HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Gestion d'un carrefour à l'aide d'un réseau de Petri

Rapport du projet

Auteur : Bise Léonard
Professeur : Zysman Eytan
Filière : Informatique Embarquée (ISEC)
Cours : Programmation Concurrente (PConc)
Date : 10 Mai 2018

Table des matières

1 Introduction

Dans le cadre du cours Programmation Concurrente (PConc), il nous a été demandé de créer un programme qui simule le comportement d'un carrefour. Le carrefour est composé de deux flux, un Sud-Nord et l'autre Ouest-Est ou les voitures circulent dans un seul sens. A l'intersection des deux flux sont placé deux feux de signalisation, un par flux, ils permettent ou non au voiture de passer.

La gestion du système est fait grâce à un réseau de Petri qui réagit aux événement et qui produit des actions.

2 Cahier des Charges

Dans le cadre du projet il nous est demandé de créer un projet codé en java comprenant les éléments suivants :

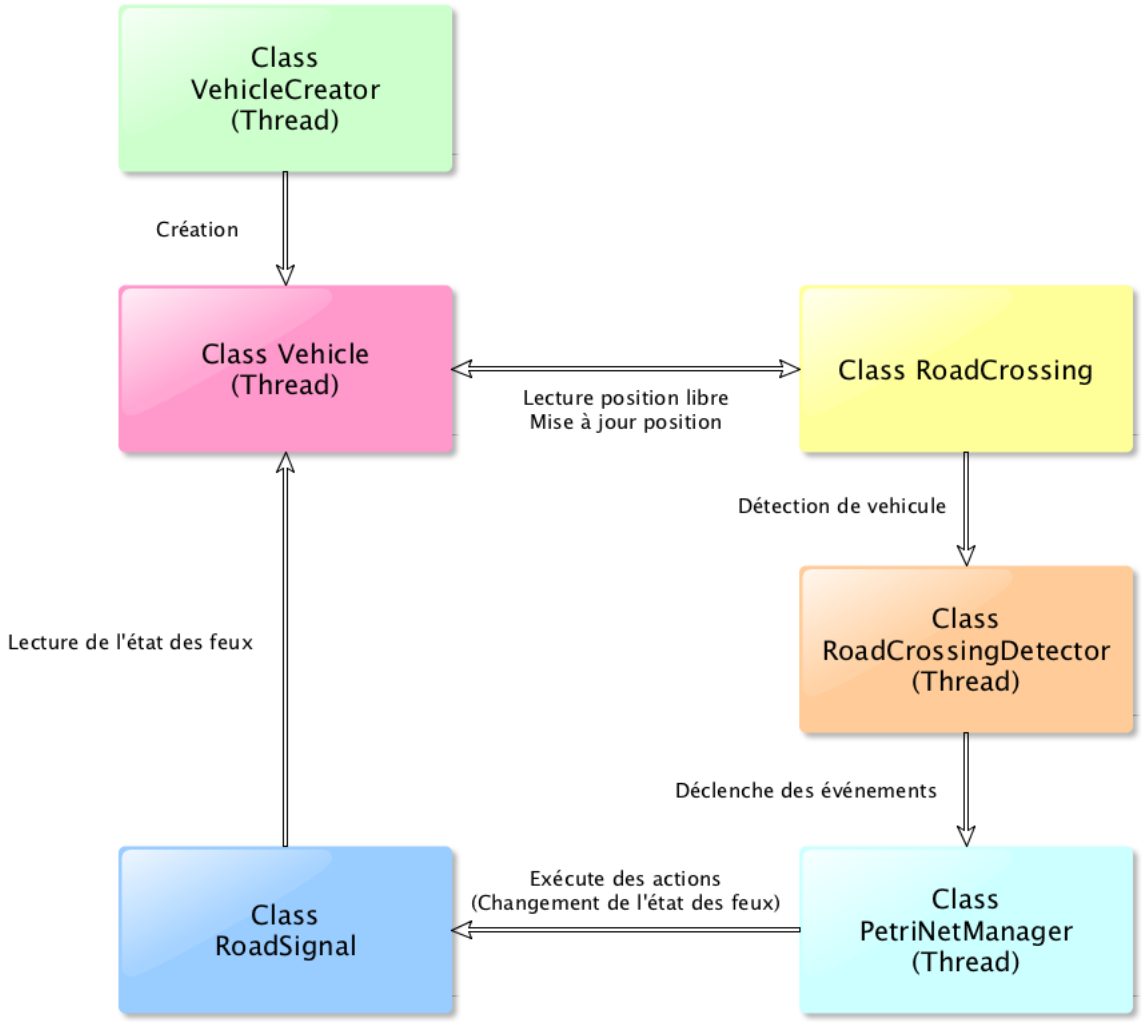
- Créer un gestionnaire de réseau de petri auquel il est possible de donner un fichier de configuration pour créer un paysage (création des places, transition, arcs de pre et post incidence).
- Lorsque le réseau de Petri est créé, il est possible de lancer la simulation, à chaque tick le gestionnaire va faire évoluer le réseau de Petri en regardant quelle places sont sensibilisée, il va ensuite consommer et produire des jetons aux places requises.
- Lorsqu'un jeton arrive dans une place, si l'utilisateur a lié une action à la place, le gestionnaire va lancer l'exécution de l'action
- L'environnement éxtérieur doit pouvoir ajouter des événements dans une queue, ce qui permettra aux transitions d'être franchies lorsqu'elles sont sensibilisées.
- Utiliser le gestionnaire de réseau de Pétri afin d'implémenter un réseau qui permet la gestion d'un carrefour avec deux flux perpendiculaire de voiture et qui est à sens unique. Il existe deux réseau de Petri qui sont décrit dans les slides de Monsieur Zysman, un avec timer intégré et l'autre sans. Pour ce projet le plus simple des deux sera utilisé, c'est à dire sans timer intégré. Le réseau de Petri est défini avec détail dans les slides à la page 8. C'est ce réseau qui a été utilisé comme base pour écrire le contenu du fichier roadCrossingRDP.cfg qui est servie pour initialiser le réseau de Petri.

