AURÉLIEN PLISNIER

Computing project

Parcels on the Move

Esteban Zimanyi, Stefan Eppe PROJ-H-402

Abstract

In this paper the setting-up of a parcel handling system for an imaginary delivery company is discussed. The system at the state of proof of concept supports a parcel life cycle from the transport order to the delivery. It is composed of four applications. A PHP console supports transport orders encoding and visualization. A Java desktop application solves instances of the Pickup and Delivery Problem and plans delivery routes. A Java Android application allows delivery men to monitor their route. Finally, a Java HTTP server receives messages from the Android application and updates a central database accordingly.
Beforehand, a formal Paired Pickup and Delivery Problems is described, then a metaheuristic called "Ruin and Recreate" is studied and used to solve it. Finally, some of the main parcel identification techniques are briefly discussed.

Dans ce rapport la mise sur pied d'un système de gestion de colis destiné à une entreprise de colis express fictive est décrite. Le système, à l'état de preuve de concept, prend en charge le cycle de vie d'un colis de la commande à la livraison chez le destinataire. Il est composé de quatre applications: une console PHP pour l'encodage et la visualisation des ordres de transport, un programme Java de résolution de problèmes de type "Pickup and Delivery" pour la création d'itinéraires, une application Android pour smartphone permettant aux transporteurs de monitorer leur itinéraire et finalement un serveur HTTP codé en Java recevant les requêtes de l'application Android et les traduisant en actions sur une base de données centrale.
Le problème "Paired Pickup and Delivery" est formellement décrit et, en vue de sa résolution, une métaheuristique appelée "Ruin and Recreate" est étudiée. Les principales techniques d'identification de colis sont aussi brièvement discutées.

Sommaire

List of Figures

Préface

J'ai choisi le sujet de ce projet car toutes les technologies qu'il nécessitait de mettre en oeuvre étaient, à l'exception de MySQL, HTTP et Java, nouvelles pour moi. Ce fut en particulier l'occasion de faire l'acquisition de mon premier smartphone et de découvrir Android, de m'initier aux technologies RFID et NFC, d'étudier une métaheuristique.
Je remercie le Professeur E. Zimanyi et S. Eppe de m'avoir guidé, encouragé et soutenu dans cette quête.

Chapter 1

Introduction

Parcels on the Move est une petite entreprise familiale de colis express basée à Bruxelles. Jusqu'il y a peu, la gestion des colis et des itinéraires des transporteurs se faisait de manière non numérique. Réputée pour son sérieux et pour sa qualité de service, l'entreprise élargit sa clientèle professionnelle de manière régulière. Elle doit maintenant faire face à un afflux de commandes tel que la mise en place d'un système intégré de gestion des colis paraît indispensable.

Le but de ce projet est de poser les bases de ce système de gestion de colis. Le système doit couvrir le cycle de vie d'un colis de l'encodage jusqu'à la livraison chez le destinataire. Il faudra donc développer une base de données permettant de conserver les ordres de transports et les informations sur les colis et une application desktop dont le rôle sera de calculer des itinéraires optimisés pour la flotte de transporteurs. Enfin, l'utilisation d'une application mobile permettra aux transporteurs de tenir le siège de l'entreprise au courant de leur progression de manière automatique.

Le nombre de fonctionnalités envisageables est très important et leur implémentation dépasse le cadre de ce projet. C'est pourquoi ce dernier se veut avant tout être une preuve de concept: nous négligerons en grande partie les aspects d'interface utilisateur, de sécurité lors de la communication HTTP et de fonctionnalités utiles n'intervenant pas dans la conception de base du système. Des pistes d'amélioration et des idées de fonctionnalités supplémentaires seront cependant discutées à la section 5.3 de ce rapport.

1.1 Portée du Projet

Dans cette section nous délimitons la portée exacte du projet.

1.1.1 Inclus

Dans ce projet, nous considérons:

La création d'une base de données temporelle permettant d'enregistrer des informations sur les clients, les ordres de transport, les colis concernés par ces ordres, les transporteurs à disposition, les itinéraires de livraison et la progression de chaque livraison. Voir la section 4.2.

L'implémentation d'une console PHP permettant d'encoder des ordres de transport et de les enregistrer en base de données ainsi que de visualiser les ordres et colis courants, les itinéraires et les clients. Voir la section 4.3.

L'implémentation d'une application desktop permettant de calculer des itinéraires optimaux pour la flotte de transporteurs à partir des ordres de transport courants enregistrés en base de données via des appels à une librairie implémentant une métaheuristique. Dans ce but, une formalisation de notre

problème sera présentée et une métaheuristique permettant de le résoudre efficacement sera discutée. Voir la section 4.4.

L'implémentation d'une application pour smartphone Android permettant de scanner des codes-barres et des identifiants NFC et RFID. A chaque scan l'application communique avec la base de données par l'envoi de requêtes HTTP à un serveur de manière à tenir à jour le statut des colis en cours de livraison. Voir la section 4.5.

L'implémentation d'un serveur desktop répondant aux requêtes de l'application smartphone et les traduisant en actions sur la base de données. Le serveur se chargera aussi d'envoyer des notifications par email aux clients à chaque changement de statut des colis. Voir la section 4.6.

1.1.2 Non Inclus

Dans ce projet, nous ne considérons pas:

L'implémentation et la sécurisation d'un Web Service..

L'implémentation d'une véritable application desktop de gestion des ordres, des facturations et du personnel.

L'implémentation d'un véritable site web de commande à l'usage des clients.

Chapter 2

Formalisation du Problème et Résolution

2.1 Description du problème

L'entreprise souhaite la mise sur pied d'un système de communication portable, peu coûteux, simple et rapide d'emploi permettant la synchronisation en temps réel du statut de chaque colis. Ces informations sont nécessaires entre autres pour la gestion des plans de route. Le problème auquel est confrontée l'entreprise lors de la création des plans de route s'apparente à un paired pickup and delivery problem. Commençons par une description puis une formalisation de ce problème tirée de [8]:

2.1.1 Paired Pickup & Delivery Problem

Le Paired Pickup & Delivery Problem (nous l'abrévierons PDP par souci de cohérence avec notre source [8]) est une sous-classe de Vehicule Routing Problem (VRP), lui-même cousin proche du célèbre Travelling salesman problem (TSP).
Ce sont des problèmes d'optimisation combinatoire NP-difficiles: il n'existe pas d'algorithme permettant de les résoudre de manière exacte en un temps raisonnable. Il est cependant possible de faire appel à des méthodes de résolution approchée offrant des solutions proches de l'optimum en un temps acceptable. Une de ces méthodes est discutée à 2.4.

TSP et VRP sont très proches conceptuellement. Le TSP présente un voyageur de commerce devant visiter chaque ville d'un ensemble exactement une fois en minimisant le coût total de voyage, souvent assimilé à la distance totale parcourue. Le voyageur commence et termine son voyage à la même ville.
Le VRP considére une flotte de véhicules devant desservir un ensemble de destinations et il faut, ici aussi, minimiser la distance totale parcourue par les véhicules. Les véhicules commencent et terminent leur itinéraire à un entrepôt central.

Le VRP peut faire l'objet de très nombreuses contraintes en fonction de l'application. Aux véhicules sont souvent associés une capacité à ne pas dépasser et un coût relatif par unité de distance parcourue. La flotte peut être considérée comme homogène, tous les véhicules ayant les mêmes caractéristiques, ou hétérogène. Le service peut consister à transporter des colis d'un entrepôt central vers un ensemble de clients, ou bien à transporter des colis entre expéditeurs et destinataires, ou même à transporter des personnes. Chaque cas apporte son lot de contraintes supplémentaires. Des contraintes temporelles peuvent aussi être considérées: tel ou tel client ne peut être visité qu'à telle ou telle heure de la journée, la durée maximale de chaque itinéraire ne peut pas dépasser la durée d'une journée de travail standard.

