|  |
| --- |
| Isima 3ème années |
| Projet simulation |
| Simulation d’un carrefour |
|  |
| **Jean-Christophe SEPTIER Maxime ESCOURBIAC** |
| **17/03/2012** |

Table des matières

[Introduction 1](#_Toc319776593)

[1. Partie Simulation 2](#_Toc319776594)

[1.1. DEVS et Framework SimStudio : 2](#_Toc319776595)

[1.2. Modèle utilisé : 3](#_Toc319776596)

[1.3. Implémentation : 4](#_Toc319776597)

[1.3.1. Generator : 4](#_Toc319776598)

[1.3.2. TrafficLight : 4](#_Toc319776599)

[1.3.3. TrafficLightSecondaire : 4](#_Toc319776600)

[1.3.4. Collector : 5](#_Toc319776601)

[1.3.5. TrafficLightSystem : 5](#_Toc319776602)

[2. Visualizer : 6](#_Toc319776604)

[2.1. Implémentation : 6](#_Toc319776605)

[2.2. Enchainement des feux : 7](#_Toc319776606)

[Conclusion 10](#_Toc319776607)

Table des illustrations

[Figure 1: Modèle du Framework 2](#_Toc319777074)

[Figure 2: Modèle de la simulation 3](#_Toc319777075)

[Figure 3: Etat du trafficLight 4](#_Toc319777076)

[Figure 4: Visualizer 6](#_Toc319777077)

[Figure 5: Bout de code du changement d'image 7](#_Toc319777078)

# Introduction

Lors du cours de simulation, il nous a été demandé de réaliser une simulation d’un carrefour avec les feux tricolores.

Cette application devait contenir deux parties :

* Une partie trafficsimulation, qui va simuler les feux, et écrire une trace de la simulation. Cette partie est réalisée en utilisant un Framework implémentant DEVS.
* Une partie visualizer qui permet de visualiser graphiquement la trace.

Les feux sont des feux américains. Des événements peuvent avoir lieu pendant la simulation. Il est possible d’avoir des pannes, des patrouilles permettant de réparer les feux, et des piétons demandant de faire passer le feu au rouge.

Dans ce rapport, nous allons d’abord vous présenter la partie simulation, avec le modèle choisit et une explication rapide de l’implémentation, puis nous présenterons le vizualizer.

# Partie Simulation

## DEVS et Framework SimStudio :

DEVS, pour Discrete Event System Specification est une norme qui permet de définir très précisément des simulateurs à événements discrets. L’objectif est de pouvoir modéliser ces systèmes, et de simuler leur fonctionnement. DEVS se place donc dans la norme de modélisation et de simulation. C’est une norme relativement récente, peu répandue mais avec des spécifications très précises.

Avec DEVS, un système complexe se décompose en sous-systèmes. On a donc des modèles atomiques et des modèles couplés.

SimStudio est un Framework implémentant cette norme.

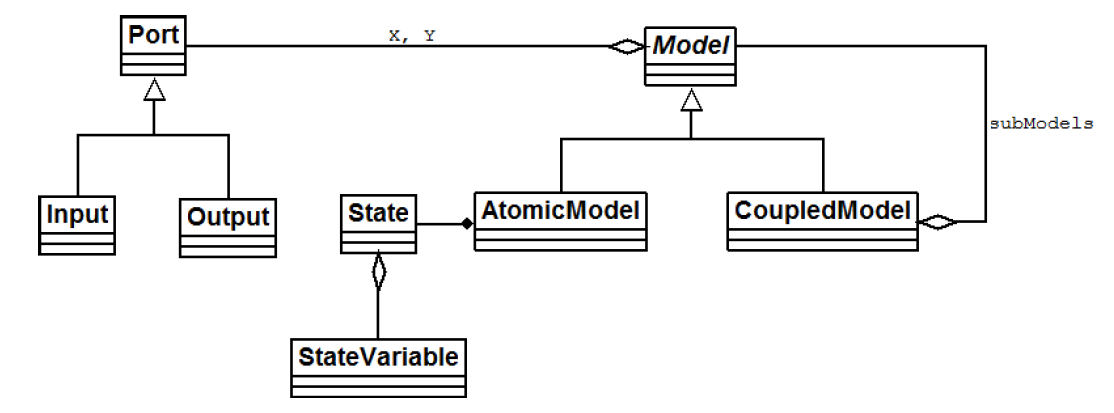


Figure : Modèle du Framework

La classe Model regroupe les attributs et les méthodes de base communes aux modèles atomiques et couplés. Elle contient des ports outputs et inputs qui permettent aux modèles de communiquer entre eux. State représente l’état d’un modèle atomique.

