Projet "Systèmes multi-agents"

Modèles "producteur/consommateur" distribués

Hong-phuc VU

Yasmine GUEDJOU

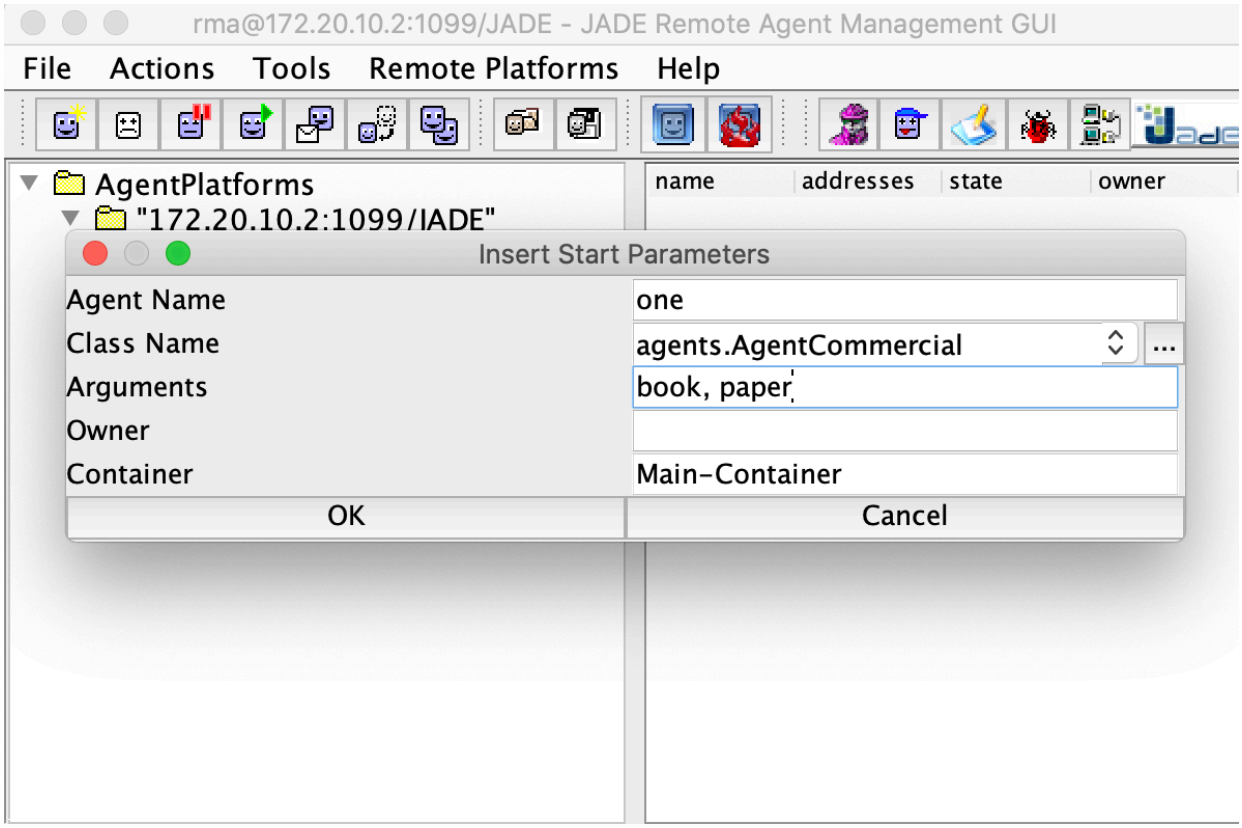
décembre 2019

Université de Cergy-Pontoise

Manuel d'utilisation

Pour faire fonctionner le programme :

1. Créer des agents de la classe AgentCommercial
2. Mettre en paramètres (dans le GUI) <marchandise à vendre> <marchandise à consommer> : book/paper/ink. Comme c'est montré dans la figure suivante :

