# RAPPORT DE PROJET

**LO43 : BASES FONDAMENTALES DE LA PROGRAMMATION OBJET**

**Printemps 2007**

## Projet Java : Application de recherche du plus court chemin sur une carte de la région de Belfort

**DELON Nicolas**

**WAJS Thomas**

# SOMMAIRE

# I) CAHIER DES CHARGES :

1) Acteurs :

Ce projet est étudié par WAJS Thomas et DELON Nicolas, étudiants à l'Université Technologique de Belfort Montbéliard (UTBM), respectivement en 2nd et 1er semestres de formation au diplôme d'ingénieur en informatique.

2) Cadre :

Ce projet est mis en place dans le cadre de l'Unité de Valeur « LO43 », enseignée à l'UTBM, et s'intitulant « Bases fondamentales de la programmation objet ».
Il est effectué durant le semestre de Printemps 2007, sur le site de Belfort, et est encadré par Franck Gechter et Jean-Charles Créput. Notre enseignant de TP et directeur/évaluateur de projet est David Meignan.

3) Objectifs :

L'objectif de ce projet est la conception et la réalisation d'un prototype d'application calculant le plus court chemin entre 2 points géographiques, à l'échelle d'une région.

L'application à développer devra donc permettre de :

- Visualiser la carte et le réseau routier en renvoyant à l'utilisateur les informations géographiques appropriées (échelle/précision, information sur la zone, coordonnées des points sélectionnés, nom de rues, etc.).
- Zoomer avant/arrière, taille réelle, vue globale.
- Editer et visualiser le système d'unités et la précision. On choisira l'unité Km avec une précision au mètre près, et non pas l'unité « pixel » initiale.
- Sauvegarder et lire le fichier de rues et routes dans ce nouveau système d'unités.
- Calculer le plus court chemin entre deux points saisis et renvoyer les informations appropriées.

Les éléments fournis pour l'élaboration de cette application sont :

- Un algorithme de calcul de plus court chemin (Dijkstra) écrit en Java ainsi qu'un exemple d'utilisation de celui-ci.
- Une carte de la région autour de Belfort (region_belfort_routes_fleuves_habitats.gif) dont l'étendue est donnée par les 2 points de coordonnées géodésiques Lambert II (897990, 2324046) et (971518, 2272510). Cette carte défini une image de fond de taille 9807 ´ 6867 qui correspond à une surface d'environ 73 km ´ 51 km, la précision est de 7.5 m par pixel. L'étendue exacte en m est donnée par les deux points Lambert II.
- Les données du réseau routier correspondant à cette région (region_belfort_streets.xml), se présentant sous forme d'un graphe sur lequel doit être appliqué l'algorithme de plus court chemin. Les coordonnées des points dans ce fichier sont données dans l'unité « pixel » de la carte de la région.

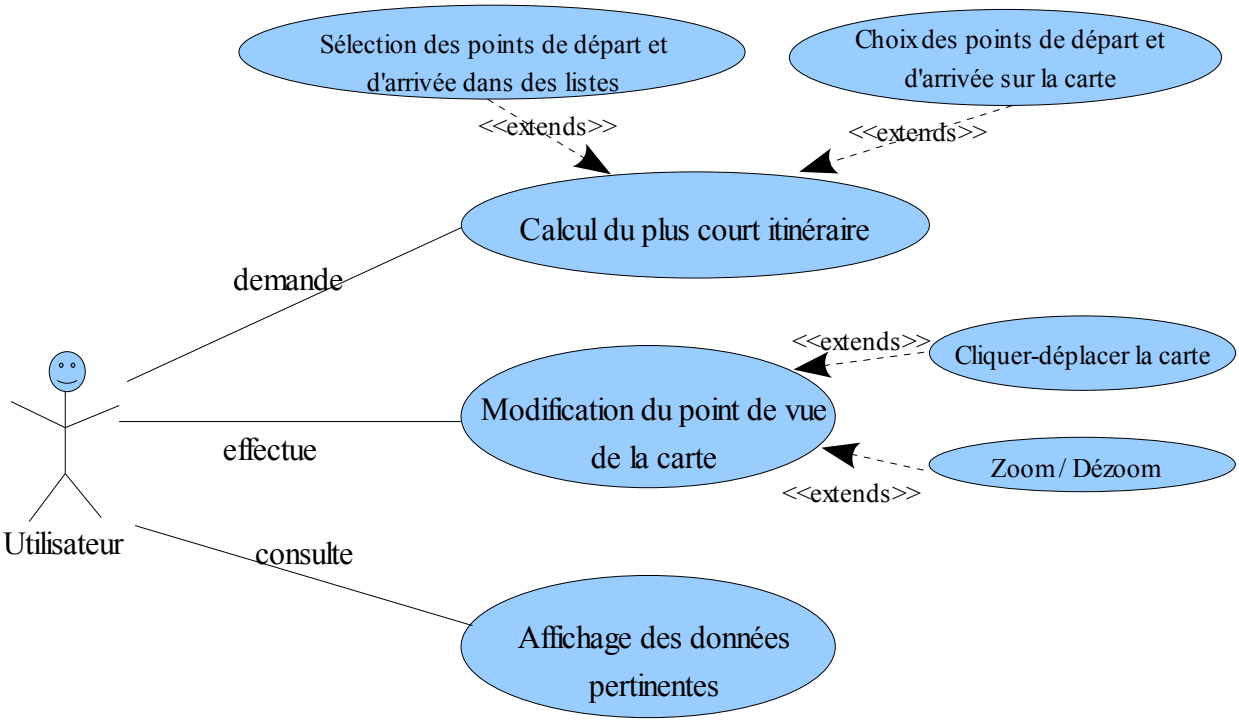
# II) DOSSIER DE SPECIFICATIONS :

