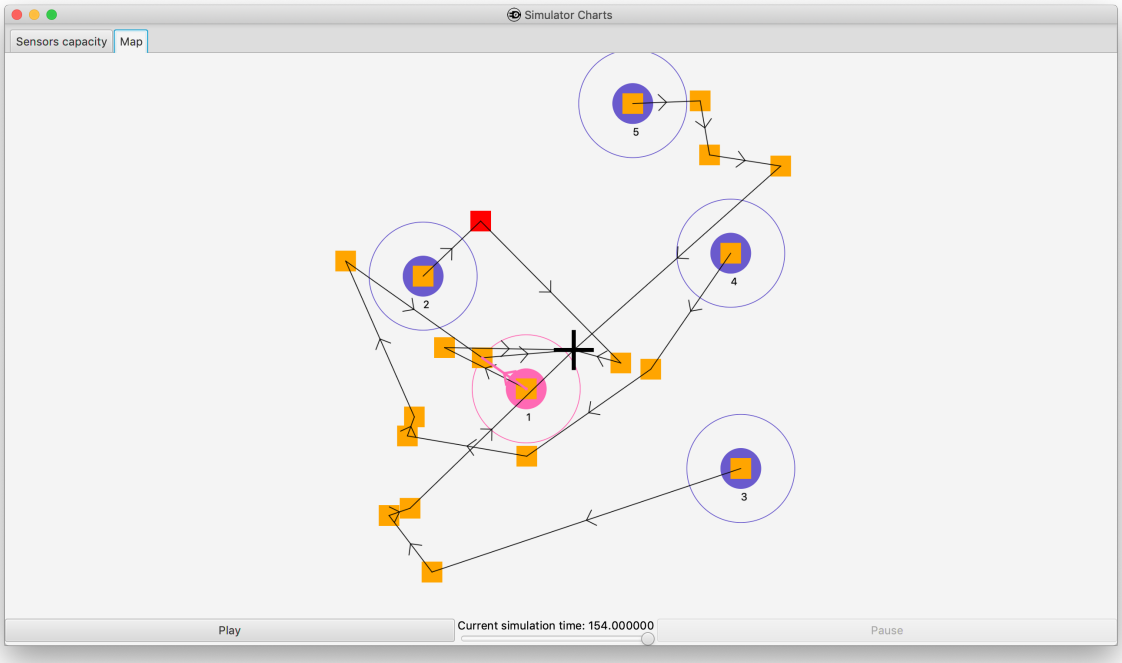


Figure 2 – *Onglet « Map » de l'interface graphique*

L'affichage est un peu dense mais les éléments suivants apparaissent :

- Une croix noire épaisse : le point marqué par cet croix représente le centre du plan (coordonnées (0;0)).

- Des carrés de couleur : ces éléments représentent les différents capteurs de la simulation et leur couleur un état :
  - Verte si la capacité est plus grande que lr.
  - Orange si la capacité est entre lc et lr.
  - Rouge si la capacité est plus petite ou égale à lc.
- Des disques de couleur : chaque disque représente un chargeur. Le cercle l'entourant représente le rayon dans lequel ses ondes EMR ont un impacte. De manière similaire la couleur permet de représenter l'état du chargeur :
  - Bleu si le chargeur est au repos.
  - Violet si le chargeur est en cours de déplacement.
  - Rose si le chargeur est en train de charger. Dans cet état des flèches roses indiqueront quels capteurs sont en train d'être chargés. De plus des flèches noires indiquent quelles routes les différents chargeurs vont emprunter.

4 Fichiers de sortie

Les fichiers de sortie se retrouvent dans le dossier « metrics ». Ce dernier contiendra un dossier par lancement du programme, puis sera éventuellement lui-même composé d'un dossier par réplication. Les fichiers seront les sorties des différentes métriques utilisées dans le fichier de configuration. Ces dernières pourront êtres définies par l'utilisateur dans le code mais certaines seront fournies de base :

- Temps total de panne des capteurs.
- Evolution du taux de panne.
- Temps total des chargeurs à être inactif à cause de contraintes.
- Capacité totale utilisée par les chargeurs.
- Evolution de la capacité des capteurs.