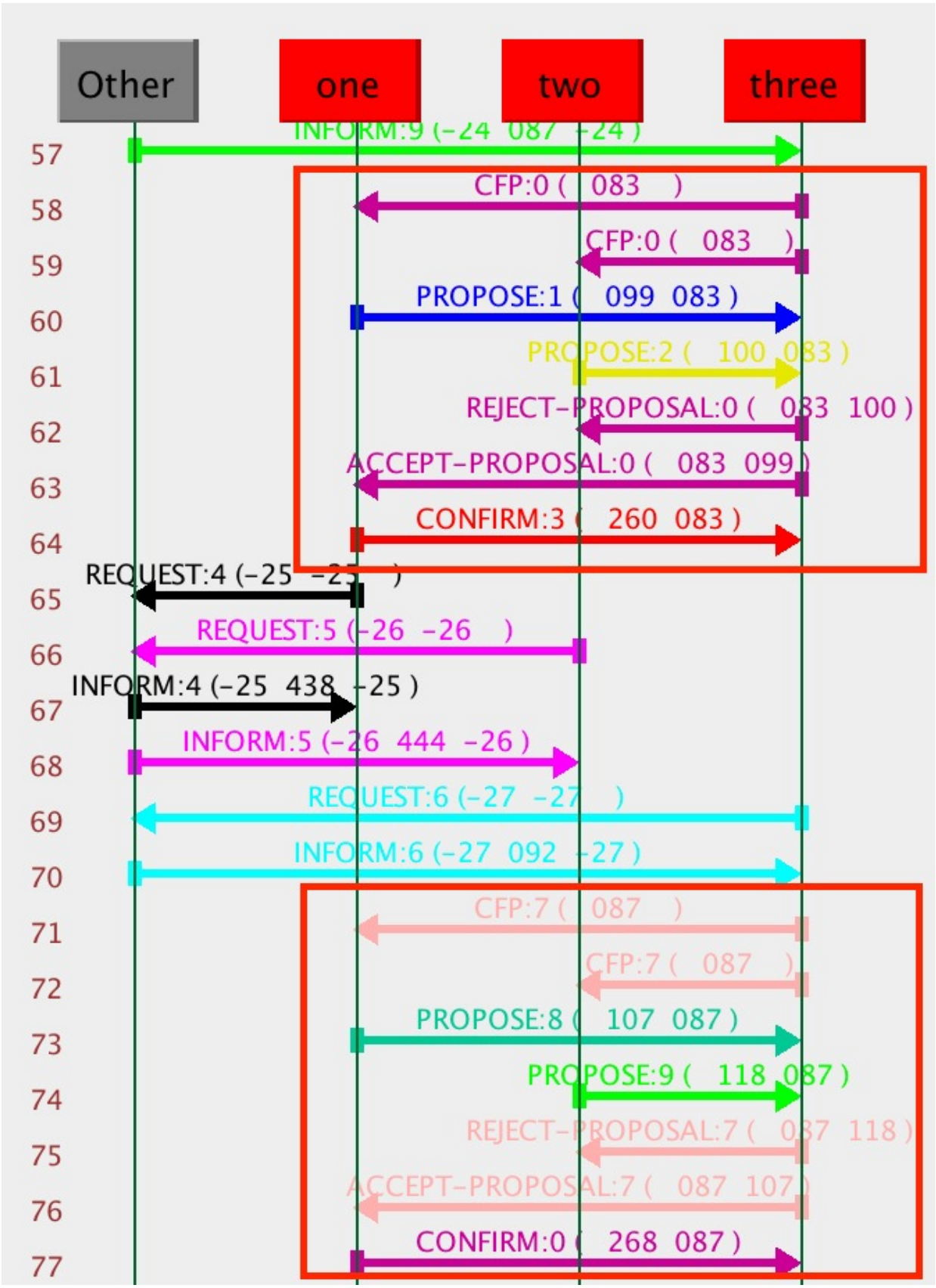


Si on ne choisit pas les arguments, le programme va choisir automatiquement entre les troix choix.

Diagramme et résultats de la solution

Rythmes de programmation

- Production : 1 s
- Consommation de produit : 1 s
- Lancement d'un achat d'un produit à consommer : 1 s
- Satisfaction et prix de produit à vendre: 1 s



Dans l'image ci-dessus :

1. L'agent **one**: cherche *paper* et vend *ink*
2. L'agent **two** : cherche *paper,* vend *ink*
3. Agent **three** : cherche *ink* vend *book*