Ce qu'il faut retenir ici, c'est qu'il existe une grande variété de VRP et que les VRP réalistes sont, de manière générale, des problèmes fortement contraints. De très faibles changements à une solution suffisent souvent à ne plus répondre à toutes les contraintes: leur espace de solutions admissibles est dit

fortement discontinu. Cet aspect des VRP a son importance pour le choix d'une méthode de résolution approchée, comme nous le verrons par la suite.

Définissons enfin le PDP dont la résolution nous occupera lors de ce projet: sa caractéristique principale est que les destinations sont couplées. Chaque couple comprend un expéditeur, chez qui un colis doit être pris en charge, et un destinataire, chez qui le colis doit être livré. Il faut donc veiller à ce que les deux parties d'un couple expéditeur-destinataire soient visitées par le même véhicule et dans le bon ordre, de manière à ce que les colis puissent être correctement livrés.

2.1.2 Notations

Nous commençons notre formalisation du PDP par la liste des notations employées.

n est le nombre de noeuds pour lesquels un colis doit être pris en charge.

$\tilde{n}$ est le nombre de noeuds pour lesquels une livraison doit être effectuée.

P est l'ensemble des noeuds de prise en charge, $P = \{1, ..., n\}$

D est l'ensemble des noeuds de livraison, $D = \{n+1, ..., n+\tilde{n}\}$

V est l'ensemble de tous les noeuds. $V = \{0, n+\tilde{n}+1\} \cup P \cup D$

A est l'ensemble de tous les arcs. $A = \{(i,j) : i, j \in V, i \neq n+\tilde{n}+1, j \neq 0, i \neq j\}$

K est l'ensemble des véhicules.

d_i est le temps passé sur le noeud i pour effectuer le service.

c_{ij}^k est le coût de parcours de l'arc reliant les noeuds i et j pour le véhicule k.

t_{ij}^k est le temps de parcours de l'arc reliant les noeuds i et j pour le véhicule k.

C^k est la capacité du véhicule k.

T^k est la durée maximale de course pour le véhicule k

x_{ij}^k est la variable de décision: vaut 1 si l'arc reliant les noeuds i et j est parcouru par le véhicule k.

Q_i^k est la charge du véhicule k en quittant le noeud i.

q_i est la charge associée à chaque noeud: positive pour un noeud de prise en charge, négative pour un noeud de livraison, nulle pour le dépôt: $q_0 = q_{n+\tilde{n}+1} = 0$.

B_i^k est le moment de début de service du véhicule k au noeud i.

2.1.3 Equations

Nous présentons ici notre problème sous forme d'équations. Les équations 1 à 9 décrivent un VRP avec capacité de véhicules limitée. Les équations 10 à 12 particularisent le VRP en PDP. L'équation 13 spécifie que les itinéraires ne peuvent pas dépasser une certaine durée. Chaque équation est décrite textuellement.

$$min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k \qquad (1)$$

$$\sum_{k \in K} \sum_{j:(i,j) \in A} x_{ij}^k = 1 \qquad \forall i \in P \cup D \qquad (2)$$

$$\sum_{j:(0,j)\in A} x_{0j}^k = 1 \qquad \forall k \in K \qquad (3)$$

$$\sum_{i:(i,n+\tilde{n}+1)\in A} x_{i,n+\tilde{n}+1}^k = 1 \qquad \forall k \in K \qquad (4)$$

$$\sum_{i:(i,j)\in A} x_{ij}^k - \sum_{i:(i,j)\in A} x_{ji}^k = 0 \qquad \forall j \in P \cup D, k \in K \qquad (5)$$

$$x_{ij}^k = 1 \Rightarrow B_j^k \geq B_i^k + d_i + t_{ij}^k \qquad \forall (i,j) \in A, k \in K \qquad (6)$$

$$x_{ij}^k = 1 \Rightarrow Q_j^k = Q_i^k + q_j \qquad \forall (i,j) \in A, k \in K \qquad (7)$$

$$max\{0, q_i\} \leq Q_i^k \leq min\{C^k, C^k + q_i\} \qquad \forall i \in V, k \in K \qquad (8)$$

$$x_{ij}^k \in \{0,1\} \qquad \forall (i,j) \in A, k \in K \qquad (9)$$

$$n = \tilde{n} \qquad (10)$$

$$\sum_{j:(i,j)\in A} x_{ij}^k - \sum_{j:(n+i,j)\in A} x_{n+i,j}^k = 0 \qquad \forall i \in P, k \in K \qquad (11)$$

$$B_i^k \leq B_{n+i}^k \qquad \forall i \in P, k \in K \qquad (12)$$

$$B_{n+\tilde{n}+1}^k - B_0^k \leq T^k \qquad \forall k \in K \qquad (13)$$

(1): La fonction objective minimise le coût total de routage.

(2): Chaque noeud doit être servi exactement une fois.

(3) et (4): Chaque véhicule commence et termine sa course au dépôt (cela ne signifie pas pour autant que chaque véhicule doit être utilisé).

(5): Conservation de flux.

(6): Contrainte de temps: un véhicule ne peut arriver au noeud j qu'après avoir pris le temps de servir le noeud précédent(i) et d'effectuer le déplacement de i à j.

(7) et (8): La capacité d'un véhicule n'est pas excédée durant sa course. Q_i^k représente la charge totale du véhicule k au noeud i.

(9): x_{ij}^k est une variable de décision: vaut 1 si le véhicule k effectue la liaison entre les noeuds i et j.

(10): L'égalité $n = \tilde{n}$ exprime que à chaque point de prise en charge est associé un point de livraison.

(11): Le point de prise en charge et le point de livraison associé doivent être desservis par le même véhicule.

(12): Une livraison ne peut avoir lieu qu'après une prise en charge.

(13): Contrainte de temps: l'itinéraire alloué à un véhicule ne peut pas dépasser une certaine durée. Permet d'éviter d'allouer un itinéraire de 30 heures à un employé ne travaillant que 8 heures par jour.

2.2 Hypothèses

Nous listons ici les hypothèses et simplifications prises lors de la résolution de notre PDP.

- Le coût du trajet d'un client vers un autre est directement proportionnel à la distance à vol d'oiseau entre les deux clients.
- Nous nous limitons à une dimension, le poids, pour caractériser la capacité des véhicules.
- Pas de défaut de client: aucun colis n'est acheminé vers ou depuis l'entrepôt de l'entreprise.
- Il y a autant de clients émetteurs que de clients récepteurs.
- L'ensemble des émetteurs et l'ensemble des récepteurs sont disjoints.
- Chaque client peut être visité à n'importe quel moment de la journée.

2.3 Choix d'une Méthode de Résolution

Comme expliqué à 2.1.1, le PDP considéré est un problème NP-difficile. Il ne sera donc pas possible de le résoudre de manière exacte et il nous faudra nous contenter d'une résolution approchée.
La plupart des métaheuristiques partent d'une solution admissible initiale et l'améliorent itérativement en explorant un voisinage de la solution courante. Nous distinguons deux classes de métaheuristiques en fonction de la taille du voisinage exploré à chaque itération: les méthodes de "Local Neighborhood Search" explorent à chaque itération des solutions très proches de la solution courante tandis que les méthodes de "Large Neighborhood Search" explorent des solutions éloignées de la solution courante.

Rappelons une des caractéristiques principales du VRP auquel nous faisons face: c'est un problème fortement contraint: de tout petits changements à la solution courante suffisent souvent à sortir de l'espace des solutions admissibles. Le nombre de mouvements permis, si on se limite à des techniques de "Local Search Neighborhood", est très limité et peu de solutions admissibles sont explorées en fin de compte.

Nous proposons donc l'utilisation d'une métaheuristique de "Large Neighborhood Search". Effectuer des sauts de géant depuis la solution courante permet d'explorer un nombre significativement supérieur de solutions admissibles et offre généralement de meilleurs résultats pour les problèmes fortement contraints.[6]

2.4 La Métaheuristique: le Principe "Ruin and Recreate"

Cette section est principalement tirée de [7]. L'idée ici est de partir d'une solution initiale, de la détruire partiellement puis de reconstruire une nouvelle solution qui sera acceptée ou non en fonction de critères prédéfinis. Cette méthode est particulièrement adaptée aux problèmes d'optimisation de routage les plus complexes ayant un espace de solutions admissibles fortement discontinu, puisque:

- Les destructions et reconstructions successives permettent d'explorer des solutions éloignées du voisinage direct de la solution initiale.
- Son potentiel de parallélisation: la partie la plus coûteuse en ressource est le calcul du coût de prise en charge d'une destination par un véhicule. Ce calcul peut être parallélisé pour chaque véhicule, puisque les véhicules sont indépendants.