## 1) Étude des fonctionnalités :

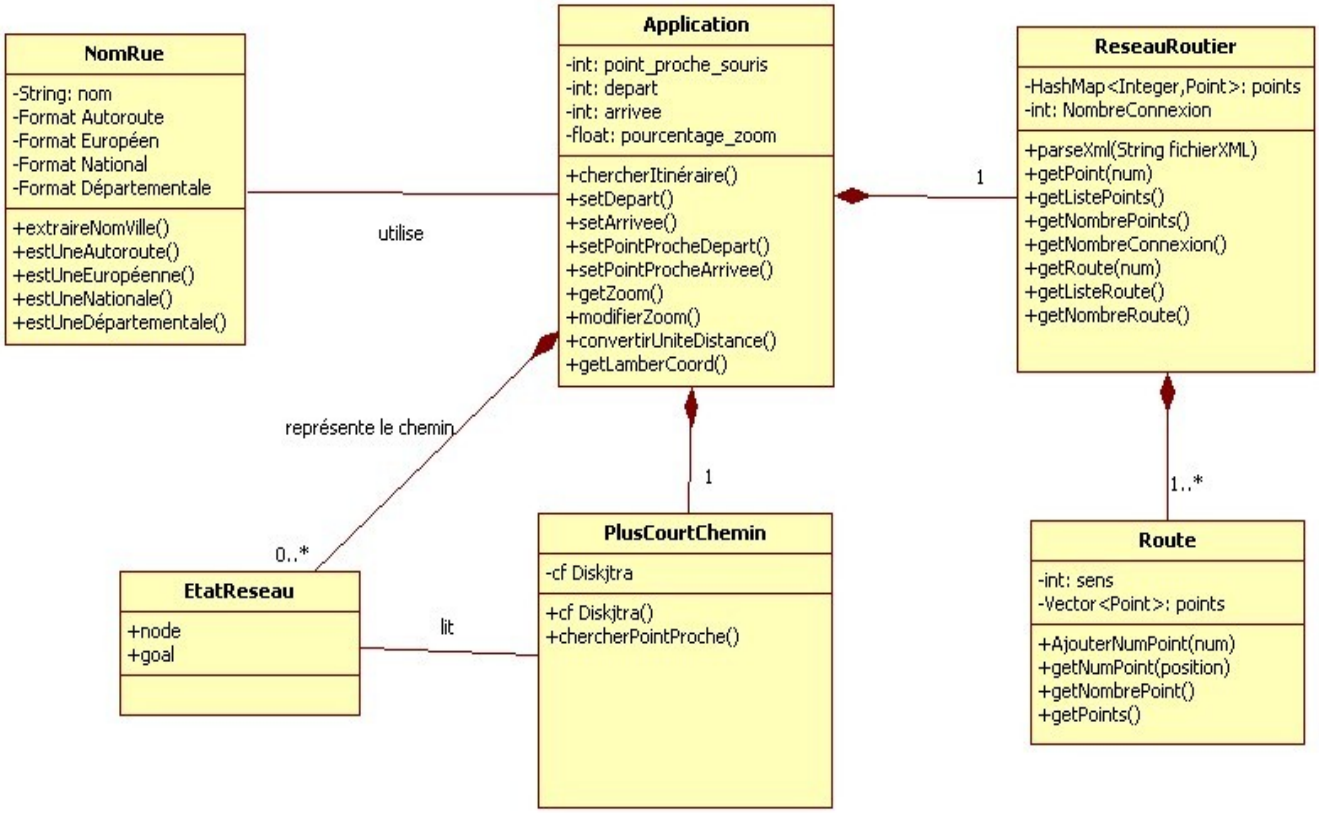
Les trois principales fonctionnalités à mettre en oeuvre dans cette application furent :

- La possibilité de calculer un plus court chemin (itinéraire) entre 2 points de la région.
- Une visualisation ergonomique de la carte : pouvoir modifier le niveau de zoom, et se déplacer dans celle-ci facilement.
- Un affichage des données pertinentes du réseau routier et de l'itinéraire demandé.