Changement de prix

Dans l'image ci-dessous : L'agent one propose la marchandise à 1$ alors que agent 2 propose à 1.25. Le système choisit le prix le moins cher donc effectue la transaction avec l'agent one. On peut également remarquer que money (argent) de l'agent one passe de 117 à 119 => cela signifie qu'il y a deux produits vendus.

```
[one] Money: 117.0, satisfaction: 94.39788366471394, Price: 1.0
[one] stockAvailableProduct: 2, stockConsumed: 0
[three] Avg Satisfaction: 100.0
[three] Money: 72.0, satisfaction: 100.0, Price: 1.0
[three] stockAvailableProduct: 3, stockConsumed: 37
Dec 15, 2019 10:33:06 PM behaviours.SellerBehaviours$OfferRequestsServer action
INFO: [two]Reçu --> (CFP) d'agent: three
Dec 15, 2019 10:33:06 PM behaviours.SellerBehaviours$OfferRequestsServer action
INFO: [one]Reçu --> (CFP) d'agent: three
Dec 15, 2019 10:33:06 PM behaviours.TransactionBehaviours handleProposal
INFO: [three]Reçu --> (PROPOSE 1 1.25): d'agent: two
Dec 15, 2019 10:33:06 PM behaviours.TransactionBehaviours handleProposal
INFO: [three]Reçu --> (PROPOSE 2 1.0): d'agent: one
Dec 15, 2019 10:33:06 PM behaviours.TransactionBehaviours rejectProposals
INFO: [three]Envoyer REJECT_PROPOSAL à l'agent: two
Dec 15, 2019 10:33:06 PM behaviours.TransactionBehaviours acceptProposal
INFO: [three]Envoyer ACCEPT_PROPOSAL à l'agent: one
Dec 15, 2019 10:33:06 PM behaviours.SellerBehaviours$SellBehaviours action
INFO: [one]Reçu --> (ACCEPT_PROPOSAL 2): d'agent: three
Dec 15, 2019 10:33:06 PM behaviours.SellerBehaviours sendConfirm
INFO: [one]Reçu --> (Transaction valide! Send Confirm!): de :three
Dec 15, 2019 10:33:06 PM behaviours.SellerBehaviours sendConfirm
INFO: [one]Vendre 2 book à 1.0/unité à three
Dec 15, 2019 10:33:06 PM behaviours.TransactionBehaviours buyMerch
INFO: [three]Reçu --> (CONFIRM 2 1.0): d'agent: one
Dec 15, 2019 10:33:06 PM behaviours.TransactionBehaviours buyMerch
INFO: [three]Acheter 2 book pour 1.0/unité de one
[two] Avg Satisfaction: 99.96616900648968
[two] Money: 111.0, satisfaction: 99.56019708436574, Price: 1.5
[two] stockAvailableProduct: 1, stockConsumed: 0
[one] Avg Satisfaction: 98.71766715090028
[one] Money: 119.0, satisfaction: 92.86282065545873, Price: 1.0
[one] stockAvailableProduct: 1, stockConsumed: 0
[three] Avg Satisfaction: 100.0
[three] Money: 70.0, satisfaction: 100.0, Price: 1.0
```

Dans l'image suivante, on remarque la décroissance exponentiellement de la satisfaction : de 74.8445 -> 69.4866 -> 63.0812 -> .. etc -> mort.

```
[one] Money: 36.25, satisfaction: 91.02764012728643, Price: 1.0
[one] stockAvailableProduct: 50, stockConsumed: 0
[one] Avg Satisfaction: 99.35513591221783
[one] Money: 36.25, satisfaction: 88.83366662217487, Price: 1.0
[one] stockAvailableProduct: 50, stockConsumed: 0
[one] Avg Satisfaction: 99.17257504044532
[one] Money: 36.25, satisfaction: 86.21075314459706, Price: 0.95
[one] stockAvailableProduct: 50, stockConsumed: 0
[one] Avg Satisfaction: 98.95206083963137
[one] Money: 36.25, satisfaction: 83.07503838102724, Price: 0.9025
[one] stockAvailableProduct: 50, stockConsumed: 0
[one] Avg Satisfaction: 98.68684738824858
[one] Money: 36.25, satisfaction: 79.32626543730437, Price: 0.8573749999999999
[one] stockAvailableProduct: 50, stockConsumed: 0
[one] Avg Satisfaction: 98.36895044129815
[one] Money: 36.25, satisfaction: 74.84457636696631, Price: 0.6859
[one] stockAvailableProduct: 50, stockConsumed: 0
```

Quand l'argent < 50 et stock à consommer = 0 et satisfaction < 90 alors le prix est baissé.

Simulation d'un modèle multi-agents "Producteur/consommateur " avec Jade

Afin de définir les actions d'un agent en fonction du temps et de son environnement, il est nécessaire d'attribuer des comportements aux agents JADE. Pour qu'un agent exécute une tâche associer à un comportement, il faut lui l'attribuer par la méthode **addBehaviour(Behaviour b)** de la classe **jade.core.Agent**.

Il existe plusieurs types de comportements pour les agents, tous issus de la classe **Behaviour** du Framework. Chacun a un moyen de déclenchement différent. Chaque Behaviour doit implémenter au moins les deux méthodes :

· **action :** qui désigne les opérations à exécuter par le Behaviour;

· **onStart :** qui exprime si le Behaviour a terminé son exécution ou pas.

Behaviour pour la gestion de la vente

La classe SellerBehaviour est un CyclicBehaviour qui gère la vente :

- L'agent propose une offre de vente (OneShotBehaviour)
- L'agent vend son produit si un autre agent répond qu'il vend ce produit (OneShotBehaviour)

Behaviour pour la gestion de l'achat

La Classe TransactionBehaviour, est un *TickerBehaviour* qui gère l'achat :

- L'agent est à la recherche d'un deal pour acheter un certain produit (OneShotBehaviour)
- L'agent va collecter toutes les propositions pendant un laps de temps (WakerBehaviour)
- L'agent va acheter un produit (WakerBehaviour)

Behaviour Agent Commercial

La classe AgenteCommecial gère la satisfaction, le rythme de production & de consommation, et le changement de prix.

- L'agent produit une marchandise périodiquement (TickerBehaviour)
- L'agent consomme une marchandise périodiquement (TickerBehaviour)