Une mise en application du principe suit les 6 étapes suivantes:

1. Construction d'une solution initiale.

2. Choix d'une méthode de destruction.

3. Choix d'un nombre de destinations à évincer de la solution.

4. Destruction: éviction des destinations.

5. Reconstruction: réinsertion des destinations.

6. Acceptation de la nouvelle solution en fonction d'une règle de décision.

2.4.1 Destruction

Ici il s'agit de détruire partiellement la solution courante: par détruire nous entendons effacer toute une partie de la solution. Nous retirons un ensemble de clients de l'itinéraire courant et relions les clients restant de manière à obtenir un itinéraire amondri: la solution "détruite".
Il y a de nombreuses manières de procéder à la destruction d'une solution, en voici quelques-unes:

Stochastique: Une destination est retirée de l'itinéraire en fonction d'une variable aléatoire.

Radiale: Chaque destination dans un rayon spatial autour d'une destination est retirée de l'itinéraire.

Temporelle: Chaque destination servie dans une certaine fourchette de temps est retirée.

Capacité: Chaque destination requérant une capacité comprise dans une certaine fourchette de poids est retirée.

Séquentielle: Un certain nombre de destinations successives est retiré de la solution.

Remarquons que tandis que la première méthode évince ou non chaque destination indépendamment l'une de l'autre, les quatre suivantes évincent un ensemble de solutions adjacentes selon un critère choisi. Notre application emploie deux méthodes de destruction partielle: stochastique et radiale.[2]

2.4.2 Reconstruction

La partie la plus complexe de l'algorithme: l'ensemble des destinations évincées est parcouru dans un ordre aléatoire et pour chaque destination nous procédons à une "best insertion": pour chaque véhicule nous examinons sa possibilité de desservir la destination, à quel moment dans son itinéraire et à quel coût supplémentaire. L'insertion de coût minimum est ensuite choisie. Dans le cas où une destination ne peut pas être insérée dans le système sans violation de contraintes, un nouveau véhicule est ajouté.
De manière générale la reconstruction consomme 90% du temps de calcul total de la méthode.

2.4.3 Critères d'acceptation

Plusieurs règles de décision peuvent être appliquées pour l'acceptation ou le rejet de nouvelles solutions. En voici quelques-unes:

Random Walk (RW): Chaque nouvelle solution est acceptée.

Greedy Acceptance (GRE): Chaque solution meilleure que la solution courante est acceptée.

Simulated Annealing (SA): Chaque meilleure solution est acceptée, et chaque solution moins bonne que la solution courante peut être acceptée avec une certaine probabilité.

Threshold Accepting (TA): Chaque solution qui n'est "pas bien pire" que la solution courante est acceptée. "Pas bien pire" étant défini par un palier.

Great Deluge Algorithm (GDA): Chaque solution en dessous d'un certain niveau de qualité (le "niveau d'eau") est refusée.

Les critères utilisés dans notre application sont GRE et TA. [2]

Chapter 3

Techniques d'Identification de Colis

Dans ce chapitre, nous proposons une courte étude des techniques d'identification de colis exploitées lors de ce projet.

3.1 RFID

Cette section est principalement tirée de [12]. RFID (radio frequency identification) est un système capable de communiquer l'identité d'une personne ou d'un objet par ondes radio. Un tag RFID est constitué d'une puce connectée à une antenne. Outre un numéro d'identification, la puce peut contenir d'autres informations comme la date de fabrication et la date limite d'utilisation d'un produit. Le contenu d'une puce peut être chiffré.

Un lecteur doit être utilisé pour récupérer les données stockées sur un tag RFID. Le lecteur envoie des ondes radio au tag et reçoit en retour les informations stockées sur la puce.
Deux grandes familles de tags RFID sont à distinguer:

- Les tags **passifs** ne possèdent pas de source d'énergie embarquée. Ils tirent leur énergie du lecteur de tag par induction. La distance de lecture est donc très courte, à peine quelques mètres. De manière générale, les tags passifs sont bon marché (moins de trente cents), ont une mémoire réduite et une durée de vie pratiquement illimitée. Ils peuvent être utilisés par exemple pour stocker le numéro d'identification d'un article de faible coût.
- Les tags **actifs** possèdent leur propre source d'énergie: typiquement une batterie lithium d'une durée de vie d'une dizaine d'années. Comme ils ne tirent pas leur énergie du lecteur, un signal beaucoup plus faible suffit à les activer et ils peuvent être lus jusqu'à une centaine de mètres. Leur mémoire plus importante, quelques kilobytes, leur permet de contenir toutes sortes d'informations sur un produit. En raison de leur coût, les tags actifs ne sont utilisés que pour les objets les plus chers.

Les tags RFID peuvent aussi être distingués selon leur fréquence de fonctionnement:

Bande	Fréquence	Distance	Application
Basse Fréquence	$100 - 500KHz$	$50cm$	Contrôle d'accès, identification animale, clé de véhicule.
Haute Fréquence	**13,56MHz**	**1m**	Contrôle d'accès, cartes intelligentes, **identification d'objets**, bibliothèques, protecion anti-vol.
Ultra Haute Fréquence	$866 - 956MHz$	$6m$	Gestion de bagages, payages, chaînes d'approvisionnement.
Micro-Ondes	$2,45GHz$	$3m$	Suivi d'objets, payages.

Nous utiliserons les tags RFID haute fréquence: ils fonctionnent à la même fréquence que NFC et peuvent donc être lus par un lecteur NFC de smartphone. De plus, c'est la famille de tags RFID dominante pour l'identification de marchandise.

3.2 NFC

Cette section est principalement tirée de [14] et [15]. NFC est l'abréviation de Near Field Communication: un standard de communication radio défini par Nokia, Philips et Sony, opérant à la fréquence de $13,56MHz$ et à très faible distance, une dizaine de centimètres typiquement[16]. L'architecture de base d'un tag NFC est la même que celle d'un tag RFID passif, la seule vraie différence étant que le tag NFC est formaté pour être lu par un lecteur NFC. Il est d'ailleurs possible de lire certains tags RFID avec un lecteur NFC comme on en trouve de plus en plus souvent dans nos smartphones. Quatre types de tags NFC peuvent être distingués en fonction de leur mémoire, débit de transmission et nombre de réécritures permises. Selon [13]:

- **Type 1:** Peut être lu et réécrit à volonté. L'utilisateur peut le configurer pour ne permettre que la lecture. Possède 96 bytes de mémoire et un débit de 106 kbit/s.
- **Type 2:** Peut être lu et réécrit à volonté. L'utilisateur peut le configurer pour ne permettre que la lecture. Possède 48 bytes de mémoire et un débit de 106 kbit/s.
- **Type 3:** Ne permet que la lecture. Possède 2 kbytes de mémoire et offre un débit de 212 kbit/s.
- **Type 4:** Ne permet que la lecture. Possède jusqu'à 32kbytes de mémoire et offre un débit allant de 106kbits/s à 424 kbits/s.

3.3 Codes-barres

Cette section est tirée de [17]. Le code-barres et un système d'encodage de données inspiré initialement du code Morse. Il existe deux grandes familles de codes-barres:

- Les codes-barres **unidimensionnels** représentent l'information sous forme de lignes verticales de largeur et d'espacement variables.
- Les codes-barres **bidimensionnels** représentent l'information à l'aide de formes géométriques réparties selon deux axes. Le QR-code est un exemple notoire de code-barres bidimensionnel.