Ces données sont 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

Afin de valider le code il est nécessaire de mettre en place des tests. Ces derniers peuvent se concrétiser en plusieurs formes :

- Unitaire
- Fonctionnel

Les tests unitaires vont permettre de tester les fondations du code en vérifiant chaque méthode de manière indépendante. En revanche les tests fonctionnels vont se focaliser sur une fonctionnalité en général sans se soucier de la manière dont les étapes intermédiaires sont réalisées.

1 Tests unitaires

Dans le cadre de ce projet les tests unitaires vont se focaliser sur la partie modèle ainsi que sur le parser de configuration. En effet toute la partie simulation est compliquée à analyser de manière indépendante.

Effectuer ces derniers nous permettra de valider les différents qui sont utilisés par le simulateur et ainsi être sûr que le comportement souhaité est bien celui implémenté.

1.1 Tests du modèle

ID	Elément	Contenu	Statut
1	Position	Test de la distance entre deux positions	✓
2	Position	Test de l'égalité entre deux positions	✓
3	Position	Test du constructeur ainsi que des getters associés	✓
4	RandomPosition	Test que les positions générées se trouvent bien dans l'intervalle défini	✓

Table 2 – *Tests unitaires sur la partie modèle*

1.2 Tests du parser de configuration

Lors d'un test de configuration, on charge une configuration prédéfinie puis vérifions que tous les éléments sont présents et qu'ils sont initialisés avec les bon paramètres.

ID	Elément	Contenu	Statut
5	SimulationParameters	Test d'une configuration valide	✓
6	SimulationParameters	Test d'une deuxième configuration valide	✓
7	SimulationParameters	Test de cas où la capacité initiale est plus grande que la capacité maximale	✓
8	SimulationParameters	Test de cas où des paramètres sont manquants	✓

Table 4 – *Tests unitaires sur la partie parsing de la configuration*

1.3 Autre tests

ID	Elément	Contenu	Statut
9	Router	Test d'une implémentation basique d'un router	✓
10	SensorCapacityMetricEvent	Test des différents constructeurs et vérification des getters associés	✓
11	SensorsCapacityMetricEvent	Test des différents constructeurs et vérification des getters associés	✓
12	LcRequestMetricEvent	Test des différents constructeurs et vérification des getters associés	✓
13	LrRequestMetricEvent	Test des différents constructeurs et vérification des getters associés	✓
14	SensorChargedMetricEvent	Test des différents constructeurs et vérification des getters associés	✓
15	TourChargeEndMetricEvent	Test des différents constructeurs et vérification des getters associés	✓
16	TourChargeMetricEvent	Test des différents constructeurs et vérification des getters associés	✓
17	TourEndMetricEvent	Test des différents constructeurs et vérification des getters associés	✓
18	TourStartMetricEvent	Test des différents constructeurs et vérification des getters associés	✓
19	TourTravelBaseMetricEvent	Test des différents constructeurs et vérification des getters associés	✓
20	TourTravelEndMetricEvent	Test des différents constructeurs et vérification des getters associés	✓
21	TourTravelMetricEvent	Test des différents constructeurs et vérification des getters associés	✓

Table 6 – *Tests unitaires sur d'autre éléments*

2 Tests fonctionnels

Tester la partie simulateur avec une approche unitaire est très difficile puisque cela implique de l'aléatoire et des processus non indépendants. Pour cela une approche plus fonctionnelle et manuelle a été entreprise. Un jeu d'instances a été réalisé et permet de vérifier visuellement que le simulateur effectue les actions théoriques.

ID	Nom d'instance	Contenu	Statut
22	test1.json	1 capteur & 1 chargeurs permettant de vérifier le fonctionnement de base tels que les déplacements, déchargements	✓
23	test2.json	1 capteur avec valeur initiale de la capacité en dessous de lr & 1 chargeurs permettant de vérifier que le statut *lr* est quand même déclenché	✓
24	test3.json	1 capteur avec valeur initiale de la capacité en dessous de lc & 1 chargeurs permettant de vérifier que le statut *lc* est quand même déclenché	✓
25	test4.json	1 capteur & 5 chargeurs permettant de vérifier que plusieurs chargeurs sont routés correctement sans conflit	✓
26	test5.json	5 capteurs & 1 chargeur permettant de vérifier que le routage de plusieurs capteurs est bien effectué	✓
27	test6.json	5 capteurs & 5 chargeurs permettant de vérifier que plusieurs chargeurs sont routés correctement avec plusieurs chargeurs sans conflit	✓

