Intelligence Artificielle (I-ILIA-026) -- Travaux Pratiques

TP 2 : Application multi-agents pour la réservation intelligente d'hôtels

- **Introduction :**

L'objectif de ce TP est de donner plus d'intelligence à un agent de type « **Agence »** qui va aider les clients à trouver un hôtel selon leurs critères de recherche. Cet agent devrait chercher la meilleure offre parmi plusieurs offres proposées par trois agents du type « **Hôtel »** on se basant sur la demande de l'agent « **Client »** et le meilleur rapport qualité/prix proposé.

Concrètement, l'agent « **Client »** doit envoyer sa demande à un agent « **Agence »** (agent intermédiaire), la demande se présente sous format d'objet qui contient le nombre de personnes, budget par nuits, la ville, le nombre d'étoile demandé (exemple : 3 personnes, 80 Euros, 3 nuitées, Paris, 3 étoiles).

- Chaque hôtel est représenté par un agent indépendant (définie dans un script python séparé), il propose un service en fonction de la demande de l'agent « **Agence »** et dispose des disponibilités variables. Si le nombre de nuits demandé par l'agence (pour satisfaire la demande du client) est supérieure au nombre de nuits disponibles, la demande doit être refusée.
- L'agent « **hôtel »** doit répondre via un message du type **CFP** à l'agent « **Agence »** en donnant une réponse indiquant la disponibilité et cout total. Si le nombre de nuits demandées dépasse 3, les Agents hôtes peuvent proposer une réduction (un taux de 25 30 40 % …). Afin de réaliser les différentes interactions, il faudra utiliser plusieurs actes de communication. Voilà les actes de communication les plus importants :
  - **INFROM** : c'est le type de message le plus utilisé, nous pouvons l'utiliser dans plusieurs contextes comme (envoyer une information, confirmation ou un accusé de réception) ;
  - **REQUEST** : c'est un type de message qui permet de faire une demande initiale. L'agent qui envoie ce message doit recevoir une notification de type INFORM par exemple ;
  - **CFP** : c'est l'abréviation de (Call for Proposal) ce type peut être utilisé pour les messages que les agents envoient pour demander une offre. Ce message doit recevoir comme réponse un message de type PROPOSE ;

- **PROPOSE** : comme son nom l'indique, c'est un message qui doit avoir comme contenu une proposition (soit un prix, un texte qui contient le nom d'un article, etc.). Ce message doit recevoir comme réponse soit (ACCEPT_PROPOSAL ou REFUSE) ;
- **ACCEPT_ PROPOSAL** : pour accepter l'offre et terminer la réservation ;
- **REFUSE** : pour refuser l'offre, l'agent qui reçoit ce type de message aura le droit de proposer une autre offre (PROPOSE). Ce type de message peut être remplacé par **REJECT_PROPOSAL** au cas où l'agent ne veut pas recevoir une nouvelle offre.

Exemples :

- Instruction pour la spécification d'un type de message ACL (d'un acte ***REQUEST***) :

MyMessage = ACLMessage(ACLMessage.REQUEST)

- Instruction pour la vérification si un message reçu représente un **CFP** (Call For Proposal)

if msg.performative == ''cfp'' *(toujours en minuscules)*

- ## Enoncé du TP :

Télécharger le code de démarrage « ***Starting_code_TP2 »*** partagé via Moodle contenant la structure et le squelette du code à compléter pour répondre aux questions de ce TP. La figure 1 illustre le plan des échanges et l'architecture à respecter.