Les codes-barres restent aujourd'hui le moyen d'identification de marchandise le plus utilisé, en raison d'un coût de mise en oeuvre d'environ 0,005 USD l'unité. Un lecteur classique de codes-barres est constitué d'une cellule photosensible et d'une source lumineuse. Il existe aussi de nombreuses applications gratuites pour smartphone permettant de lire toutes sortes de codes-barres à partir de la caméra intégrée.

Chapter 4

Implémentation

Dans ce chapitre nous décrivons l'implémentation de notre système. Nous expliquons nos choix technologiques du point de vue hardware et du point de vue software puis nous décrivons notre base de données et les applications constituant notre projet dans leur ordre typique d'utilisation.
Le colis lors de son cycle de vie est d'abord encodé en base grâce à la console de gestion des ordres de transport.

Les itinéraires de prises en charge et de livraisons sont ensuite calculés par le calculateur d'itinéraire. La progression de chaque transporteur est monitorée et enregistrée en base grâce à l'application Java Android et au serveur HTTP traduisant les requêtes de cette dernière en actions sur la base.

4.1 Choix technologiques

4.1.1 Le hardware

Nous choisissons d'utiliser un smartphone NFC Android pour plusieurs raisons:

- La possibilité de scanner des codes-barres grâce à la caméra embarquée.
- La possibilité de scanner des tags NFC et certains tags RFID grâce au lecteur NFC embarqué.
- La possibilité de se connecter à internet pour joindre le serveur distant.
- La possibilité de contacter ou d'être contacté par le siège.
- Le prix réduit de la solution: le Huawei Ascend utilisé pour ce projet a couté environ € 150.
- La présence d'une puce GPS incorporée au smartphone.

4.1.2 Le software

Java est utilisé pour le serveur du siège ainsi que l'application de calcul d'itinéraires et Java Android pour l'application smartphone, compte-tenu de notre expérience en Java.

La librairie Jsprit est employée pour l'application de calcul d'itinéraires en raison de sa facilité d'utilisation et la possibilité de définir finement et résoudre rapidement un Vehicule Routing Problem de notre choix.

Nous utilisons MySQL pour la base de données du siège: MySQL est adapté à l'implémentation d'une base de données opérationnelle devant gérer des aspects temporels et géographiques.

Nous faisons appel à l'application externe Zxing pour la lecture des codes-barres par le smartphone. Zxing est gratuit, open-source et facilement intégrable. [1] [3]

Http est utilisé pour la communication entre l'application Android et le serveur gérant la base de données: étant donnée la qualité variable du réseau accessible depuis un smartphone, il est préférable d'utiliser un protocole stateless ajoutant peu d'overhead.

Le framework Jersey [10] a été utilisé pour l'implémentation du serveur HTTP: ayant peu d'expérience en ce domaine, nous l'avons choisi en raison de l'existence de bons tutoriels expliquant comment l'utiliser. [9]

4.2 La Base de Données

La base de données a été conçue au début du projet. L'idée était de couvrir un maximum d'aspects de la gestion des colis et de servir de support à une extension future du projet. Certaines tables et certains attributs ne sont donc pour le moment pas exploités. Nous les conservons pourtant pour illustrer nos idées de fonctionnalités futures pour le système, listées à 5.3.

Dans cette section nous présentons le schéma de la base de données et nous expliquons le rôle des tables et des attributs dont le nom n'est pas assez explicite.