Table 8 – *Tests fonctionnels*

2.1 Instances des tests fonctionnels

Code source I.1 – *test1.json*

```
{
  "end": 100,
  "seed": 1,
  "environment": [
    {
      "class": "fr.mrcraftcod.simulator.rault.sensors.LrLcSensor",
      "count": 1,
      "parameters": {
        "powerActivation": 1,
        "position": {
          "class": "fr.mrcraftcod.simulator.positions.Position",
          "parameters": {
            "x": 1,
            "y": 1
          }
        },
        "dischargeSpeed": 1,
        "maxCapacity": 50,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 50
```

```
          }
        },
        "lc": 15,
        "lr": 40
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.chargers.Charger",
      "count": 1,
      "parameters": {
        "transmissionPower": 5,
        "radius": 2.7,
        "maxCapacity": 1000000,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 1000000
          }
        },
        "speed": 2
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.rault.routing.RaultRouter",
      "count": 1,
      "parameters": {
      }
    }
  ]
}
```

Code source I.2 – *test2.json*

```
{
  "end": 100,
  "seed": 1,
  "environment": [
    {
      "class": "fr.mrcraftcod.simulator.rault.sensors.LrLcSensor",
      "count": 1,
      "parameters": {
        "powerActivation": 1,
        "position": {
          "class": "fr.mrcraftcod.simulator.positions.Position",
          "parameters": {
            "x": 1,
            "y": 1
          }
        },
        "dischargeSpeed": 1,
        "maxCapacity": 50,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 30
          }
        },
        "lc": 15,
        "lr": 40
      }
    },
```

```
    {
      "class": "fr.mrcraftcod.simulator.chargers.Charger",
      "count": 1,
      "parameters": {
        "transmissionPower": 5,
        "radius": 2.7,
        "maxCapacity": 1000000,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 1000000
          }
        },
        "speed": 2
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.rault.routing.RaultRouter",
      "count": 1,
      "parameters": {
      }
    }
  ]
}
```

Code source I.3 – *test3.json*

```
{
  "end": 100,
  "seed": 1,
  "environment": [
    {
      "class": "fr.mrcraftcod.simulator.rault.sensors.LrLcSensor",
      "count": 1,
      "parameters": {
        "powerActivation": 1,
        "position": {
          "class": "fr.mrcraftcod.simulator.positions.Position",
          "parameters": {
            "x": 1,
            "y": 1
          }
        },
        "dischargeSpeed": 1,
        "maxCapacity": 50,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 10
          }
        },
        "lc": 15,
        "lr": 40
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.chargers.Charger",
      "count": 1,
      "parameters": {
        "transmissionPower": 5,
        "radius": 2.7,
```

```
        "maxCapacity": 1000000,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 1000000
          }
        },
        "speed": 2
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.rault.routing.RaultRouter",
      "count": 1,
      "parameters": {
      }
    }
  ]
}
```

Code source I.4 – *test4.json*

```
{
  "end": 100,
  "seed": 1,
  "environment": [
    {
      "class": "fr.mrcraftcod.simulator.rault.sensors.LrLcSensor",
      "count": 1,
      "parameters": {
        "powerActivation": 1,
        "position": {
          "class": "fr.mrcraftcod.simulator.positions.Position",
          "parameters": {
            "x": 1,
            "y": 1
          }
        },
        "dischargeSpeed": 1,
        "maxCapacity": 50,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 50
          }
        },
        "lc": 15,
        "lr": 40
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.chargers.Charger",
      "count": 5,
      "parameters": {
        "transmissionPower": 5,
        "radius": 2.7,
        "maxCapacity": 1000000,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 1000000
          }
```

```
        },
        "speed": 2
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.rault.routing.RaultRouter",
      "count": 1,
      "parameters": {
      }
    }
  ]
}
```

