**Université de Strasbourg**
UFR Maths-Informatique

Licence 3 - Semestre 6

# Le voyageur de commerce

Jonathan HAEHNEL & Marc PAPILLON

Strasbourg, le 3 mai 2012

# Table des matières

# Introduction

Nous allons débuter l'écriture de ce rapport par une courte introduction. Petite présentation de l'équipe : *Jonathan Haehnel et Marc Papillon*, élèves de licence informatique actuellement au 6ème semestre.

L'objectif du projet est le **développement d'une application orientée objet distribuée permettant de résoudre le problème du voyageur de commerce**. Pour rappel, le problème du voyageur de commerce consiste, étant donné un ensemble de villes séparées par des distances données, à trouver le plus court chemin qui relie toutes les villes. Le problème est plus complexe qu'il n'y parait, en effet, nous avons du effectuer une longue analyse, afin de savoir quel methode nous allons utilisée.

Nous avons choisit de réaliser notre application distribuée en Java avec la bibliothèque RMI [1]. Par ailleurs, nous avons sélectionné l'**algorithme des colonies de fourmis** pour la résolution du problème du voyageur de commerce, celui-ci nous a paru le plus facile à implémenter *(cf. section 2.2)*.

Afin de rendre l'application plus intéractive avec l'utilisateur et, afin de mieux visualiser le résultat, nous avons mis en place une interface graphique Swing. *(cf section 2.4)*

1. Remote method invocation : technologie qui permet la communication via le protocole HTTP entre des objets Java éloignés physiquement les uns des autres

# Chapitre 1

# Etat des lieux

## 1.1 Fonctionnalités

De base, notre application n'avait pas de fonctionnalités bien dégagées, car c'est un sujet très ouvert. A l'aide de notre interface graphique, nous avons pu en dégagées certaines :

| **Description de la tâche** | **Type** |
|---|---|
| Transformation du problème en structures de données plus concrétes (carte, villes, chemins) | Core |
| Résolution du problème à l'aide de l'algorithme des colonies de fourmis | Core |
| Mise en place de la distributivité (plusieurs clients travaillent ensembles) | Core |
| Syncronisation avec le serveur | Core |
| Création d'une carte et placement des villes | Gui |
| Visualisation de l'évolution du plus court chemin au courant de l'execution | Gui |
| Affichage des détails d'une ville | Gui |

TABLE 1.1 – Liste des fonctionnalités

## 1.2 Répartition des taches

En debut de projet, nous nous sommes répartis les tâches équitablement et selon les envies de chacun.

**Taches de Marc :**
- Analyse du sujet
- Création des structure de bases
- Implémentation de l'algorithme des colonies de fourmis
- Rédaction du rapport

**Tache de Jonathan :**
- Analyse du sujet
- Création des structure de bases
- Réalisation de l'interface graphique de l'application
- Rédaction du rapport

# Chapitre 2

# Mise en place

## 2.1 Structure City-Map et Way

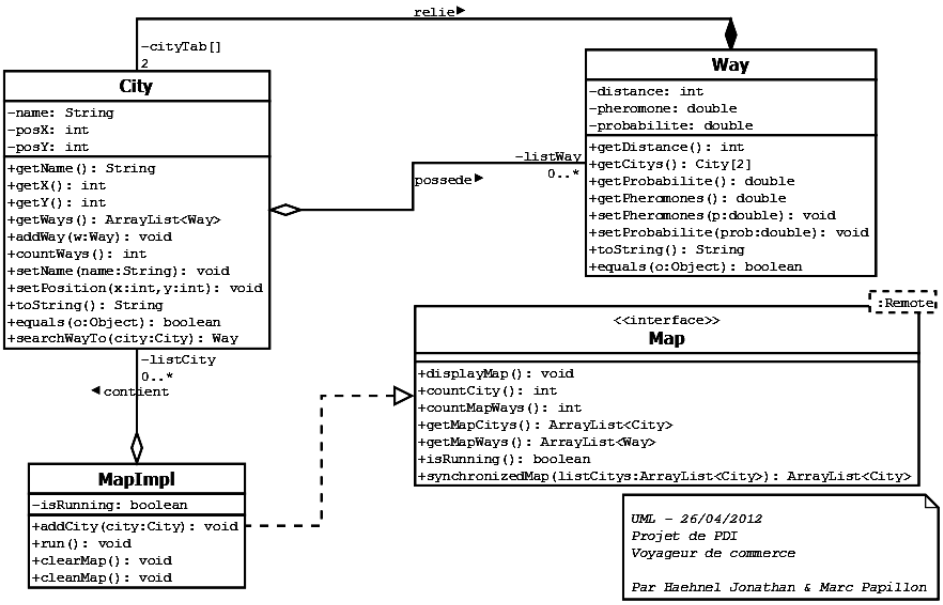


FIGURE 2.1 – UML des classes

Comme nous utilisons RMI, nous avons utilisé une interface Map. Ceci permettra aux clients d'accéder à la **classe distante implémentant Map** (MapImpl). Cette classe permet d'accéder à la map du serveur, à compter le nombre de ville, le nombre de chemin, etc.