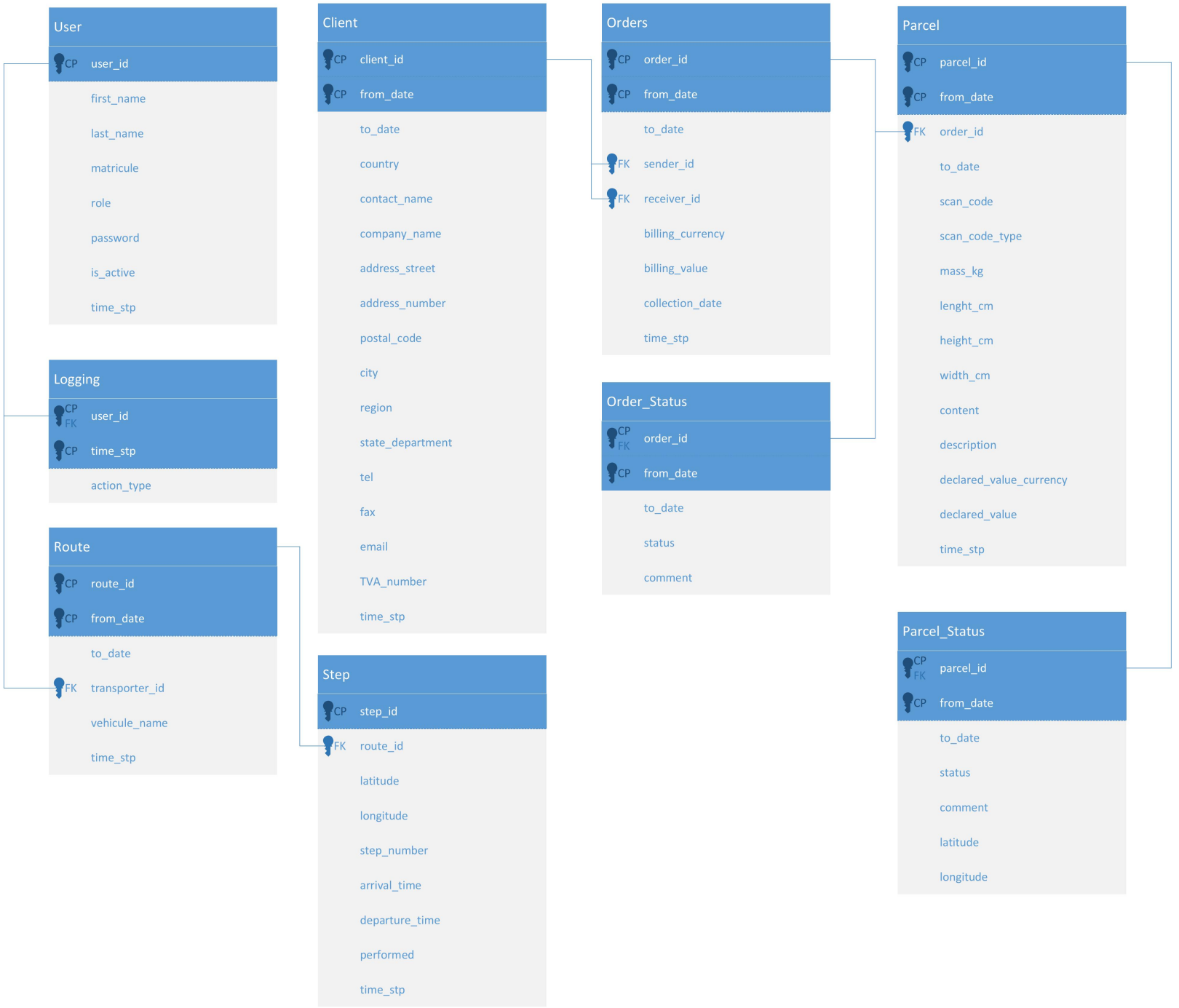

Figure 4.1: Schéma de la base de données

- **User** contient les utilisateurs de l'application: des transporteurs. Les attributs "is_active" et "role", non exploités, sont destinés à faciliter la gestion des utilisateurs.
- **Client** contient les clients: expéditeurs et destinataires.
- **Orders** contient les ordres de transport. L'attribut "collection_date", non exploité, doit permettre de traiter le cas où la prise en charge d'un colis ne peut pas se faire dès que la commande a été faite par le client.
- **Order_Status** est une table non exploitée qui doit permettre aux employés de gestion de monitorer la progression d'un ordre de transport: est-il traité, en cours de traitement, à traiter, le paiement a-t-il été reçu, etc ?
- **Parcel** contient les colis pris en charge par l'entreprise.
- **Parcel_Status** enregistre l'historique des statuts des colis. Dans notre preuve de concept nous considérons uniquement trois statuts: chez l'expéditeur, en transit et livré. A chaque transition le lieu et l'instant du scan sont enregistrés avec le nouveau statut. L'attribut "comment", non utilisé, pourra contenir un commentaire éventuel de la part du transporteur lorsqu'il scan le colis.
- **Route** contient les itinéraires. L'attribut "transporter_id", non utilisé, permet d'attribuer l'itinéraire à un employé. L'attribut "vehicule_name" propose l'emploi d'un véhicule spécifique pour effectuer la course. Il faut aussi souligner que chaque itinéraire a une durée de validité de 15h ("to_date" est initialisé à "from_date" + 15h): les itinéraires sont calculés tôt le matin puis parcourus (ou non) en journée. Le lendemain il faut refaire le calcul d'itinéraires de manière à prendre en compte les commandes de la veille.
- **Step** contient les étapes constitutives d'un itinéraire. Pour chaque étape nous conservons ses coordonnées, son ordre de parcours dans l'itinéraire, l'instant prévu d'arrivée du transporteur à l'étape et l'instant prévu de départ du transporteur de l'étape.
- **Logging** est une table non exploitée contenant un logging des actions effectuées sur la base: pour chaque action on enregistre l'utilisateur, l'instant et un court descriptif de l'action.

4.3 La Console de Gestion des Ordres de Transport

Notre console est une application PHP très simple permettant d'ajouter des ordres de transports, des clients et des colis à la base de données et de les visualiser, ainsi que les itinéraires de la journée, par la suite.

4.3.1 La Structure du Code

Nous proposons une courte description des 14 fichiers PHP constituant cette application.

- **about:** page destinée à contenir un "à propos" pour notre site.
- **addOrders:** permet à l'utilisateur d'entrer un ordre de transport en donnant des informations sur l'expéditeur, le destinataire, la facturation et le colis.
- **footer:** un petit footer apparaissant sur toutes nos pages.
- **head:** le header de nos pages.
- **help:** page d'aide.
- **insertOrder:** ce code s'occupe de l'insertion en base de données des informations entrées sur la page "addOrders".

- **menu:** un menu contenant un lien vers les pages de visualisation des ordres, des clients, des colis, des itinéraires et vers la page d'ajout d'ordre. Ce menu est inclu sur toutes nos pages.
- **seeClient:** page d'affichage des clients courants (dont l'attribut "to_date" est supérieur à maintenant).
- **seeOrders:** page d'affichage des ordres de transport courants.
- **seeParcels:** page d'affichage des colis courants.
- **seeRoutes:** page d'affichage des itinéraires de la journée.
- **seeStatus:** page d'affichage de l'historique d'un colis donné.
- **seeSteps:** page d'affichage des étapes constituant un itinéraire.
- **welcome:** page de bienvenue du site.

4.3.2 Démonstration

L'application consiste en un ensemble d'écrans présentant dans des tables le contenu de la base de données plus un écran d'ajout d'ordres de transport présentant 3 lignes de cases à remplir: une ligne pour les informations sur l'expéditeur, une ligne pour les mêmes informations sur le destinataire et une ligne pour la description du colis.

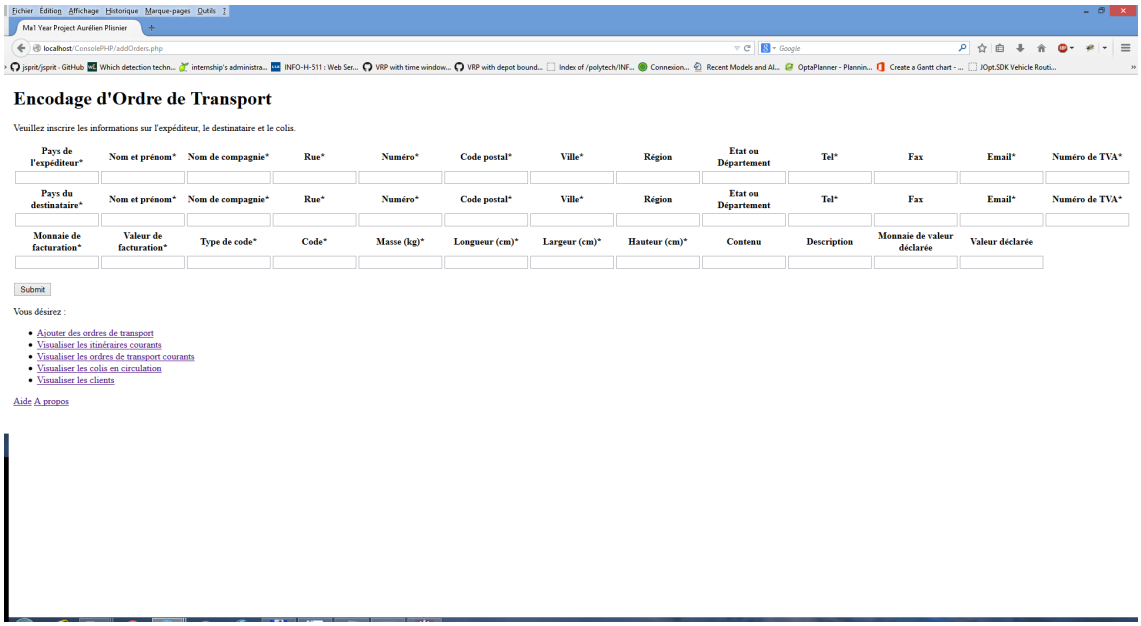

Figure 4.2: Ecran d'ajoût d'ordre de transport

4.4 Le Calculateur d'Itinéraire

Le calculateur d'itinéraire est une application Java tournant au siège de l'entreprise, son rôle est de dériver un plan de route pour la flotte de véhicules de l'entreprise à partir des ordres de transport présents dans la base de données. Le calculateur d'itinéraires utilise la librairie Java Jsprit. [2] Cette librairie permet de résoudre les vehicule routing problems (VRP) dont notre PDP est une sous-classe et les traveling salesman problems (TSP). Elle implémente une métaheuristique basée sur le principe de "Ruin and Recreate" présenté à la section 2.4.

4.4.1 La Structure du Code

Le code est divisé en 4 packages:

- **Main.JspritTest** contient la classe Main.

  Main: Instancie un constructeur de problème et un solveur, commande la construction du problème et passe le résultat au solveur.

- **dbHandler** contient les classes ayant à interagir avec la base de données.

  ProblemBuilder: Construit un PDP à partir des ordres de commandes et des clients stockés en base.

  RouteStorer: Enregistre en base les itinéraires calulés par le solveur.

- **GMapClient** contient les classes (tirées de [11]) permettant d'obtenir auprès de Google Maps les coordonnées des adresses stockées en base.

  Location: Structure de données contenant une localisation.

  MapClient: Classe permettant d'obtenir une adresse exacte et les coordonnées associées auprès de GoogleMaps à partir d'un String.

  XMLHelper: Permet d'interpréter le XML renvoyé par Google Maps.

- **solver** contient le solveur de PDP.

  PDPSolver: Résout un PDP et instancie un RouteStorer pour initier l'enregistrement de la solution.

4.4.2 Démonstration

Les grandeurs utilisées pour cette démonstration sont sans unité. C'est un des sujets d'amélioration discutés à la section 5.3. Testons l'application pour un problème simple: un set de 13 ordres de transport est considéré. Ils concernent 26 clients disséminés à travers Bruxelles. La flotte comporte un ensemble de 5 véhicules partagés en 3 classes:

Classe 1: Véhicules de capacité 120 et de coût d'exploitation par unité de distance 1.

Classe 2: Véhicules de capacité 500 et de coût d'exploitation par unité de distance 3.

Classe 3: Véhicules de capacité 1000 et de coût d'exploitation par unité de distance 7.