Code source I.5 – *test5.json*

```
{
  "end": 100,
  "seed": 1,
  "environment": [
    {
      "class": "fr.mrcraftcod.simulator.rault.sensors.LrLcSensor",
      "count": 5,
      "parameters": {
        "powerActivation": 1,
        "position": {
          "class": "fr.mrcraftcod.simulator.positions.RandomPosition",
          "parameters": {
            "minX": -10,
            "minY": -10,
            "maxX": 10,
            "maxY": 10
          }
        },
        "dischargeSpeed": 1,
        "maxCapacity": 50,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 50
          }
        },
        "lc": 15,
        "lr": 40
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.chargers.Charger",
      "count": 1,
      "parameters": {
        "transmissionPower": 5,
        "radius": 2.7,
        "maxCapacity": 1000000,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 1000000
          }
        },
        "speed": 2
      }
    },
```

```
    {
      "class": "fr.mrcraftcod.simulator.rault.routing.RaultRouter",
      "count": 1,
      "parameters": {
      }
    }
  ]
}
```

Code source I.6 – *test6.json*

```
{
  "end": 100,
  "seed": 1,
  "environment": [
    {
      "class": "fr.mrcraftcod.simulator.rault.sensors.LrLcSensor",
      "count": 5,
      "parameters": {
        "powerActivation": 1,
        "position": {
          "class": "fr.mrcraftcod.simulator.positions.RandomPosition",
          "parameters": {
            "minX": -10,
            "minY": -10,
            "maxX": 10,
            "maxY": 10
          }
        },
        "dischargeSpeed": 1,
        "maxCapacity": 50,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 50
          }
        },
        "lc": 15,
        "lr": 40
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.chargers.Charger",
      "count": 5,
      "parameters": {
        "transmissionPower": 5,
        "radius": 2.7,
        "maxCapacity": 1000000,
        "currentCapacity": {
          "class": "fr.mrcraftcod.simulator.capacity.Capacity",
          "parameters": {
            "value": 1000000
          }
        },
        "speed": 2
      }
    },
    {
      "class": "fr.mrcraftcod.simulator.rault.routing.RaultRouter",
      "count": 1,
      "parameters": {
```

```
        }
      }
    ]
}
```

J Intégration continue

Afin d'améliorer le développement, un système d'intégration continue a été mis en place. Cela permet d'effectuer certaines actions lorsqu'une modification sur le code est envoyé sur le serveur Git. Nous pouvons entre autres faire de la vérification de code grâce à une compilation et des tests.

Voici les différentes étapes qui sont réalisées dans ce projet sont les suivantes :

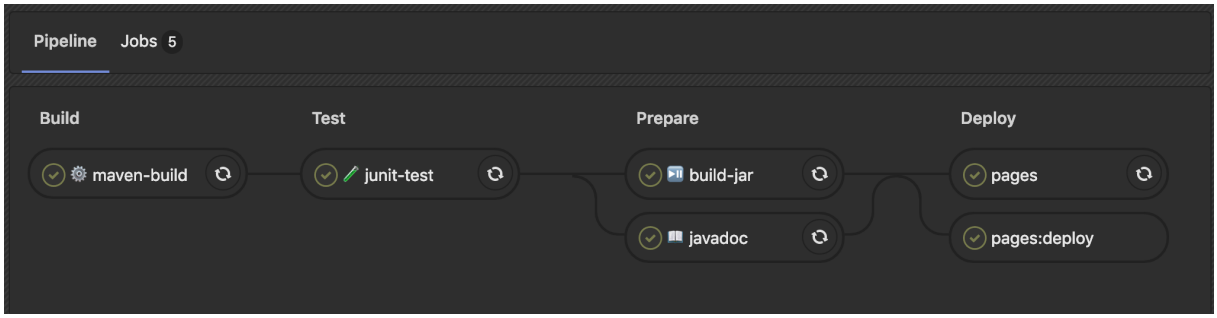


Figure 1 – *Etapes du CI*

Afin de passer à une étape suivante, toutes les tâches de l'étape doivent être validées.

- Etape build :
  - Effectue la compilation du projet
- Etape test :
  - Effectue les tests unitaires du projet
- Etape prepare :
  - build-jar : Construit un jar exécutable du projet
  - javadoc : Génère la Javadoc du projet
- deploy :
  - Déploie les différents éléments sur une page web (Javadoc, résultats des tests de couverture)

Si nous prenons l'exemple de la Javadoc, après la fin du job concerné, il est possible de télécharger cette dernière pour le code à cet instant t :