MapImpl contient en attribut une liste de ville, et une ville contient une liste de chemin. La liste de chemin est simplement tous les chemins menant à une autre ville. Enfin nous avons une classe Way qui représente un chemin contenant sa longueur, le taux de phéromones s'y trouvant et les deux villes à l'extrémité du chemin.

## 2.2 Algorithme des colonies de fourmis

Nous avons choisi d'implémenter l'algorithme de la colonie de fourmi, un algorithme connu et réputé pour ses performances dans le problème du voyageur de commerce.

Ici, une machine est considérée comme une fourmi. Sa mise en place nécessite qu'une fourmi parcourt un cycle puis d'y laisser des phéromones quand elle refait le chemin en sens inverse. Plus le chemin pris sera court, plus le taux de phéromones déposé sera grand. Le choix de la destination est pondéré par le taux de phéromone se trouvant sur le chemin pour y aller.

Dans notre version de l'algorithme, la "visibilité" d'une ville est aussi prise en compte, plus la ville est loin de la fourmi au moment de son choix, moins le chemin pour y aller a de chance d'être pris. Ainsi les fourmis s'orientent naturellement vers les chemins les plus proches d'elles.

A chaque fin de cycle, les phéromones se trouvant sur les chemins s'évaporent pour ne pas que les fourmis se rapprochent trop vite d'une solution qui pourrait ne pas être exacte.

## 2.3 Système répartis

Plusieurs options nous permettaient de faire un système répartie, celle que nous avons choisi est la plus simple et la plus rapide d'execution. Il suffit d'intéragir directement sur la Map distante du serveur pour que tous les clients aient les même informations à un moment t.

Nous avons aussi mis en place un système de synchronisation avec la map du serveur au client, qui se met fait non pas à chaque itération de l'algorithme de la fourmi mais par exemple toutes les 20 itérations afin de ne pas surcharger le serveur.

## 2.4 Graphical User Interface

**Notre interface graphique a plusieurs objectifs :**

- Interagir avec l'utilisateur
- Permettre à l'utilisateur d'ajouter facilement des villes ou relancer rapidement l'algorithme plusieurs fois.
- Visualiser en direct l'évolution du plus court chemin.
- Savoir en temps réel, combien il y a de phéromones sur chaque chemin (détails des villes)

L'affichage de l'interface graphique s'effectue automatiquement après le lancement du programme serveur. Cette dernière possède deux grands modes de fonctionnement : le mode « dessin »et « execution »que nous allons détaillés ci-dessous.

### 2.4.1 Le mode « Dessin »

On accède à ce mode lors du démarrage de l'application ou en cliquant sur les bouton « Dessiner »et « Réinitialiser ». Ce mode permet à l'utilisateur de dessiner la carte et d'y rajouter par un simple clic une nouvelle ville. Lorsque l'on se trouve dans ce mode, les clients ne peuvent pas utiliser la Map distante.