Tous les transporteurs commencent leur journée de travail à l'instant 0 et la terminent à l'instant 500. Un itinéraire ne peut donc pas dépasser 500 unités de temps.
Terminons notre description du problème par la liste des ordres pris en charge:

Numéro de livraison	Coordonnées de l'expéditeur	Coordonnées du destinataire	Masse
0	(50.8422767, 4.2994471)	(50.8394124, 4.3129309)	2
1	(50.8366058, 4.3824622)	(50.8071547, 4.3330522)	2
2	(50.8354359, 4.3000387)	(50.8568656, 4.3047678)	13
3	(50.872516, 4.3125711)	(50.8019742, 4.4201214)	31
4	(50.8313568, 4.3548698)	(50.8750456, 4.386963)	6
5	(50.8579456, 4.3112202)	(50.8693237, 4.3897301)	126
6	(50.8434549, 4.3603023)	(50.8828427, 4.3166348)	2
7	(50.859699, 4.4143559)	(50.8495051, 4.3374508)	31
8	(50.8092927, 4.3686351)	(50.8574003, 4.3585653)	13
9	(50.8186665, 4.3394036)	(50.8380831, 4.3011285)	124
10	50.8471839, 4.3628479)	(50.8187511, 4.3401948)	24
11	(50.8118356, 4.3848377)	(50.8762402, 4.3990322)	55
12	(50.85, 4.35)	(50.8557782, 4.3064613)	3

L'application effectue l'extraction des données concernant les ordres de transport, les clients et les colis associés dans la base de données. La traduction de ces données en un problème tel que spécifié au dessus est ensuite faite. Les informations concernant la flotte sont hardcodées dans l'application.

Une fois le problème construit, une solution est calculée. Jsprit propose un affichage en console des résultats sous forme de trois tables, nous les reprenons ci-dessous.

La première table décrit le problème reçu par l'application: les 13 ordres sont traduits en 13 jobs, un "Service" désigne une livraison de l'entrepôt vers un client, tandis qu'un "Shipment" désigne une livraison d'un expéditeur vers un destinataire. Le nombre de véhicules à disposition est fini.

problem	
indicator	value
nJobs	13
nServices	0
nShipments	13
fleetsize	FINITE

La deuxième table reprend un résumer de la solution: le coût total de routage est la somme des distances parcourues par chaque véhicule multipliée par leur coût d'utilisation par unité de distance. Le nombre de véhicules nécessaires pour résoudre le problème est aussi cité.

solution	
indicator	value
costs	648.1994888715647
nVehicles	2

La dernière table décrit les itinéraires calculés: il y a ici deux itinéraires, et à chaque itinéraire est associé un véhicule, les deux véhicules appartiennent à la classe 1. La colonne "activity" décrit le type d'activité: départ de l'entrepôt, retour à l'entrepôt, prise en charge de colis ou livraison de colis. La colonne "job" donne le numéro d'ordre de transport pris en charge. "arrTime" donne l'instant d'arrivée à la destination, "endTime" l'instant de départ et "cost" décrit les coûts.

detailed solution

route	vehicle	activity	job	arrTime	endTime	costs
1	vehicle1_2	start	–	undef	0	0
1	vehicle1_2	pickupShipment	0	0	1	0
1	vehicle1_2	deliverShipment	0	15	16	14
1	vehicle1_2	pickupShipment	10	66	67	64
1	vehicle1_2	pickupShipment	1	90	91	87
1	vehicle1_2	deliverShipment	1	148	149	144
1	vehicle1_2	deliverShipment	10	163	164	158
1	vehicle1_2	pickupShipment	9	165	166	159
1	vehicle1_2	deliverShipment	9	208	209	201

1	vehicle1_2	end	–	214	undef	206
2	vehicle1_1	start	–	undef	0	0
2	vehicle1_1	pickupShipment	2	7	8	7
2	vehicle1_1	deliverShipment	2	30	31	29
2	vehicle1_1	pickupShipment	5	37	38	35
2	vehicle1_1	pickupShipment	3	53	54	50
2	vehicle1_1	pickupShipment	4	113	114	109
2	vehicle1_1	pickupShipment	8	140	141	135
2	vehicle1_1	pickupShipment	11	157	158	151
2	vehicle1_1	deliverShipment	3	195	196	188
2	vehicle1_1	pickupShipment	7	254	255	246
2	vehicle1_1	deliverShipment	11	278	279	269
2	vehicle1_1	deliverShipment	5	290	291	280
2	vehicle1_1	deliverShipment	4	298	299	287
2	vehicle1_1	deliverShipment	8	332	333	320
2	vehicle1_1	pickupShipment	6	347	348	334
2	vehicle1_1	pickupShipment	12	360	361	346
2	vehicle1_1	deliverShipment	7	374	375	359
2	vehicle1_1	deliverShipment	6	414	415	398
2	vehicle1_1	deliverShipment	12	444	445	427
2	vehicle1_1	end	–	460	undef	442

4.5 L'application Java Android

L'application doit permettre aux transporteurs de:

- Scanner lors de la prise en charge et de la livraison des colis préalablement sauvegardés en base de données par le siège.
- Afficher à l'écran du smartphone un résumé des informations concernant l'état du colis, stockées dans la base de données du siège.
- Permettre la mise à jour automatique et à distance du statut du colis dans la base de données du siège selon une succession de statuts prédéfinie.

Voici ci-dessous un schéma présentant les différents statuts que peut avoir un colis sous forme de rectangles bleus. Les transitions dues à l'utilisation de l'application smartphone par un transporteur sont représentées par des flèches vertes. Les transitions dues à l'action des employés du siège (non prises en charge par l'application Android) sont repréentées par des flèches rouges.
L'état "en entrepôt" ne sera pas utilisé dans le cadre de ce projet (voir hypothèses section 2.2).

4.5.1 La Structure du Code

Le code est divisé en 5 packages:

- **activities** contient les activités:

  MainActivity: Classe principale du projet: affiche un écran de login et redirige vers l'écran de Scan.

  ScanActivity: Ecran de scan: offre l'option de scanner une puce RFID [4], invoque Zxing pour scanner un code-barres[1], affiche le statut du colis scanné et permet à l'utilisateur de mettre à jour ce statut.

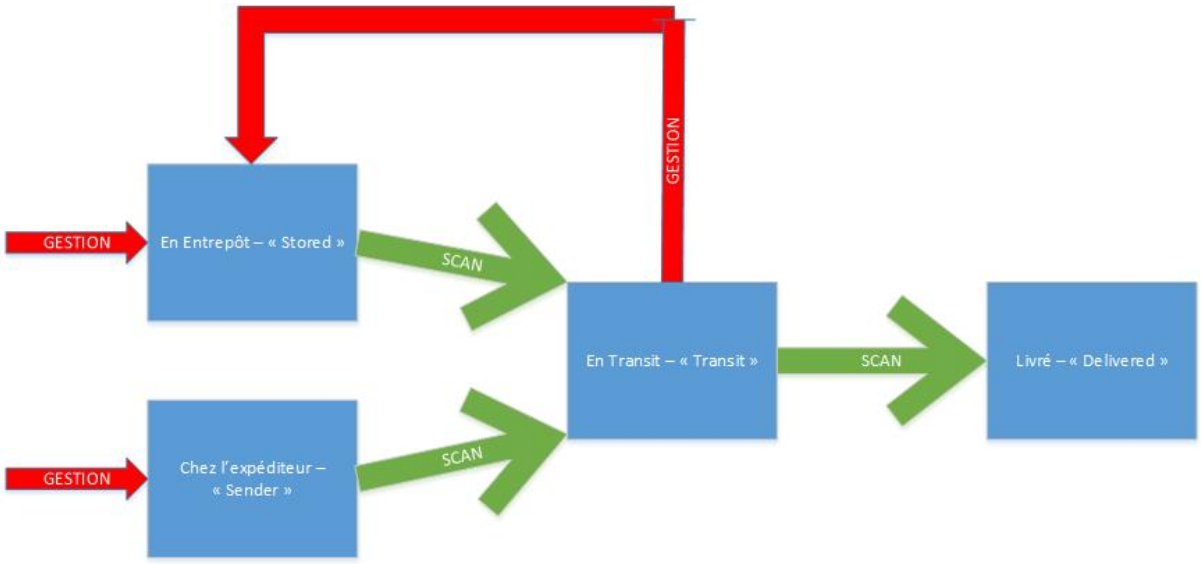

Figure 4.3: Les statuts du colis et leurs transitions

- **com.google.zxing.integration.android** contient le code permettant d'intégrer l'application de scan de codes-barres Zxing:

  IntentIntegrator: Une classe utilitaire permettant l'intégration avec le scanner de codes-barres Zxing via des Intent's. Classe tirée du repository Github de Zxing [3].