## Modèle utilisé :

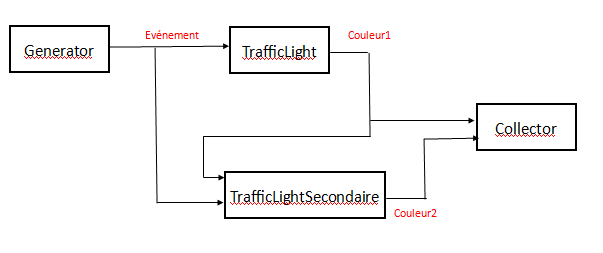


Figure : Modèle de la simulation

Le générator est un modèle qui va créer des événements aléatoires. Il envoie les événements sur le TrafficLight et le TrafficLightSecondaire. Les événements peuvent être Failure(panne), Recovery(passage d’une patrouille qui répare le feu en cas de panne), walker1(Piéton sur le feux 1), walker2 (Piéton sur le feux 2).

Le TrafficLight est le modèle qui simule les feux principaux. C’est un feu américain : l’enchainement des couleurs est vert-orange-rouge-orange-vert… Il reçoit les événements du générator. En cas de panne, il clignote en orange. Il se remet dans le fonctionnement normal si la patrouille passe lors d’une panne. Si un piéton demande de passer, et que le feu est au vert, celui va passer à l’orange, pour qu’il puisse ensuite passer.

Le trafficLightSecondaire simule les feux secondaires. Il reçoit la couleur des feux principaux et prend la couleur inverse de ceux-ci. Ils clignotent s’ils sont en panne.

Le collector est le modèle qui permet de créer la trace de la simulation. Il récupère les couleurs du TrafficLight et du TrafficLightSecondaire et l’écrit dans un fichier txt.

## Implémentation :

### Generator :

Cette classe va permettre de générer tous les événements aléatoires.

Le temps de génération d’un futur événement est tiré aléatoirement. Il est possible pour l’utilisateur de changer le temps minimum et le temps minimum d’apparition d’un événement en modifiant une classe Configuration. Ensuite, un tirage au sort permet de choisir quel type d’événement va être générer. Les probabilités d’avoir un type d’événement est également modifiable dans la classe modification.

L’événement sera envoyé sur un port *Source.out*.

### TrafficLight :

Cette classe génère la couleur des feux. Elle prend les événements sur un port nommé *TARGET.COMMAND*. Il y a plusieurs états possibles que l’on a mis dans un enum.

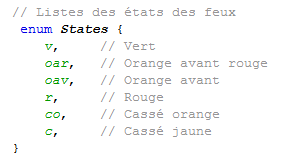


Figure : Etat du trafficLight

La méthode *deltaInt* permet de changer l’état en fonctionnement normal du TrafficLight. DeltaEx va permettre de changer l’état quand il reçoit un événement extérieur.

La couleur du feux sera envoyé sur un port *TARGET.COLOR*.

### TrafficLightSecondaire :

Cette classe n’a pas de comportement interne. L’état change en recevant les informations des feux principaux sur un port *SECONDAIRE.COLOR*. L’état sera l’état inverse des feux principaux, sauf en cas de panne, ou l’état est le même. La couleur de ce feu sera envoyée sur le port *TARGET.COLOR*.

### Collector :

Cette classe récupère les informations de feux sur son port *SINK.STORE*. Elle va ensuite écrire dans un fichier SINK.txt les feux avec le temps actuel de la simulation est la couleur des feux (la couleur est précédée par 2 pour différentier les feux secondaires).

### TrafficLightSystem :

Cette classe permet de créer le modèle de simulation. Il créé les sous-modèles vu précédemment et relie les ports entre eux pour qu’ils puissent communiquer entre eux.

# 

# Visualizer :

Cette partie permet de visualiser la trace créée par la simulation afin de tester le programme.

## Implémentation :

Notre programme permet d’ouvrir un fichier trace, et de lancer une simulation.

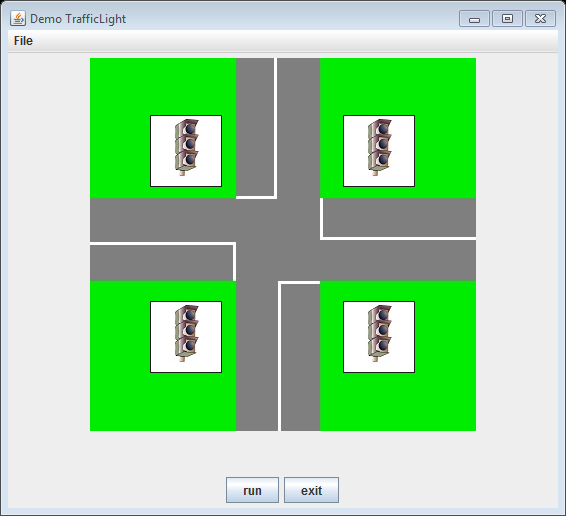


Figure : Visualizer

Un menu File permet de choisir le fichier trace en utilisant un JFileChooser. Le bouton run permet de lancer la simulation. Le visualiser va lire le fichier trace, et attendre le temps de la simulation, puis va afficher les images selon les informations reçu. Les images ont été créées sous paint.