En utilisant le gestionnaire de réseau de Petri, il faudra créer les éléments suivants et les faire intéragir avec le réseau pour faire évoluer le système.

- Un objet responsable de la gestion des deux routes, il permettra aux véhicules de savoir s'ils peuvent avancer ou si un véhicule se trouve déjà dans la case devant eux. La gestion se fait grâce à deux vecteurs un par flux, le croisement est donc présent dans chacun des deux flux
- Deux threads, un par flux, qui créeront à des temps aléatoire de nouveaux véhicules, ils seront créés au début de la route
- Un objet véhicule, cet objet, dès sa création, commencera à avancer de case en case en direction de la fin de la route. Lorsqu'il arrivera juste devant le carrefour il vérifiera si le feu est vert avant de pouvoir entrer dans le carrefour. Lorsqu'il atteint la fin de la route il disparait
- Deux détecteurs, un par flux, qui auront comme tâche de déclencher un événement lorsqu'ils détecteront un véhicule présent juste avant le croisement
- Deux détecteurs, un par flux, qui auront comme tâche de déclencher un événement lorsqu'ils détecteront que le croisement est vide
- Deux feu de signalisation, un par flux, qui autoriseront ou non les véhicule du flux à passer dans le croisement

3 Architecture

L'image suivante montre l'architecture générale de l'application.



4 Le mécanisme du Réseau de Petri

La classe PetriNetManager implémente le gestionnaire de réseau de Petri. D'autres classes sont utilisées pour la gestion des réseaux, elles sont décrites ci-dessous.