  IntentResult: Classe encapsulant le résultat d'une lecture de code-barres invoquée via IntentIntegrator. Classe tirée du repository Github de Zxing [3].

- **nfc** contient du code nécessaire à l'utilisation de NFC pour scanner une puce RFID:

  NFCForegroundUtil: Classe gérant le parsing NFC tirée de [4].

- **rest** contient le code permettant de communiquer avec le serveur via REST:

  RestClient: Reçoit les paramètres de la requête REST, construit l'URL correspondante, exécute la requête et reçoit les résultats. Code tiré en grande partie de [5].

- **geolocation** Contient la classe:

  CurrentLocation: Permet d'obtenir les coordonnées du smartphone.

4.5.2 Démonstration

L'écran de lancement de l'application est un écran de "Login" standard: l'utilisateur doit donner son matricule et son mot de passe. Une fois le bouton "Submit" enfoncé, le matricule et le mot de passe hashé sont envoyés au serveur distant pour comparaison avec le contenu de la base de données. Le serveur renvoie un booléen à l'application Android.

Si le Login est réussi, l'écran de scan apparaît. Un bouton "Scan Barcode" permet de faire appel à l'application Zxing à tout moment. Si un tag lisible est approché du smartphone, la procédure de scan est lancée, elle, automatiquement.

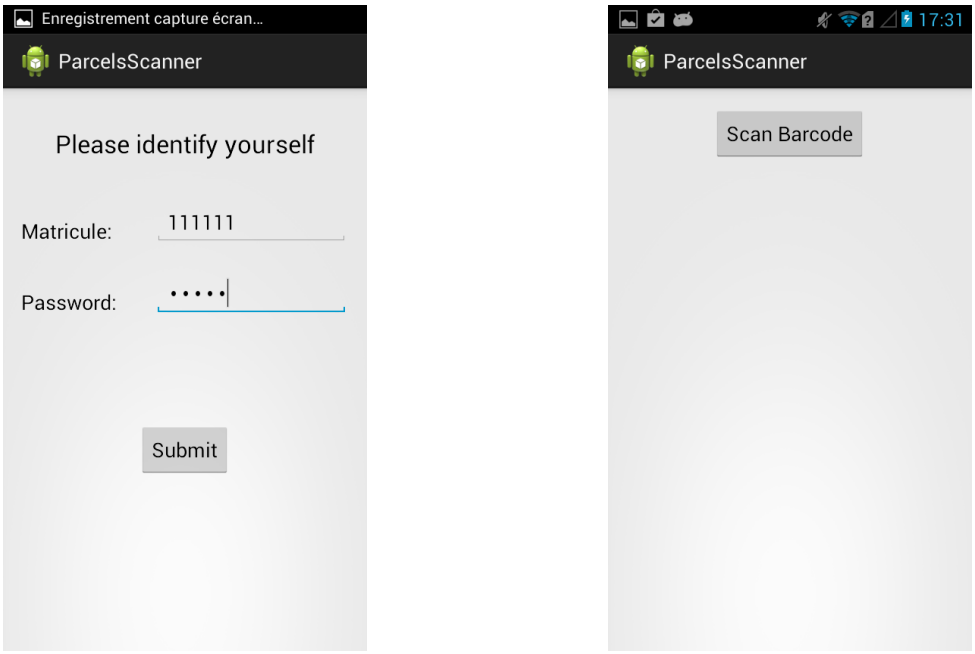

Figure 4.4: L'écran de Login et l'écran de Scan

Si le bouton "Scan Barcode" est enfoncé, l'application passe la main à Zxing. Une fois le scan terminé, on retourne à l'application principale et le contenu du code-barres est envoyé via HTTP au serveur distant de manière à obtenir les informations du produit.

Les caractéritiques du produit apparaissent à l'écran: le nom du produit, une description et le statut actuel du produit.

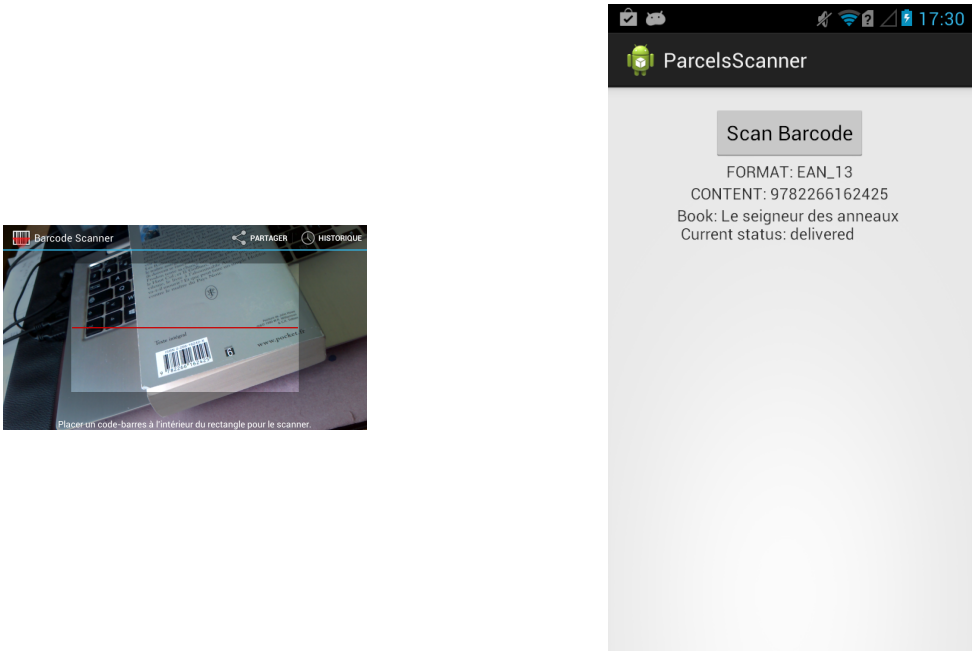

Figure 4.5: Appel à Zxing et affichage des caractéristiques du produit après scan de code-barres

Un résultat similaire apparaît si un tag lisible est approché du smartphone.Il faut remarquer la

présence d'un bouton "Update Status". Cela vient du fait que le statut du colis est différent de "delivered". Il peut donc encore évoluer. Une pression de ce bouton enclanche une demande de mise à jour du statut du colis auprès du serveur distant. Le bouton disparaît après pression: une seule mise à jour de statut est permise lors de chaque action de scan. Une fois la mise à jour effectuée, le nouveau statut et un bref message de succès sont affichés.

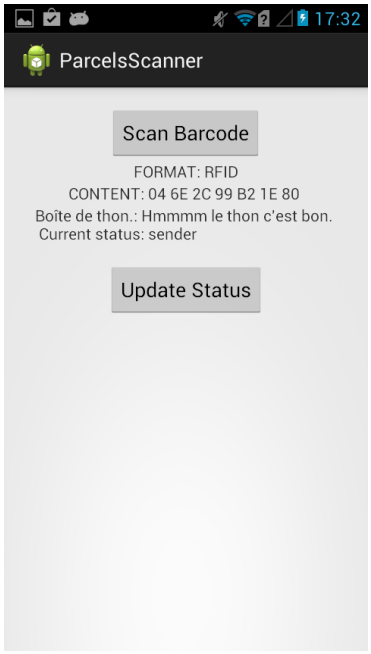

Figure 4.6: Affichage des caractéristiques du produit après scan de tag RFID

4.6 Le Serveur HTTP

Le serveur est chargé de recevoir les requêtes des scanners transmises à distance via Http et de les traduire en commandes sur la base de données de l'entreprise. Le serveur est programmé en Java.