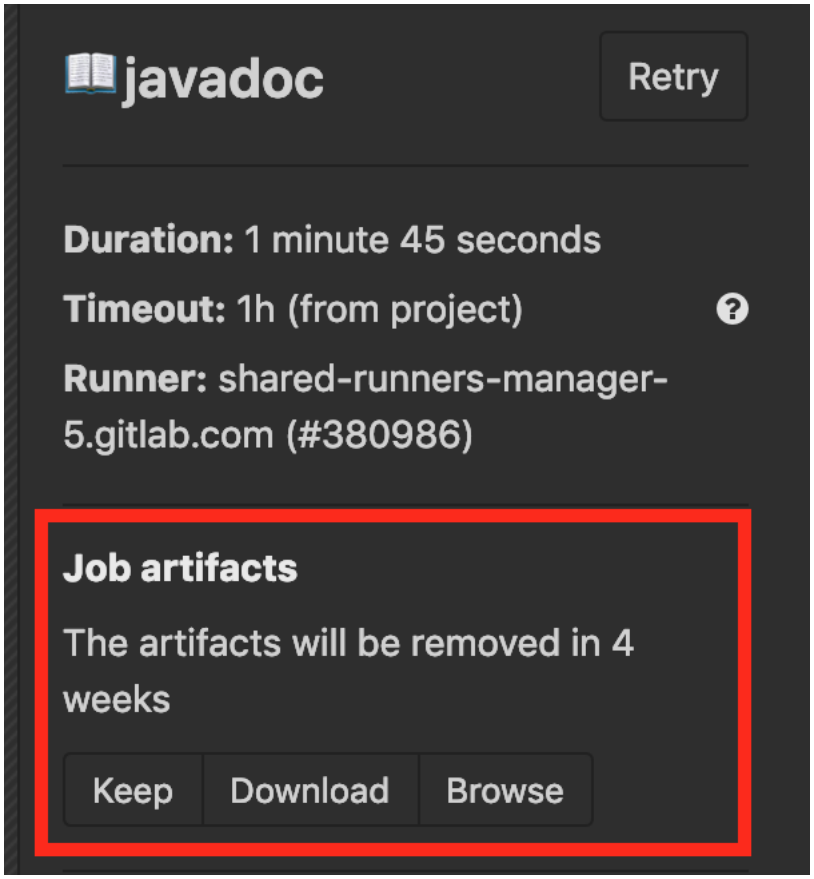


Figure 2 – *Téléchargements des fichiers du job*

De plus les tests unitaires produisent un résultat de test de couverture permettant ainsi de se rendre compte de l'efficacité de ces derniers.

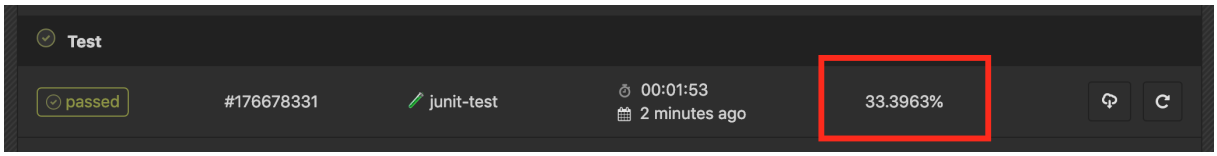


Figure 3 – *Pourcentage de couverture*

Ces résultats sont aussi disponibles sur la page principale du projet sous forme de badges.

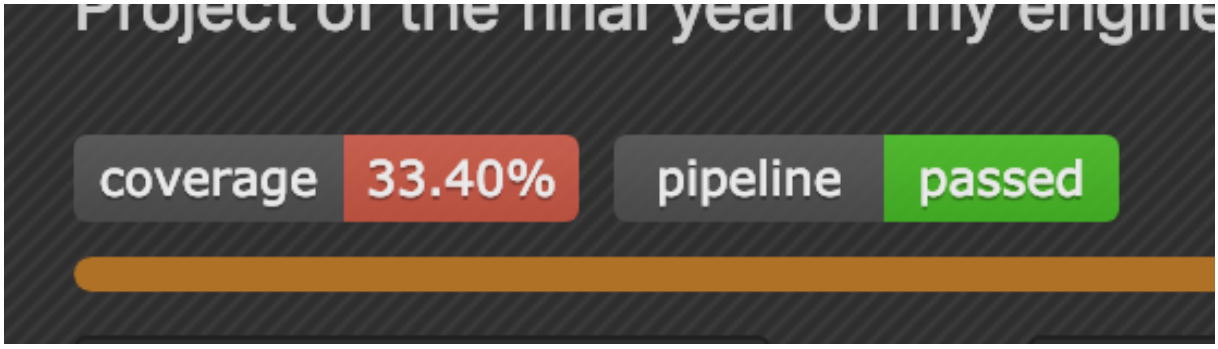


Figure 4 – *Page principale du projet*

La phase de déploiement permet de mettre en ligne la Javadoc ainsi qu'une version plus poussée su rapport de couverture des tests.