- PetriNetManagerTest : Contient quelques tests du bon fonctionnement du gestionnaire de réseau
- PetriArc : Décrit les caractèristiques lié aux arcs de pré et post incidence comme le type (simple, test, inhibit) et leur poids
- PetriPlace : Une place dans un réseau de Petri ainsi que sont action liée si elle en as une
- PetriPlaceInterface : Une interface qui permet à la classe qui l'implémente d'être ensuite déclenchée par le gestionnaire de réseau lorsqu'un jeton entre dans la place qui y est liée.
- PetriTransition : Contient les informations relative à une transition faisant partie du réseau de Petri

La classe PetriNetManager est la classe centrale du gestionnaire de réseau de Petri. Après l'instanciation de la classe, un réseau de Petri peut être initialisé en utilisant la méthode loadFromTextFile avec un fichier de configuration. Ceci créera toute les places, transitions et arcs requis pour le réseau décrit.

Une fois le réseau initialisé l'utilisateur à la possibilité de faire les actions suivantes :

- Lier une place du réseau avec une classe implémentant l'interface PetriPlaceInterface avec la méthode setPlaceAction. Lorsque le gestionnaire du réseau détecte l'entrée d'un jeton dans la place, il déclenchera automatiquement l'exécution de l'action
- Lancer le gestionnaire de réseau de Petri avec la méthode start. Cette action lancera l'exécution du Thread de PetriNetManager qui après chaque tick effectuera la simulation du réseau
- Une fois le gestionnaire lancé, l'utilisateur doit déclencher des événements liés à une certaine transition au moyen de la méthode newEvent ce qui permettra le franchissement de la dite transition si elle est préalablement sensibilisée

Le coeur de la simulation du réseau de Petri est effectuée par la méthode step. Cette méthode simule un pas de l'évolution du réseau de Petri, elle est cadencée par la durée d'un tick, c'est à dire un temps entre deux pas.
La liste suivante décrits les opérations effectuées par la méthode step, elle se découpe en quatre phases.

- Phase 0 : Les actions des places qui ont eu de nouveaux jetons sont appelées
- Phase 1 : Le gestionnaire détermine les transitions qui sont sensibilisées, c'est à dire que les places qui y sont liées contiennent le nombre de jeton requis, et les ajoute dans une liste de transition sensibilisée. La liste est ensuite mélangée pour que les transitions aient un ordre aléatoire

— Phase 2 : Pour chaque tranisition qui sont dans la liste des transitions sensibilisées, le gestionnaire vérifie si l'événment associé a été déclenché, c'est à dire si il est présent dans la queue d'événement. Si c'est le cas, les jetons des places qui sont liées à la transition sont consommés et elle est ajoutée à la liste des transitions franchies. La liste des transitions sensibilisées est à nouveau parcourue pour vérifier si certaines transitions ne le sont plus suite à la consommation de jetons
— Phase 3 : Les jetons nécessaire sont produit dans les places liées aux transitions qui ont été franchies

5 Implémentation des acteurs

Le projet de gestion de carrefour demande la création de différents acteurs qui sont listé ci-dessous.

- RoadCrossing : La grille sur laquelle les voitures se déplace. Cette classe est passive, elle ne fait que d'être modifié par d'autre classe, elle n'as pas son propre Thread. Les instance de Vehicle, l'utilise pour savoir si elles peuvent avancer ou si un autre véhicule se trouve déjà devant eux
- RoadCrossingDetector : Les detecteurs utilisés pour déclencher des événements dans le réseau de Petri. Il y'en a deux par flux. Un placé juste avant le carrefour déclenchera l'événement lié à la détection de flux voulant entrer dans le carrefour, cela permet au réseau de Petri de faire passer le feu de l'autre flux au rouge et ainsi permettre au premier flux de pouvoir passer
- RoadSignal : Le feu routier, c'est un élément passif qui ne fait que d'avoir un état, soit rouge ou alors vert. Il y a un feu par flux, les véhicules lorsqu'ils arrivent devant le carrefour doivent veiller à vérifier que le feu est vert avant de pouvoir y rentrer. L'état des feux est modifiés au travers de deux actions du réseau de Petri.
- EventManager : C'est une classe qui permet de définir des événements réél et les associer a un événement du réseau de Petri en utilisant à l'interne des classes de type Event. C'est donc l'interface entre le monde réél et le réseau de Petri
- RoadCrossingEventManager : Utilise la classe EventManager pour définir les événements qui sont propres au problème du carrefour (par exemple arrivée devant le carrefour ou sortie du carrefour)
- SignalStateAction : Cette classe implémente l'interface PetriPlaceAction et défini donc l'action à effectuer lorsqu'un feu doit changer d'état. Elle directement appellé par le gestionnaire de réseau de Petri quand nécessaire et elle contient une réference au feu qu'elle doit gérer.
- Vehicle : Définit le comportement d'un véhicule, après avoir été créé il commence à avancer régulièrement le long de son flux en utilisant la grille définie par la classe RoadCrossing. Juste avant de rentrer dans le carrefour la couleur du feu sera vérifiée pour voir si il est possible d'avancer. Lorsqu'elle arrive à la fin de la route, le véhicule a terminé sa vie et est détruit.
- VehicleCreator : Ce Thread est responsable de la création de véhicule dans un certain flux de manière régulière
- RoadSignalTimerAction : Cette classe représente l'action a effectuer lorsqu'un véhicule arrive devant le carrefour et que sont feu est rouge. Dans ce cas, un timer va être enclenché au terme duquel la classe RoadSignalTimerTask sera exécutée
- RoadSignalTimerTask : La tâche liée à RoadSignalTimerAction. Elle déclenche un événement qui permet au système de mettre le feu du flux qui est actif au rouge et de passer celui de l'autre flux au vert