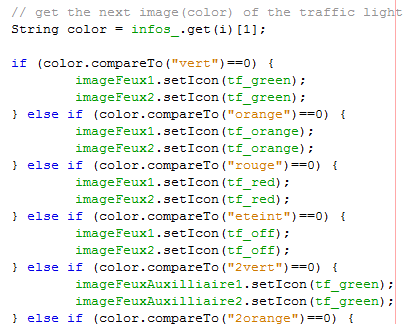


Figure : Bout de code du changement d'image

## Enchainement des feux :

Cet enchainement montre un fonctionnement normal du feux.

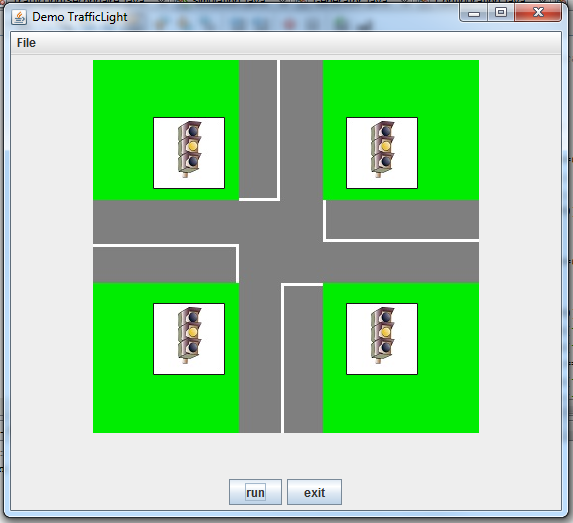
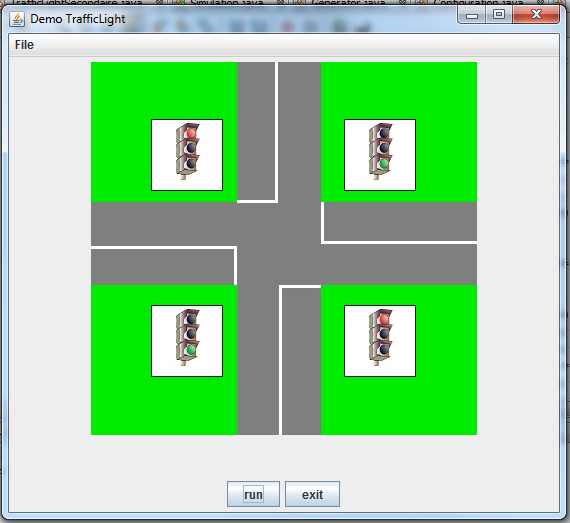


Figure : Simulation temps : 0s Figure 5: Simulation temps : 5s

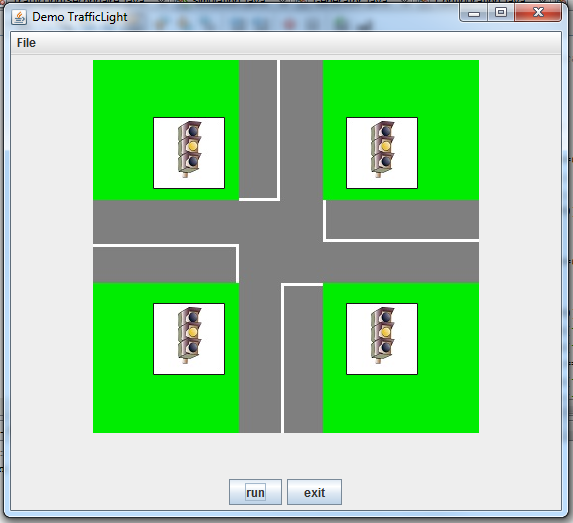
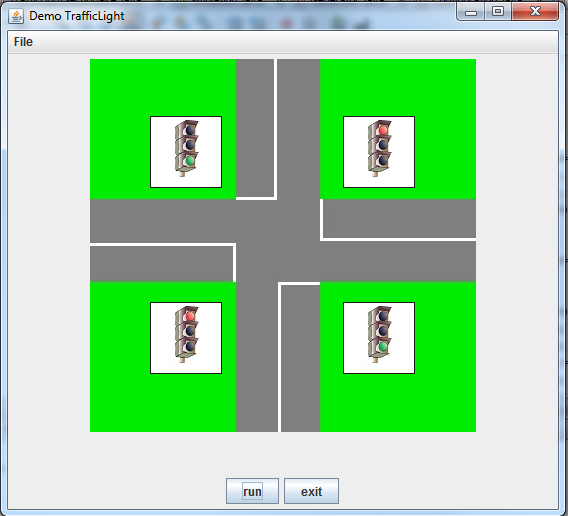