Simulator

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
fr.mrcraftcod.simulator.rault.events		0%		0%	44	44	113	113	35	35	6	6
fr.mrcraftcod.simulator.metrics.listeners		9%		0%	36	42	80	97	22	28	2	5
fr.mrcraftcod.simulator.rault.utils		0%		0%	23	23	81	81	15	15	3	3
fr.mrcraftcod.simulator		45%		44%	30	52	83	133	16	35	2	4
fr.mrcraftcod.simulator.rault.sensors		0%		0%	24	24	47	47	13	13	2	2
fr.mrcraftcod.simulator.simulation		22%		0%	22	25	48	62	17	20	1	2
fr.mrcraftcod.simulator.metrics		49%		14%	18	34	33	71	12	27	0	4
fr.mrcraftcod.simulator.utils		68%		81%	25	43	26	67	21	32	0	4
fr.mrcraftcod.simulator.chargers		71%		54%	24	43	21	76	13	31	0	1
fr.mrcraftcod.simulator.simulation.events		0%		n/a	7	7	17	17	7	7	3	3
fr.mrcraftcod.simulator.capacity		27%		0%	11	17	18	28	8	14	1	3
fr.mrcraftcod.simulator.sensors		76%		55%	16	33	11	59	7	23	0	1
fr.mrcraftcod.simulator.rault.metrics.events		89%		n/a	1	11	2	22	1	11	1	11
fr.mrcraftcod.simulator.routing		93%		n/a	1	5	1	10	1	5	0	1
fr.mrcraftcod.simulator.positions		100%		100%	0	19	0	36	0	15	0	2
fr.mrcraftcod.simulator.exceptions		100%		n/a	0	4	0	10	0	4	0	1
fr.mrcraftcod.simulator.metrics.events		100%		n/a	0	2	0	4	0	2	0	2
Total	3,173 of 4,764	33%	155 of 222	30%	282	428	581	933	188	317	21	55

Figure 5 – *Rapport JaCoCo*

K Comptes rendus hebdomadaires

Compte rendu n°1 du 23/09/2018

- Réception 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:

- Langage 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 11/11/2018

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 18/11/2018

Avancement:

- Analyse logicielle.
- Avancement dans le code.

Compte rendu n°9 du 25/11/2018

Avancement:

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

Compte rendu n°10 du 02/12/2018

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 09/12/2018

Avancement:

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

Compte rendu n°12 du 15/12/2018

- Soutenance.

Compte rendu n°13 du 23/12/2018

Avancement:

- Début d'ajout du TSPMTW.

Compte rendu n°14 du 13/01/2019

Avancement:

— Ajout du TSPMTW.
— Fix du TSP.

Compte rendu n°15 du 20/01/2019

Avancement:

— Début de la création d'une interface.

Compte rendu n°16 du 20/01/2019

Avancement:

— Corrections sur le TSP & TSPMTW.
— Ajout d'une carte dans l'interface.

Compte rendu n°17 du 27/01/2019

Avancement:

— Ajout de nouvelles métriques.
— Amélioration de la carte.

Problème:

— Le rapport ne compile plus à cause d'un package qui a eu une mise à jour. Réparation OK.

Compte rendu n°18 du 03/02/2019

Avancement:

— Correction de problèmes où la map n'affichait pas correctement le statut de certains éléments.
— Correction de NullPointerExceptions lors de la mise à jour des zones en conflit.
— Un chargeur charge tous les capteurs dans son rayon.
— Ajout d'états supplémentaires dans la map lorsqu'un chargeur est en train de charger et lorsqu'un chargeur est en cours de déplacement.
— Ajout d'animations dans la map.
— Chargement des métriques depuis le fichier de configuration.