Le diagramme de cas d'utilisation suivant présente ces différentes fonctionnalités.



## 2) Etude des données statiques :




Ce modèle ne représente que la partie « modèle », c'est à dire les opérations concernant la gestion de l'itinéraire.

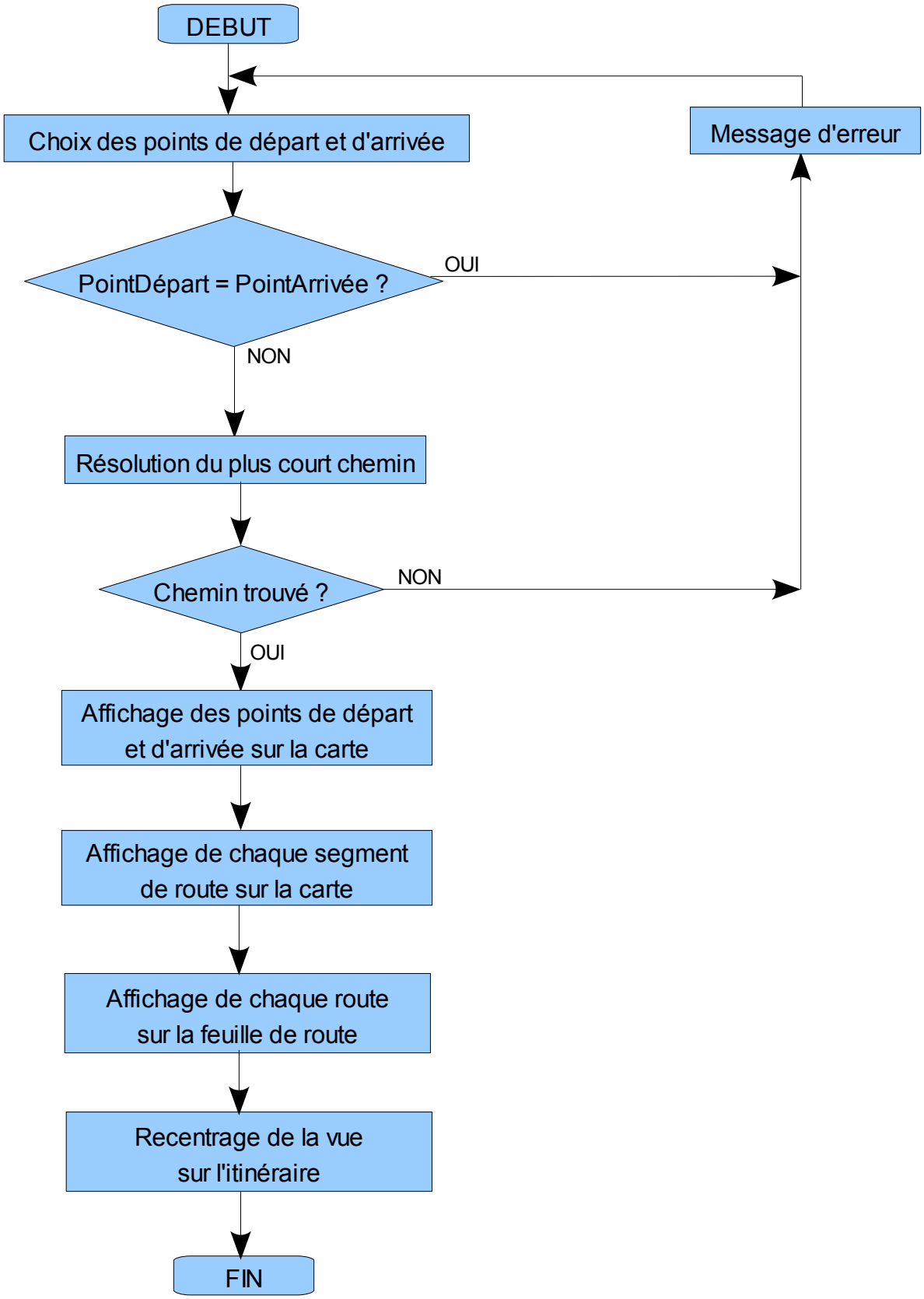
La classe Application regroupe en plus beaucoup d'opération de gestion de la vue ( zoomer, déplacer, recentrer etc... )., et d'actualisation des informations envoyées à l'utilisateur dans l'IHM.