Figure 6: Simulation temps : 7s Figure 7: Simulation temps : 12s

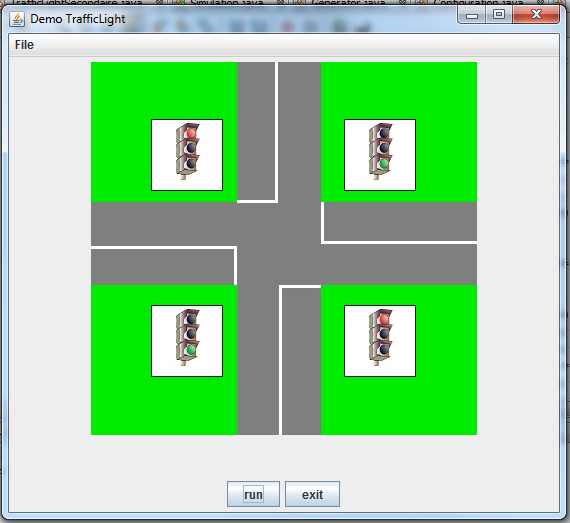


Figure 8: Simulation temps : 14s

Cette séquence va se répéter tant que le feu n’est pas en panne. Les temps seront plus courts en cas de demande d’un piéton.

Cet enchainement montre le fonctionnement d’un feu en panne.

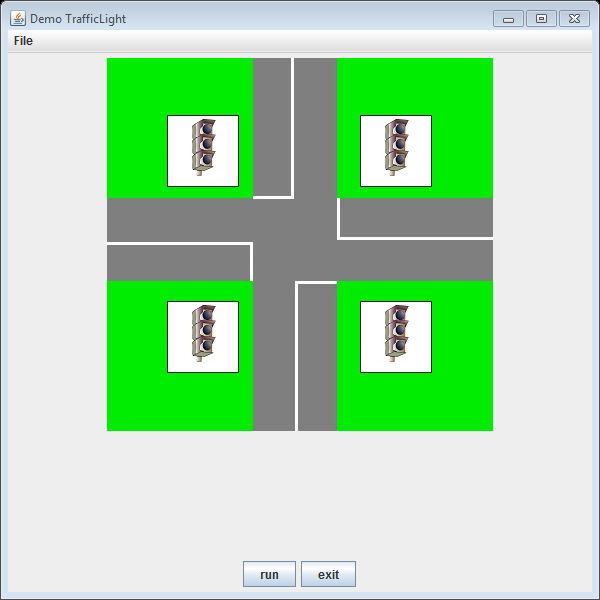
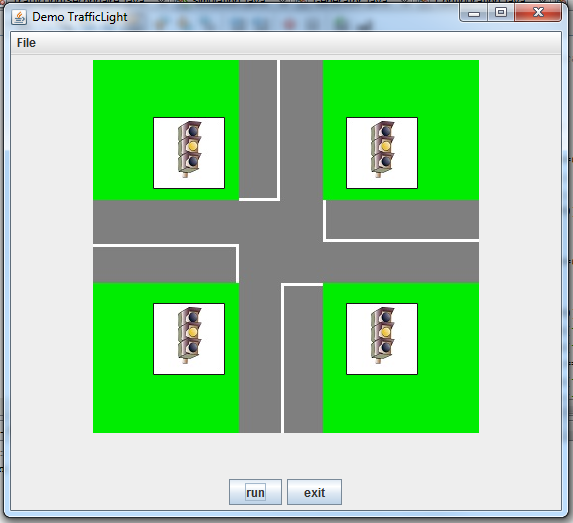


Figure 9: Simulation temps : 0s Figure 10: Simulation temps : 1s

Cette séquence va s’enchainer jusqu’à ce qu’une patrouille vienne réparer les feux.

# Conclusion

Ce TP nous a permis de réaliser une simulation de feux tricolores. Nous avons pu utiliser ce Framework, qui permet de réaliser très rapidement une simulation, une fois le principe compris.

Le système de sous modèle permet de décomposer un modèle en plusieurs afin de rendre plus facile l’implémentation d’une solution.

Ce TP nous a également permis d’appliquer la norme DEVS dans un cas concret, permettant de démontrer son utilité de l’utiliser pour rendre une simulation plus facile à réaliser.