TP2: Livraison spatiale

Cédric Buron

30 novembre 2020

Introduction

Dans ce TP, nous nous intéressons à une compagnie spatiale s'occupe de livraison. Chaque planète est gérée par un **gestionnaire de planètes** qui assigne des livraisons depuis la planète dont il est responsable vers une autre planète. Ce TP se focalisera principalement sur les aspects environnement, interaction et organisation du système multiagents.

Environnement

Dans ce TP, l'environnement a deux composantes : la première concerne l'espace. Celui-ci est peuplé d'objets immobiles, les planètes, qui sont représentées par les gestionnaires qui les contrôlent. La seconde concerne les biens à livrer, qui sont des objets qui seront manipulés par les vaisseaux.

Question 1– Caractérisez l'environnement du système comme nous l'avons vu en cours. Justifiez.

Question 2– La classe `Item` n'hérite pas de la classe `Agent` de mesa. En quoi cette entité ne constitue-t-elle pas un agent ?

Un autre élément de l'environnement est le réseau de routes utilisé par les agents, `SpaceRoadNetwork`. Ce dernier est modélisé par un agent pour des raisons pratiques, mais il n'est ni autonome ni réactif. Le réseau de route constitue un graphe dont les nœuds sont les planètes et les arêtes sont les routes entre les planètes ; ce sont ces routes qui sont empruntées par les agents. Ce graphe est modélisé au moyen du package `networkx`.

Communication : introduction à spade

Il existe donc deux principaux types d'agents dans ce TP : les agents `PlanetManager` génèrent des biens et utilisent le **Contract Net Protocol** pour les proposer à la livraison. Les vaisseaux `Ship` essaient d'obtenir les biens qui leurs correspondent le mieux. Ces agents ont chacun besoin de connaître l'ensemble des agents de l'autre type. Les `PlanetManager` sont quant à eux obligés de connaître les autres `PlanetManager` afin d'en déterminer un qui soit le destinataire du bien. Dans tous les cas, ces agents sont des agents capables de communiquer. Pour représenter cela, nous allons utiliser un autre framework multi-agents, le framework spade (*smart python agent developing framework*). Nous utiliserons ce frameworks dans les TP 4 et 5. Pour installer sapde, ouvrez un terminal et tapez :

```
pip install spade
```

Puisque nous utliserons aussi networkx, installez-le également en tapant :

```
pip install networkx
```

Contrairement à mesa, qui est conçu pour la simulation et le test d'hypothèses de recherches, spade est conçu spécifiquement pour faire de l'échange de message dans des environnements où les agents sont mobiles, peuvent se connecter depuis plusieurs machines etc. Les messages sont formatés en utilisant le protocole xmpp. Cependant, spade n'implémente pas lui-même un serveur xmpp, uniquement des clients pour les agents. Il va donc nous falloir installer un serveur xmpp. Nous utiliserons *prosody*, qui est conseillé dans la doucumentation de spade. Pour installer prosody, rendez-vous sur la page de prosody et **suivez scrupuleusement les instructions**. Et surtout **Ne restez pas bloqués**. Si vous n'arrivez pas à installer prosody, envoyez-moi un message. Une fois prosody installé, lancez un terminal et tapez :

```
sudo prosodyctl start
for i in {0..50};
    do sudo prosodyctl register ship-$i localhost password-ship-$i;
done
for i in {0..50};
    do sudo prosodyctl register planet-$i localhost password-planet-$i;
done
```

si vous êtes sous unix et

```
prosodyctl start
for ($i=0; $i -le 50; $i++) {
    prosodyctl register ship-$i localhost password-ship-$i
}
for ($i=0; $i -le 50; $i++) {
    prosodyctl register ship-$i localhost password-planet-$i
}
```

si vous êtes sous windows powershell.

Une fois cela fait, nous avons donc un serveur qui tourne, et tous les agents que nous utiliserons inscrits. Pour cela, nous allons définir un agent spade de base capable de récupérer des messages et d'en envoyer. Cet agent sera utilisé comme un composant de l'agent mesa.

Notez que les actions des agents sont représentés par des `Behaviour`, qui peuvent être au moins `OneShot` (c'est à dire lancés une unique fois) ou `Periodic` c'est à dire lancés à intervalles réguliers (ici 0.01s). Lorsque l'agent est lancé, on ajoute aux comportements de l'agent son `PeriodicBehaviour` de réception. Ce dernier va recevoir les messages et les mettre dans la liste de messages correspondant à la boîte aux lettres de l'agent. Pour éviter les accès concurrents, on utilisera un **mutex** ou mutual exclusion.

Les agents capables de communiquer auront donc en commun de pouvoir envoyer des messages et d'avoir un "composant" qui est en fait un agent spade. On construit donc une classe qui correspond à ces agents, `CommunicatingAgent`.