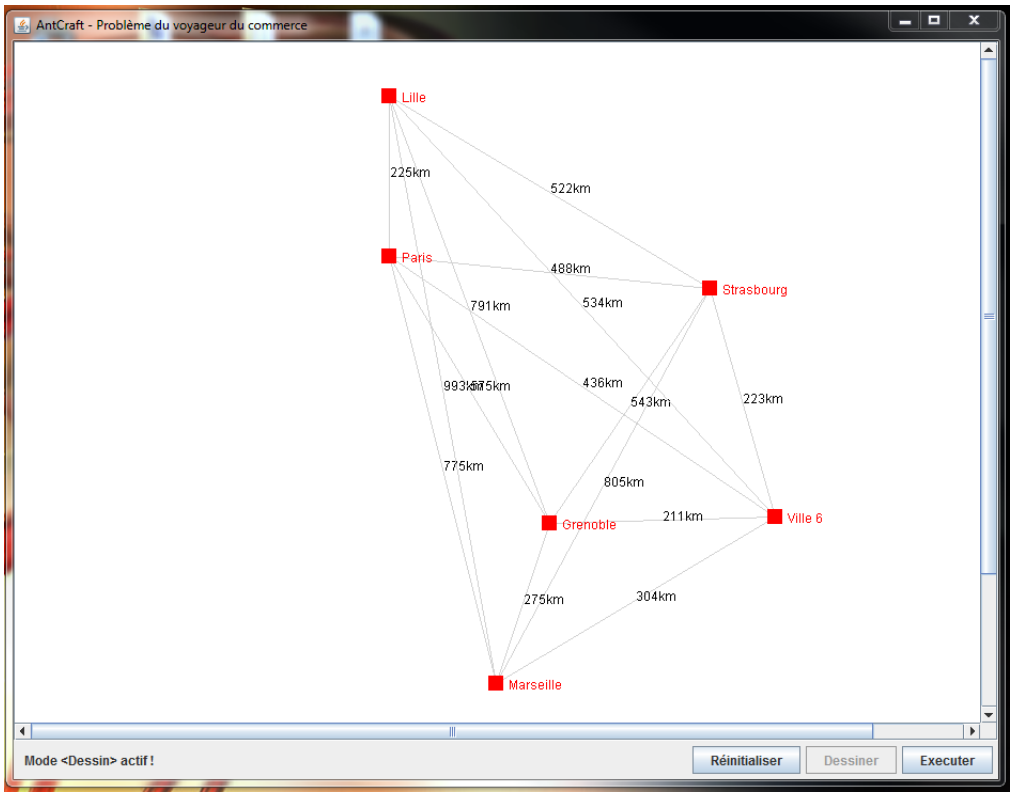


FIGURE 2.2 – Interface en mode dessin

### 2.4.2 Le mode « Execution »

Comme son nom l'indique, ce mode permet l'éxecution de la Map et le calcul du plus court chemin. Il se déclanche en cliquant sur le bouton « Executer »qui lance le compteur et le thread de rafraichissement de la map (delai par défaut de 500ms). Les clients peuvent donc utiliser l'objet Map distant. Une fois un plus court chemin déterminé, le thread de rafraichissement s'arrête et les clients sont déconnectés.

De plus, en cliquant sur une ville, une petite boite de dialogue apparait qui contient les informations de la ville sélectionnée (destination, distance et nb de phéromones). Celle-ci est aussi rafraichie toutes les 500ms.

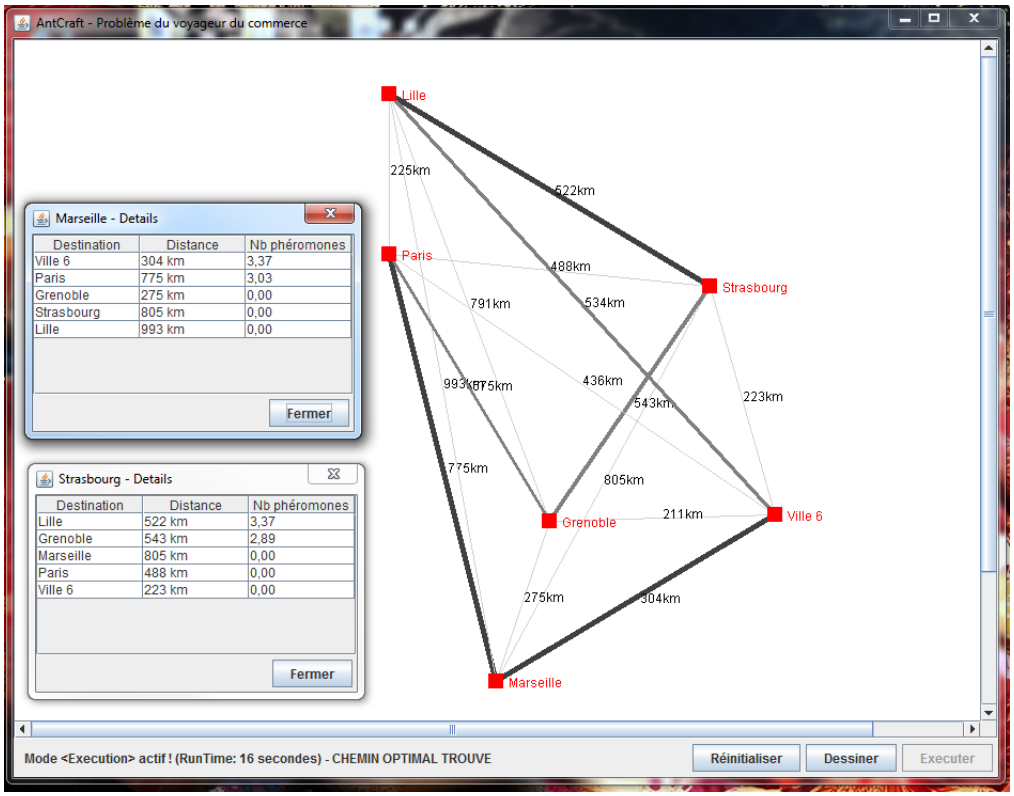


FIGURE 2.3 – Interface en mode execution

# Conclusion

La mise en place d'une **telle structure n'a pas été facile, surtout dans le choix des paramètres de réglages** de l'algorithme (taux d'évaporation, importance de la visibilité par rapport aux phéromones...). Cela a nécessité de nombreux tests et la solution n'est pas *encore assez rapprochée de la solution optimale* pour permettre une réelle application de l'algorithme dans un cas concret.