Compte rendu n°19 du 10/02/2019

Avancement:

— Le simulateur n'a plus de champs statiques ce qui permet d'avoir un environnement isolé par réplication.
— Ajout d'un argument pour lancer la simulation sans UI.
— Ajout d'un paramètre pour lancer plusieurs réplications à la fois.

Compte rendu n°20 du 17/02/2019

Avancement:

— Correction d'un problème où certaines requêtes Lc n'étaient pas traitées.
— Ajout de tests.
— Build automatique de la Javadoc.

Compte rendu n°21 du 03/03/2019

Avancement:

— Arrête OR-Tools si le temps de calcul est trop long (permet d'éviter les blocages de ce dernier).
— Exécutions de simulation pour obtenir des premiers résultats.

Compte rendu n°22 du 10/03/2019

Avancement:

— Ajout de métriques supplémentaires.
— Correction d'un problème avec le rapport qui ne compilait plus.
— Amélioration d'OR-Tools en définissant un espace de recherche réduit.
— Ecriture du cahier de tests.
— Ecriture de la partie guide d'utilisation.
— Complétion de la partie guide d'installation.

Compte rendu n°23 du 17/03/2019

Avancement:

— Finalisation de la documentation Javadoc.
— Finalisation du cahier développeur.
— Ajout du poster.

Compte rendu n°24 du 24/03/2019

Avancement:

— utilisation du simulateur pour générer des résultats.
— Correction d'une métrique d'évènement ne contenant pas la bonne valeur.
— Correction d'un problème lié au routing.
— Ecriture de la partie mise en œuvre.

L Webographie

[WWW1] *Travelling salesman problem*. URL : https://en.wikipedia.org/wiki/Travelling_salesman_problemhttps://en.wikipedia.org/wiki/Travelling_salesman_problem.

M 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] 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.

[15] 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.

N Acronymes

EMR electromagnetic radiation. 2, 8, 9, 12, 14, 23, 75

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, 43

WET wireless energy transfer. 1

Projet Recherche & Développement

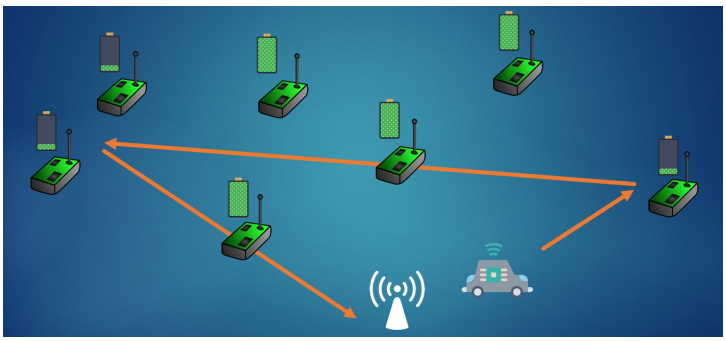
Amélioration d'un protocole de rechargement de capteurs

Thomas Couchoud

Encadrement : Tifenn Rault

Contexte

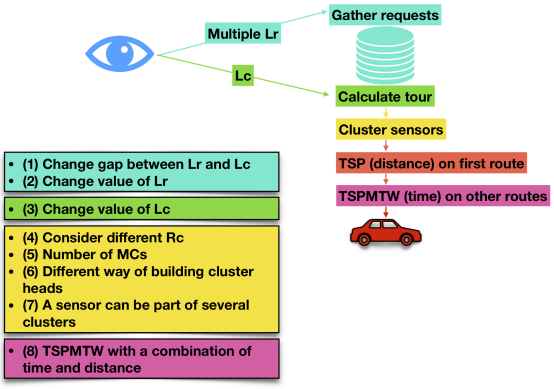
- Utilisation de chargeurs sans fils afin de recharger un cluster de capteurs.
- Contraintes sur le rechargement:
  - Chargement à distance
  - Chargement point à multipoint
  - Chargement à la demande
  - Pris en compte de contraintes d'ondes électromagnétiques