Elle s'occupe aussi de la liaison entre la déclaration des écouteurs de la partie « controller » et leur ajout sur les différents composants de la partie « view » (boutons, ComboBox, etc...).

Enfin elle concentre la plupart des constantes de l'application telle que les valeurs de zoom minimum et maximum, les noms des types de routes (autoroutes, européennes ...)

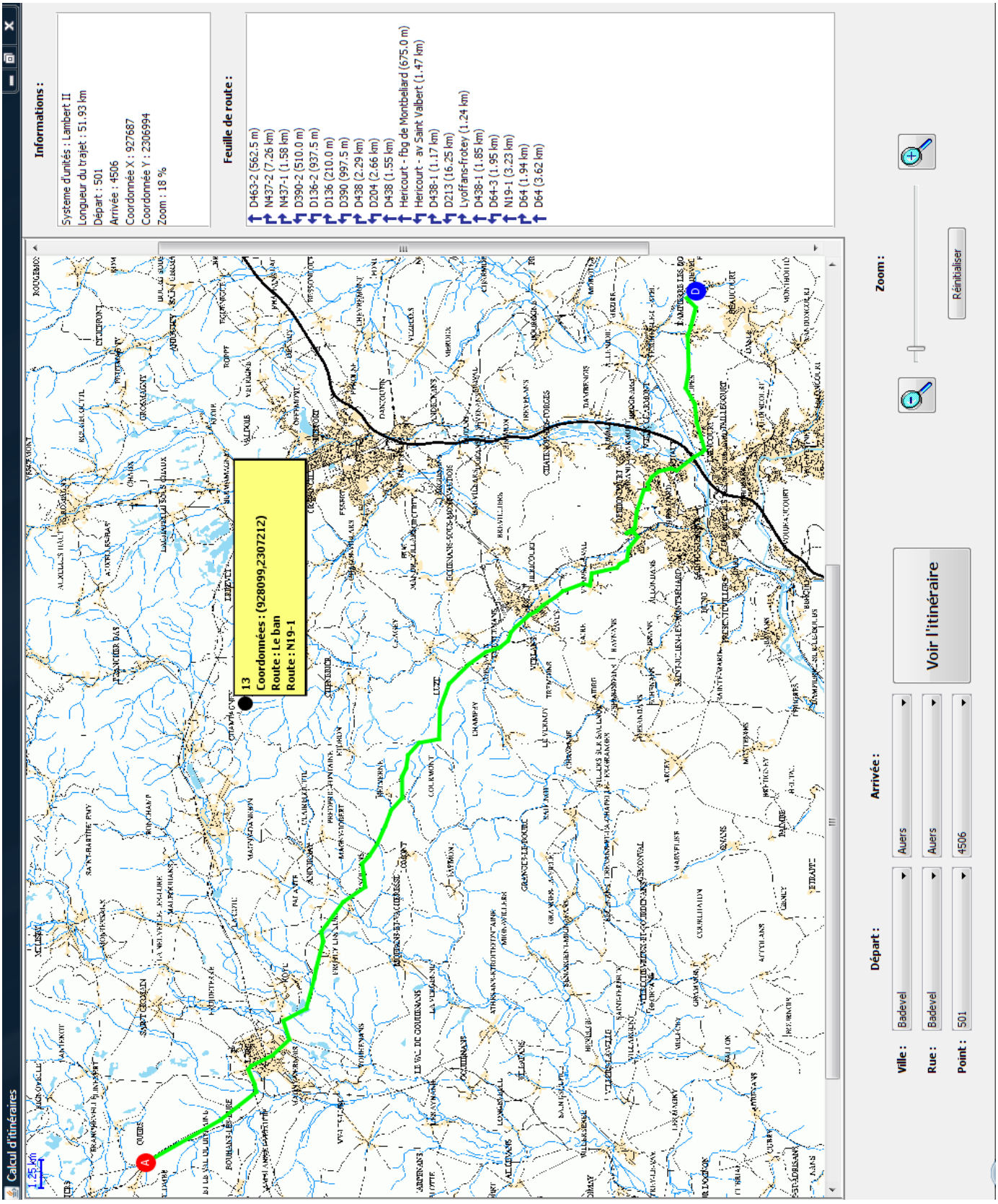
3) Etude des traitements :

Voici un organigramme décrivant rapidement le cas principal d'utilisation de l'application : la sélection des 2 points de départ et d'arrivée, et les traitements qui en découlent.



# III) DOSSIER DE CONCEPTION :

## 1) Présentation de l'interface homme-machine :



## 2) Utilisation de l'application :

On peut donc distinguer 3 zones distinctes sur cette IHM.