4.6.1 La Structure du Code

Le code est divisé en 3 packages:

- **webService** contient les classes chargées de recevoir les requêtes REST.

  LoginHandler: Reçoit un couple matricule / mot de passe d'utilisateur lors d'une requête de login. Renvoie 1 ou 0.

  ScanHandler: Reçoit un couple format (type de code-barres ou RFID) et contenu du code scanné et renvoie un texte contenant des infos sur l'objet scanné s'il se trouve dans la base de données, un message d'erreur autrement.

  StatusUpdateHandler: Reçoit un couple format / contenu du scan ainsi que latitude / longitude et met à jour l'état de l'objet dans la base de données.

- **model** contient le code de gestion de la base de données MySQL via Jdbc.

  DbHandler: Gère la connexion et les requêtes à la base de données.

- **email** contient la classe nécéssaire à l'envoi d'emails aux clients.

EmailSender: Envoie un mail à l'expéditeur et au destinataire d'un colis. Ce mail contient le nom du colis, une description du colis, le statut précédent du colis et le nouveau statut du colis. L'email est envoyé depuis l'adresse Gmail de l'entreprise.

4.6.2 Démonstration

La démonstration du serveur est presque entièrement comprise dans la démonstration de l'application Android, puisque le rôle principale du serveur est de répondre aux solicitations de cette dernière.
La seule fonctionalité du serveur qui n'a pas été présentée est l'envoi automatique de mail. Un mail est envoyé à l'expéditeur et au destinataire d'un colis lors de chaque mise à jour du statut de ce colis.
Le contenu d'un mail est:
"Dear Client,
The status of your parcel (Name: Hansaplast; Description: Paquet de sparadraps.) has just changed from stored to transit.
Best regards,
The PotM Team."

Chapter 5

Conclusion

En guise de conclusion nous proposons de tirer la liste des enseignements de ce projet, nous discutons ensuite les problèmes et difficultés principales qu'il a fallu surmonter lors de sa réalisation. Nous effectuons finalement une critique de notre travail et nous proposons des pistes d'amélioration pour notre système.

5.1 Enseignements

Lors de ce projet nous nous sommes familiarisés avec la programmation Android: comment structurer une application Android, comment faire appel à des applications externes et exploiter leurs résultats, comment utiliser les modules du smartphone.

Nous avons appris à établir une communication HTTP rudimentaire entre un client Android et un serveur Java.

Nous avons mis en place et exploité une base de données temporelle.

Nous avons revu la notion de services web et nous en avons utilisés dans nos applications.

Nous avons revu les notions d'heuristique et de métaheuristique. Nous avons étudié divers vehicule routing problems, en particulier le paired pickup and delivery problem. L'étude de la métaheuristique "ruin and recreate" nous a permis de comprendre le fonctionnement général d'une heuristique et la différence d'applicabilité qui existe entre les heuristiques locales et à "sauts de géant".

Nous nous sommes familiarisés avec diverses technologies de scan, en particulier RFID, NFC et nous nous sommes intéressés au fonctionnement des codes-barres.

5.2 Problèmes et Difficultés

La difficulté principale rencontrée lors de ce projet est aussi ce qui l'a rendu si intéressant à mener: l'exploitation de technologies qui nous étaient inconnues jusqu'alors. Au moment de commencer le projet:

- Nous n'avions jamais touché un smartphone Android, et encore moins programmé pour Android.
- Nous n'avions que peu de connaissances en terme d'heuristiques et de métaheuristique.
- Nous ne savions pas ce que NFC et RFID veulent dire.

Nous avons nous-mêmes proposé le sujet de ce projet. L'autre grande difficulté a donc été de définir et délimiter le projet, de manière à ce qu'il soit intéressant à accomplir et représente une masse de travail

digne d'un projet d'année. Beaucoup de temps a été perdu pour cela et ce n'est qu'assez tard dans l'année que la portée définitive du projet a été définie de manière complètement univoque.

5.3 Critique et Pistes d'Améliorations

L'étendue de ce projet étant assez large, certains aspects ont dû être négligés. Nous listons ici les points devant être améliorés selon nous pour faire passer notre système de l'état de preuve de concept à celui d'application utilisable.

- La communication HTTP entre l'application Android et le serveur Java est très rudimentaire et non sécurisée.
- Les interfaces utilisateur sont, elles-aussi, plutôt rudimentaires.
- Une photo et un commentaire écrit devraient pouvoir être ajoutés lors d'un scan, de manière à enregistrer des informations sur l'état du colis lors de sa course.
- Le cas du défaut de client devrait être traité.
- Le simple cycle "expéditeur - transit - destinataire" ne suffit pas à décrire dans la réalité l'état d'un colis.
- Le transporteur devrait pouvoir visualiser son itinéraire théorique et sa position sur une carte depuis son smartphone.
- Un calcul de différences entre les itinéraires calculés et les itinéraires réels devrait être inclus. (En l'état nous enregistrons déjà toutes les informations permettant d'effectuer ce calcul.)
- L'application Android devrait permettre de recueillir la signature d'un client lors d'une prise en charge ou d'une livraison de colis.
- La console PHP devrait être étendue de manière à permettre une gestion complète des ordres de transport et de leur facturation, une gestion de la flotte de véhicules et des transporteurs, etc...
- Un site web attractif permettant aux clients de passer commande, de payer et de suivre l'état de leur commande en temps réel devrait aussi être ajouté au système.
- La calculateur d'itinéraires n'est pas encore utilisable en l'état: il faut ajouter des unités de manière à obtenir des grandeurs réelles pour les coûts de routages et durées d'itinéraire. Pour cela il faudrait effectuer des tests en situation d'utilisation réelle.

Bibliography

[1] Sue Smith *http://code.tutsplus.com/tutorials/android-sdk-create-a-barcode-reader--mobile-17162* Tutoriel Zxing

[2] Sue Smith *https://github.com/jsprit* Jsprit

[3] Repo Git Zxing *https://github.com/zxing/zxing* Repository Git de Zxing

[4] Android Worksop *http://androidworkshop.tumblr.com/* Tutoriel Android NFC

[5] Luke Lowrey *http://lukencode.com/2010/04/27/calling-web-services-in-android-using-httpclient/* Exemple de client Rest sur Java Android

[6] D. Pisinger, S. Ropke *A General Heuristic for Vehicule Routing Problems* Computers & Operations Research 34, 2007: pp2403-2435.

[7] Gerhard Schrimpf, Johannes Schneider, Hermann Stamm-Wilbrandt, Gunter Dueck *Record Breaking Optimization Results Using the Ruin and Recreate Principle* December 24, 1998: IBM Scientific Center Heidelberg

[8] Sophie N. Parragh, Karl F. Doerner, Richard F. Hartl *A survey on pickup and delivery problems. Part2: Transportation between pickup and delivery locations* April 16, 2008: Institut für Betriebswirtschaftslehre, Universität Wien Brünnerstr. Austria.

[9] Lars Vogel *http://www.vogella.com/tutorials/REST/article.html* 2012.

[10] *https://jersey.java.net/* Site web de Jersey.

[11] *http://cs.ulb.ac.be/public/teaching/infoh511* Page du cours INFO-H-511 : Web Services.

[12] Luc Stevens *Survey of possible uses of RFID in money handling* Nationale Bank van België

[13] *http://www.radio-electronics.com/info/wireless/nfc/near-field-communications-tags-types.php* Radio-Electronics.com : NFC Tags and Tag Types.

[14] *http://electronics.howstuffworks.com/nfc-tag1.htm* howstuffworks: What's an NFC tag.

[15] *http://electronics.howstuffworks.com/near-field-communication.htm* howstuffworks: How NFC works.

[16] Judith Vanderkay*http://nfc-forum.org/newsroom/nokia-philips-and-sony-establish-the-near-field-c* Nokia, Philips And Sony Establish The Near Field Communication (NFC) Forum, March 18 2004. NFC Forum.

[17] *http://en.wikipedia.org/wiki/Barcode*