Environnement de capteurs avec un chargeur

Objectifs

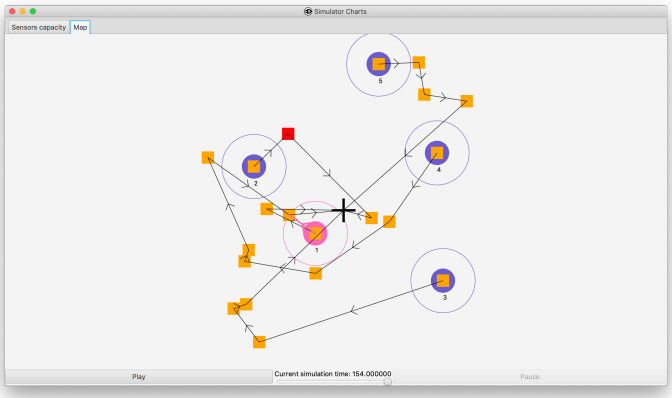
Le projet reprenant un papier de Mme Rault a pour but de partir d'une solution déjà proposée pour le problème et de l'améliorer. Afin de valider les propositions de changement un simulateur devra être réalisé.



Changements proposés

Mise en œuvre

- Utilisation d'OR-Tools
  - Résolution de TSP & TSPMTW
- Développé en JAVA
- Permet de définir ses propres types de capteurs, chargeur, algorithmes de routage et plus encore



Carte du simulateur

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Projet Recherche & Développement

Amélioration d'un protocole de rechargement de capteurs

Thomas Couchoud

Encadrement : Tifenn Rault

Contexte

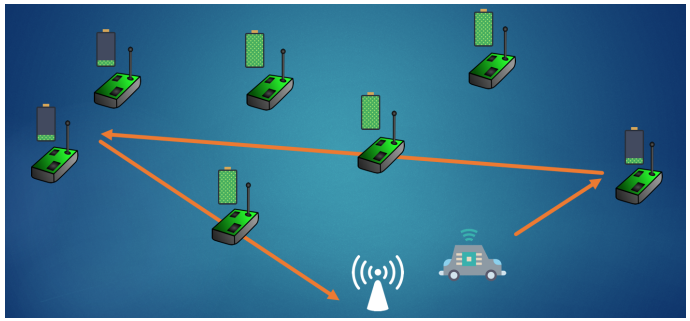
- Utilisation de chargeurs sans fils afin de recharger un cluster de capteurs.
- Contraintes sur le rechargement:
  - Chargement à distance
  - Chargement point à multipoint
  - Chargement à la demande
  - Pris en compte de contraintes d'ondes électromagnétiques

Objectifs

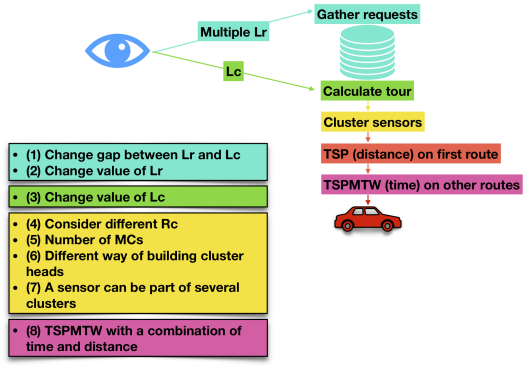
Le projet reprenant un papier de Mme Rault a pour but de partir d'une solution déjà proposée pour le problème et de l'améliorer. Afin de valider les propositions de changement un simulateur devra être réalisé.

Mise en œuvre

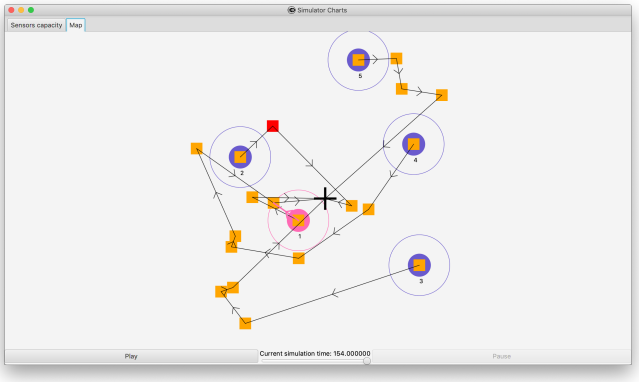
- Utilisation d'OR-Tools
  - Résolution de TSP & TSPMTW
- Développé en JAVA
- Permet de définir ses propres types de capteurs, chargeur, algorithmes de routage et plus encore



Environnement de capteurs avec un chargeur



Changements proposés



Carte du simulateur

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Informatique

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

Tuteur académique
Tifenn Rault

Étudiant
Thomas Couchoud (DI5)