### a) La carte :

Dans un premier temps, la zone centrale concerne la carte.

Cette zone affiche plusieurs éléments :

- L'image de fond qui représente la carte de la région.
- Un aperçu de l'échelle actuelle dans le coin haut-gauche du cadre.
- L'itinéraire calculé, sour forme graphique, si l'utilisateur en a fait la demande.
- Les informations du point le plus proche de la souris de l'utilisateur.

L'utilisateur peut déplacer le point de vue de la carte en effectuant un « Drag & Drop » ou en utilisant les 2 barres de défilement, et peut zoomer / dézoomer dessus grâce à la molette.

L'utilisateur peut aussi, grâce au click-droit de sa souris, afficher un menu contextuel sur le point le plus proche de la souris. Ce menu permet plusieurs choses :

- Définir le point le plus proche de la souris comme point de départ / d'arrivée de l'itinéraire.
- Changer la couleur du tracé de l'itinéraire.
- Activer / désactiver le filtre anticrènelage sur la carte (filtre permettant d'atténuer l'effet escalier au bord des droites du graphique).

### b) Le panneau des informations :

Dans un second temps, la zone de droite est dédiée à l'affichage de diverses informations.

Une première zone de texte affiche les informations liées à l'état courant de l'application :

- Un éventuel message.
- Le système d'unité courant.
- La longueur totale du trajet calculé.
- Les numéros des points de départ et d'arrivée du trajet calculé.
- Les coordonnées actuelles de la souris, dans le système d'unité courant.
- Le niveau de zoom.

Une seconde zone affiche la feuille de route du trajet calculé. Pour chaque changement de route, le nom de la nouvelle route est affiché ainsi que la distance à parcourir sur cette route. Une petite icône affiche pour chaque changement de route si il s'agit d'un virage à droite ou d'un virage à gauche.

c) Le panneau des contrôles :

Enfin, la partie basse de l'IHM est dédiée au contrôle de l'application. Elle est composée de 2 zones.

La zone de gauche permet à l'utilisateur de sélectionner les point de départ et d'arrivée du trajet à calculer. Ces 2 points peuvent être choisis grâce à 3 listes déroulantes chacuns : une première pour le choix du type de route ou de la ville, une seconde pour le choix de la route, et une 3ème pour le choix du point.

La zone de droite permet un contrôle du zoom, de trois manière différentes :
- Un slider permet une modification progressive et continue.
- Deux boutons permettent une modification discrète par pas de 10%.
- Trois autres autres boutons permettent de se situer sur les trois valeurs remarquables que sont la vue globale, la vue en gros plan et le zoom initial (50%).

## 3) Description de l'application

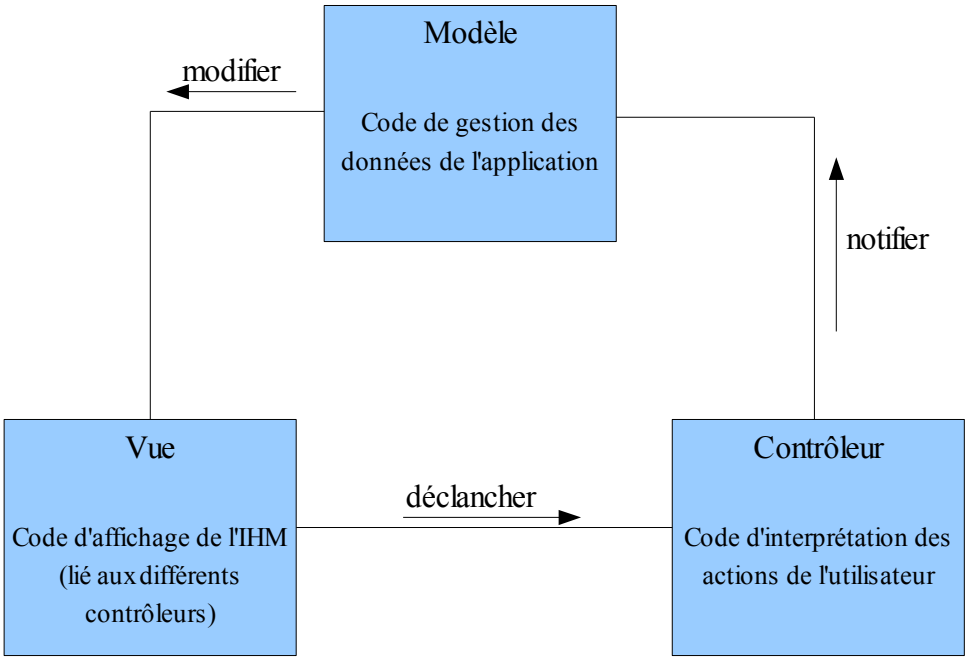
### a) Utilisation du modèle MVC :

L'architecture Modèle Vue Contrôleur (MVC) est un motif de conception pour le développement d'applications logicielles qui sépare le modèle de données, l'interface utilisateur et la logique de contrôle.