On peut alors définir les agents capables de communiquer, le `PlanetManager`. Cet agent génère à temps régulier un nouvel `Item` qu'il faut livrer à une autre planète. Il génère alors un *call for proposal* pour transporter cet item, qu'il transmet à tous les `Ship` se trouvant sur sa planète.

Le second type d'agents est le `Ship` qui est capable de se déplacer sur le `SpaceRoadNetwork`, et qui pour le moment affiche les messages qu'il a reçus. Notez que les comportements de déplacement ont déjà été implémentés pour vous. De manière à raccourcir le TP, certains apects d'adaptation à l'environnement ont déjà été implémentés. C'est en particulier le cas de l'adaptation aux changements du `SpaceRoadNetwork` que vous devrez implémenter par la suite. C'est aussi cet agent qui livre les `Item` et ajoute donc une unité au `model_reporter` du modèle.

Interaction et Organisation

Pour le moment, tout est statique car le seul message utilisé est un *call for proposal* envoyé par les `PlanetManager`, qui est uniquement reçu et affiché par le `Ship`. Dans cette partie, nous allons

faire en sorte que le TP fonctionne, en particulier que les `Item` soient pris en charge par le `Ship` qui les préfère. Pour ce faire, nous allons mettre en place un *Contract Net Protocol* parcellaire, en laissant de côté tout ce qui se passe après l'`accept_proposal` ou le `reject_proposal`. Commencez par observer l'`AgentCommunicator`. Comme nous l'avons vu, cette classe est un agent de spade à laquelle sont ajoutées deux `Behaviour`, l'un qui reçoit, un `CyclicBehaviour` qui tourne toutes les 0.01s, et un qui sert à l'envoi, qui est un `OneShotBehaviour`.

Mettez en place un *Contract Net Protocol* de manière à ce qu'un `Ship` fasse une proposition lorsqu'il reçoit un *call for proposal* (CFP) avec pour corps : la version sérialisée du bien, et l'utilité calculée par le `Ship`. Il doit alors arrêter de répondre aux CFP afin d'éviter de s'engager sur deux `Item` différents, jusqu'à avoir une réponse du `PlanetManager`. Le *proposal* est envoyé au `PlanetManager` qui récolte tous les messages reçus, puis, une fois qu'un certain temps (ici `WAITING_TIME`) est écoulé, il vérifie s'il a des propositions. Si tel est le cas, il choisit le `Ship` ayant la meilleure utilité et lui envoie un `accept_proposal` et un `reject_proposal` aux autres. Sinon, il relance un `call_for_proposal` vers un autre `PlanetManager` et remet le bien dans `items_to_ship`. Lorsqu'un `Ship` reçoit un `accept_proposal`, il récupère l'item et le ramène.

Question 3– Lancez la simulation. Comment évoluent le rapport entre le nombre de biens présents dans le système et le nombre de biens livrés, en fonction du nombre de planètes et du nombre de vaisseaux. Ajoutez une image de la courbe.

Question 4– * De quelle organisation s'agit-il ? Justifiez votre réponse

Environnement dynamique

La dernière étape du TP est de modifier l'environnement pour le rendre dynamique. En effet, comme sur nos routes, il arrive que les autoroutes de l'espace soient encombrées... par des météorites. Dans ce TP, on va considérer que ces autoroutes spatiales peuvent être dans trois états : en parfait état (leur état actuel, auquel cas les agents vont à leur vitesse maximale), encombrés (auquel cas il vont à mi-vitesse par rapport à leur vitesse maximale et s'affcihent en rouge) ou complètement impraticables (dans ce cas, les agents sont totalement immobilisés et la route ne doit pas être affichée). Notez que ces modifications sont déjà prévues pour l'agent `Ship`, inutile de le modifier. Modifiez le code pour prendre en compte ces changements. Un changement de statut intervient avec une probabilité `PROBA_ISSUE_ROAD`. S'il y a un changement de statut, la route a 50% de chance de passer dans chacun des états dans lequel il n'était pas : **Question 5–** Caractérisez l'environnement du système comme nous l'avons vu en cours. Justifiez.

Question 6– Comment les agents s'adaptent-ils à ce changement d'environnement ? À quelle caractéristique fondamentale des agents cela correspond-il ?

Question 7– Lancez la simulation. Quelle différence observez-vous avec la situation précédente ?

Question 8– (Bonus) Ajoutez le `ROAD_BRANCHING_FACTOR` aux paramètres à faire varier et étudier l'influence de ce facteur sur le nombre de biens livrés.

Si certains éléments du framework mesa vous ont posé problème, merci de l'indiquer à la fin du fichier `reponses.md`