— RoadCrossingGUIThread : La tâche qui est responsable de rafraîchir les informations affiché sur l'interface graphique

6 Les échanges entre acteurs

Le déclenchement des événements se fait par le biais de la classe RoadCrossingEventManager, elle est instanciée dans la fonction main et sa référence est donnée à tous les acteurs qui doivent déclencher des événements liés à des transitions du réseau de Petri. Cette classe fait l'interface entre le monde réél et le réseau de Petri, au moyen de la classe EventManager qui contient une référence vers le PetriNetManager, cela lui permet de rajouter des événements à la queue du gestionnaire et ainsi de permettre le franchissement des transitions. Cette classe définit les événements suivant.

- triggerCarBeforeCrossing : Un véhicule est présent devant le carrefour. Cet événement est déclenché par un détecteur dans le cas ou le feu est rouge ce qui permet au système de faire passer l'autre feu au rouge et ainsi de permettre au véhicule de franchir le carrefour
- triggerCrossingEmpty : Le carrefour est vide. Déclenché par un détecteur, il permet de s'assurer que le carrefour est vide avant de passer le feu du flux bloqué au vert
- triggerCarExitCrossing : Un véhicule a quitté le carrefour. Il est utilisé pour pouvoir faire quitter le jeton de la place "FA ou FB Dans carrefour"
- triggerGreenLight : Un véhicule a passer le feu vert. Permet à la transition "Autorisation FA/FB" d'être franchie
- triggerTimer : Déclencher par RoadSignalTimerTask afin de permettre le changement de vert au rouge

La classe RoadCrossing qui gère les deux routes, est aussi responsable de l'instanciation des deux feux de signalisation. Lorsque le véhicule arrive juste avant le carrefour, il récupère l'instance de RoadSignal de son flux afin de vérifier son état et éventuellement passer le carrefour si possible.

7 Conclusion

J'ai eu beaucoup de plaisir à travailler sur ce projet, malheureusement du au manque de temps la structure de l'application n'est pas aussi bien pensée que je l'aurais voulu. En effet les interactions entre les différents acteurs peuvent certainement être mieux réalisées. De plus, le code de certaines méthodes est clairement brouillon et je l'aurais volontiers réécris. La gestion des détecteurs peut être mieux réalisée, un détecteur devrait vivre jusqu'à la détection de l'événement pour lequel il est programmé, ensuite envoyer un événement au réseau de Petri et puis se terminer, au lieu de cela pour l'instant le détecteur avant le croisement est permanent. J'ai par contre eu le temps de modifier le comportement du détecteur au croisement pour suivre ce schéma. J'aurais également voulu pouvoir utiliser des fichiers XML au lieu de text pour la configuration du réseau. Les bases ont été mise en place pour se faire mais je n'ai pas eu le temps de les terminer.