En Java, ce modèle est facilement mis en place grâce à la séparation dans des *package* des objets du modèle, de la vue ou du contrôleur.

Les objets attentifs aux actions de l'utilisateur (instances des classes qui implémentent les interfaces *ActionListener*, *MouseInputListener*, ...) font donc partie de la section « Contrôleur » du modèle MVC. Les classes dérivées des composants Swing (*JFrame*, *JPanel*, *JscrollPane...*) permettent l'affichage de l'IHM et font partie de la section « Vue » de ce modèle. Enfin, les classes qui régissent les données invisibles de l'application (*ReseauRoutier*, *PlusCourtChemin*, ...) sont elles associées à la partie « Modèle » de l'architecture MVC.

Voici le diagramme de collaboration présentant les interractions entre les éléments du modèle MVC tel que nous l'avons mis en place dans notre projet :



b) Présentation des différentes classes du projet

La classe Main (Main.java) ne fait partie d'aucun package et permet seulement le lancement de l'application.

Voici les autres classes que nous avons créé lors du développement :

Modèle (package « model ») :

- Application : il s'agit de la classe créée par le lanceur. Cette classe contient et gère tous les éléments de l'application : elle créé les différents objets, lies les composants de l'IHM (Vue) aux écouteurs d'évènement (Contrôleur), et contient tout le code permettant de bonnes relations entre les différents objets de l'application.
- ReseauRoutier : classe gèrant les données du réseau routier (routes, points), en parsant celles-ci depuis un fichier XML, et grâce a des méthodes permettant une lecture aisée de ces données depuis l'application.
- Route : gère les données des routes (points, sens).
- PlusCourtChemin : classe mettant en place l'algorithme de calcul de plus court chemin (Dijkstra). Elle permet de résoudre un plus court chemin dans un réseau de Noeuds et d'Arcs.
- EtatReseau : des instances de cette classe composent le plus court chemin calculé par l'algorithme Dijkstra. Elle contient un état d'un chemin : position actuelle, position prochaine.
- NomRue : classe d'analyse de chaine de caractère permettant la décomposition des noms des rues afin d'en extraire leurs catégories.

Vue (package « view ») :

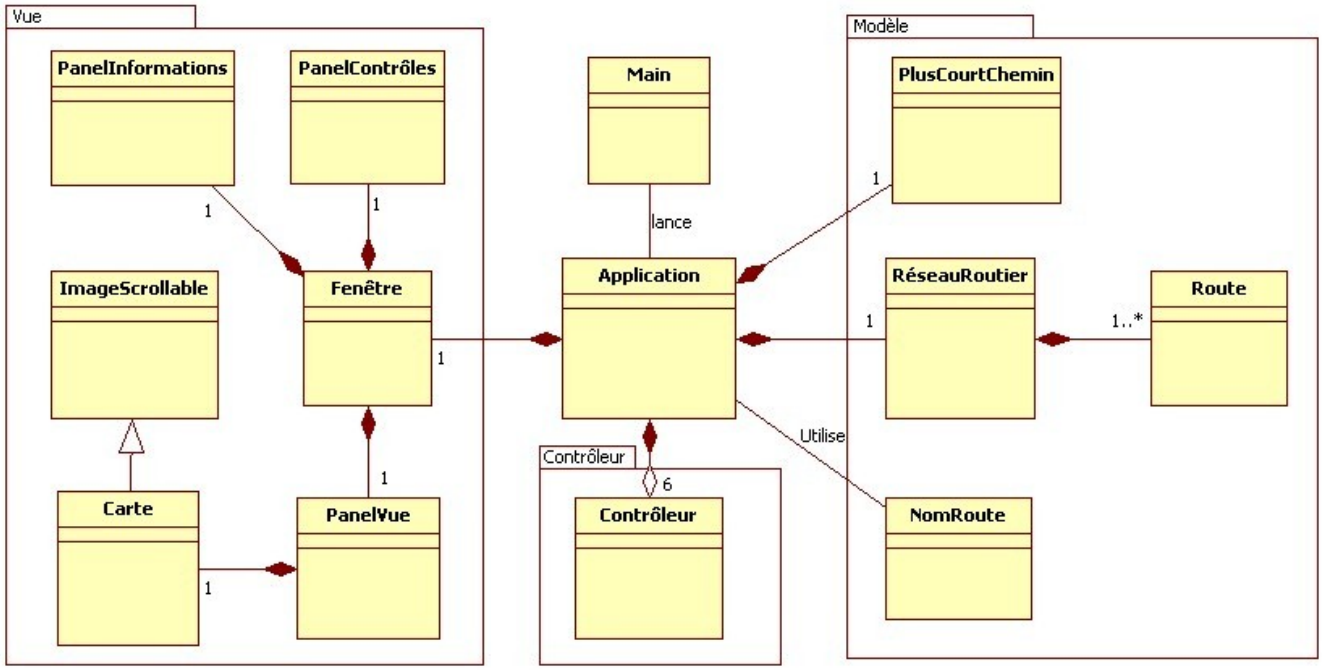
- Fenetre : classe dérivée de la classe *JFrame*. Elle est initialisée par l'application, et affiche la fenêtre contenant les différents panneaux.
- PanelControles : classe dérivée de la classe *JPanel*. Elle constitue la partie basse de la fenêtre, et contient la plupart des éléments permettants le contrôle de l'application par l'utilisateur : des listes pour choisir un itinéraire, des boutons pour contrôler le zoom.
- PanelInformations : classe dérivée de la classe *JPanel*. Elle constitue la partie de droite de la fenêtre, et met en place 2 listes contenant respectivement les informations sur l'itinéraire et la feuille de route.
- PanelVue : classe dérivée de la classe *JScrollPane*. Cette classe contient la carte de la région (instance de la classe Carte).
- ImageScrollable : classe permettant à la grande image d'être scrollée.
- Carte : classe dérivée de la classe ImageScrollable. Cette classe rajoute le traçage des différentes informations sur l'image de la carte de la région, et met en place un menu contextuel.

Contrôleur (package « controler ») :

- ControlleurBoutons : classe implémentant l'interface *ActionListener*. Elle écoute les actions sur les boutons « Voir l'itinéraire », « Zoom+ » et « Zoom- ». Ainsi, lorsque ceux-ci sont pressés, un objet *ActionEvent* est envoyé à la méthode *actionPerformed* de cette classe. La classe appelle alors une méthode de l'application en fonction de la source de l'action.
- ControlleurCarte : classe implémentant les interfaces *MouseInputListener* et *MouseWheelListener*. Elle permet la modification du point de vue de la carte avec la souris : un « Drag & Drop » pour le déplacer, la molette pour zoomer / dézoomer.
- ControlleurComboBox : classe implémentant l'interface *ActionListener*. Elle écoute les changements dans les ComboBox des villes et des rues, afin d'en notifier l'application dans le but de mettre à jour leurs ComboBox filles.
- ControlleurMenuContextuel : classe implémentant l'interface *ActionListener*. Elle permet de déclancher les actions adaptées lors du click sur un élément du menu contextuel de la carte.
- ControlleurSlider : classe implémentant l'interface *ChangeListener*. Elle permet le fonctionnement du slider de zoom.
- ControlleurScrollBar : classe implémentant l'interface *MouseListener*. Elle permet de détecter un déplacement de la carte effectué à partir des ScrollBar.

c) Diagramme de classe :

Voici un diagramme résumant les relations entre les classes qui composent l'application :



## 4) Explication des choix de conception :

### a) Les fonctionnalités :

L'application devant permettre de gérer des itinéraires, nous avons commencé par définir la méthode de sélection de départ et de l'arrivée. Nous avons choisi un système de liste déroulante, permettant de choisir la ville, puis la route et éventuellement le point exact. Ces informations sont extraites directement depuis le nom des routes du fichier XML, en considérant le format générique Ville – Route. Une fois le trajet sélectionné, un bouton lance la recherche.

Nous avons ensuite pensé à une méthode supplémentaire accordant plus de souplesse à l'utilisateur. Il s'agit de cliquer directement sur la carte pour choisir ces points. Pour cela nous avons commencé par afficher le point le plus proche du curseur de la souris. Si l'utilisateur utilise le clique-droit, on lui propose de définir ce point comme départ ou arrivée via un menu.

Le Zoom/Dézoom de la carte peut s'effectuer de différentes façons. Initialement nous n'avions implémenté que trois boutons (plus, moins et réinitialiser). Cette méthode n'était pas très ergonomique car il fallait cliquer plusieurs fois pour faire un gros zoom.

C'est pourquoi nous nous sommes orienté vers un slider, qui permet une gestion rapide et fine de la taille de la carte. Pour finir nous avons rajouté deux boutons pour se rendre aux deux zoom extrêmes (vue globale et gros plan).

Les informations données à l'utilisateur consistent en des informations diverses et une feuille de route. Ces deux fonctions sont assurés par des *JList*. Nous avons sélectionné ce composant Swing pour la facilité d'ajout/suppression d'éléments séparément (par comparaison à une zone de texte par exemple).

### b) La gestion de la carte :

Initialement nous avions implémenté une méthode lourde de gestion de la carte, et très peu adaptée au zoom à la volée. Il s'agissait d'insérer un *JLabel* dans le *JScrollPane*, et d'utiliser la méthode *setImage()* pour changer la carte. Un changement de zoom appelait quant à lui la méthode *getScaledInstance()* sur l'image pour la redimenssioner. Au vu des piètres performances, nous avons opté pour une autre solution.

La technique de « custom painting » est celle que nous avons utilisé et rend quant à elle de très bon résultat. Il s'agit de créer une classe personnalisée héritant d'un *JComponent* quelconque qui remplace la méthode *paintComponent()* de ce composant (la méthode *paint()* étant dangereuse car elle risque de repeindre la bordure et de déborder du composant).

Nous avons donc créé une classe Carte héritant de *JLabel*, qui redéfinit la méthode *paintComponent()* héritée. Cette dernière fais appel à la fonction *drawImage()*, puis peint éventuellement les objets utiles à la représentations du trajet demandé par l'utilisateur.

Afin de gérer le zoom avec cette nouvelle méthode, nous avons utilisé un objet de type *AffineTransform* qui est mis à jour à chaque modification du zoom.

Nous avons ensuite remarqué que se déplacer sur la carte était assez fastidieux. Après quelques recherches nous avons découvert l'interface *Scrollable* qui permet d'implémenter un « Scroll-Savy Client », c'est à dire des déplacements par scrolls facilités. Nous avons alors fait hérité notre classe Carte d'une nouvelle classe ImageScrollable, qui implémente cette interface Scrollable.

Mais nous trouvions cela insuffisant, le déplacement étant encore assez laborieux. Nous avons donc rajouté un system de drag à l'aide d'un écouteur de type *MouseInputListener*. Nous avons ajouté des changement de l'image du curseur afin d'améliorer le feedback de l'utilisateur.

Finalement nous avons remarqué que lors d'un zoom/dézoom de la carte, le point de référence n'était pas conservé ce qui était très désagréable. Nous avons alors rajouté un système de sauvegarde de ce point (pris au centre de l'écran), pour resituer la vue dessus.

c) Divers :

Nous avons aussi rajouté quelques fonctionnalités pour le confort de l'utilisateur. Le menu de clic-droit permet de sélectionner la couleur du tracé du trajet, et d'activer le filtre anticrènelage.

La feuille de route indique aussi dans quel sens tourner. Cette information est récoltée suite à un calcul mathématique d'angle entre les deux axes.

Enfin, lors de la sélection d'un itinéraire grâce aux listes déroulantes, la vue se met automatiquement au bon zoom et au bon emplacement afin d'afficher le trajet calculé en globalité.

# III) BILAN :

## 1) Capacités et performances de l'application :

Au final, l'application semble assez performante, car l'utilisateur dispose d'une interface ergonomique et que tous les traitements se font instantannéments.

En particulier, la gestion du zoom sur la carte est satisfaisante grâce à l'utilisation des méthodes paintComponent() et repaint() proposée par Swing. Nous avions essayé les méthodes lourde d'AWT qui étaient beaucoup trop lente ( 2 secondes pour recalculer la nouvelle image .)

## 2) Analyse critique des résultats :

L'application consomme beaucoup de mémoire virtuelle (environ 100Mo), ceci est principalement du à la taille de l'image, et aurait peut-être pu être optimisé, en découpant l'image pour ne garder que la zone visible par exemple.

Un filtrage de l'image aurait été utile pour les zoom bas, car la qualité de celle-ci se dégrade facilement lors de la réduction de sa taille.

Le contrôle du système d'unité n'a pas été implémenté, à cause d'un manque de temps.

Le layout de la zone contrôle n'est pas non plus finalisé. Les composants ne se redimmensionnent pas très bien lors d'une réduction de la taille de la fenêtre, et l'espace en trop n'est pas très bien géré. A cette occasion, nous avons pu constater la difficulté de réaliser un layout correct à la main.

## 3) Extension de l'application :

Afin d'étendre l'application, plusieurs fonctionalités sont envisageables. Nous n'avons pas implémenté ces fonctionnalités car elles ne rentrent pas dans le cahier des charges, et nous souhaitions avant tout proposer une application conforme aux besoins. Les idées que nous avons eu comportent, entre autre :

- Ajout de points/routes sur la carte
- Sauvegarde de ces nouvelles données dans le fichier XML
- Changement de l'image Carte
- Changement du fichier XML correspondant à la Carte
- Formulaire de recherche libre des noms de route