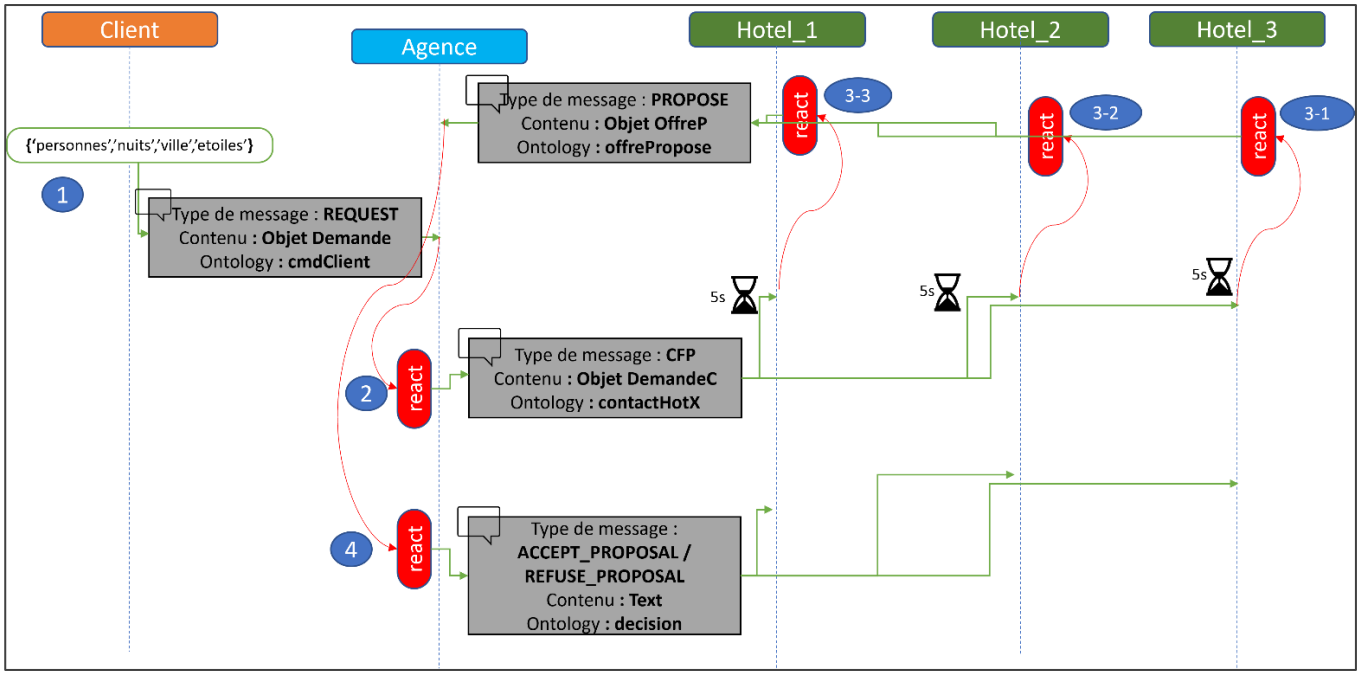

Figure 1 : Les interactions entre les différents agents

- **Question 1 :** compléter le code de **MainAgent.py**, pour démarrer les cinq Agents (Client, Agence, Hotel_1, Hotel_2 et Hotel_3) *(inspirez-vous de précédent)*

- **Question 2 :** En s'inspirant du code de la classe **Client**, compléter le code des classes : **Agence**, **Hotel_1**, **Hotel**_2 et **Hotel_3** afin que chaque agent puisse marquer son démarrage par un message « **bonjour je suis l'agent …** ». Compiler et tester le programme ?

- **Question 3 :** L'agent « **Client »** doit envoyer une demande à l'agent « **Agence »** pour réserver une chambre, en définissant les variables : nombre de personnes, nombre de nuits souhaitées, ville, nombre d'étoiles. Le message doit être envoyé avec une ontologie « *cmdClient »* pour qu'elle soit reconnue par **l'Agence**.

  Compléter le programme pour réaliser cet envoie de commande dans la classe « **Client » ?**

  Note : le message doit avoir un format d'objet, veuillez utiliser pickle comme au TP1.

- **Question 4 :** Une fois la demande du « **Client »** reçue par l'agent **« Agence »**, ce dernier doit transmettre l'objet de la demande aux trois agents « **hôtel »** (créés précédemment).

  Dans la classe « **Agence »**, encodez les trois fonctions permettant de contacter les agents de type « **hôtel »** après réception du message venant du client avec l'ontologie « *cmdClient »* dans la fonction *react*.

  Note : pour bien simuler les interactions, l'envoie des messages doit être fait via le comportement **call_latter**. Cette fonction doit appeler les 3 fonctions successivement avec un intervalle de temps de 3 secondes.

- **Question 5 :** chaque agent « **Hôtel »** reçoit la demande de l'agent « **Agence »** (**CFP**) et proposera en retour son offre s'il existe une chambre disponible.

  Dans les classes **Hotel_1.py**, **Hotel_*2.py*** et **Hotel_*3.py***, compléter le programme pour permettre aux trois agents « **Hôtel** » de réagir à la demande de de l'agent **« Agence »** de type *CFP* et envoyer une réponse à cette dernière selon les disponibilités de chaque agent « **Hôtel »** à l'aide d'un message de type **« PROPOSE »** mentionnant les informations suivantes : nombre de personne accepté dans la chambre, nombre de nuits disponibles, Ville, Nombre d'étoiles et le pourcentage de réduction que l'agence pourra avoir en fonction du nombre de nuit.

  Note : nous considérons ici que chaque hôtel dispose de suffisamment de chambres de même catégorie (ex. chambre double) et de nuitées pour répondre aux demandes des clients. Le prix, le nombre d'étoiles, la ville peuvent être différents d'un hôtel à l'autre.

Remarque :

- Il ne faudra pas oublier de donner un Type **PROPOSE** aux messages envoyés par les agents **Hôtel** pour qu'ils soient reconnus au niveau de l'agent **Agence**.

  Proposition_Hotel = ACLMessage(ACLMessage.REQUEST)

- Il faudra aussi mentionner les ontologies *contactHot1, contactHot2 et contactHot3* dans les messages envoyés par **« Agence »** vers les trois « **Hôtel »**.

➢ **Question 6 :** L'agent « **Agence »** doit sélectionner l'offre selon la demande de l'agent « **Client »**

- Comparer l'offre des hôtels et la demande du client
- Refuser les offres inadéquates (par exemple, refuser un hôtel trop loin) et envoyer un message de type « **REFUSE »** à l'hôtel en question.
- Calculer le prix total de chaque offre d'hôtel non refusée.
- Sélectionner la meilleure offre d'hôtel en fonction du prix et envoyer un message de type ***ACCEPT_ PROPOSAL*** au bon hôtel.

Compléter le programme pour établir l'échange décrit ci-dessus.

➢ **Question 7 :** après sélection de la meilleure offre d'hôtel, l'agent « **Agence »** doit confirmer la réservation au Client concerné. Encoder cette transaction et clôturer la demande (voir Fig. 1).

➢ **Question 8 :** améliorer le code de l'agent « **Agence »** pour tenir compte de l'offre suivante :

- Si le nombre de nuitées demandés par le client est supérieur à 3, l'agent « **l'Agence »** peut appliquer le taux réduction offert par chaque hôtel.

Encoder cette fonctionnalité et tester avec l'exemple suivant :

- **Demande Client :** 2 personnes, 220 Euros, 3 nuitées, Mons, 3 étoiles
- **Offre « Hotel_1 » :** 2 personnes, 60 Euros/nuit, Charleroi, 3 étoiles, 20%
- **Offre « Hotel_2 » :** 2 personnes, 70 Euros/nuit, Mons, 3 étoiles, 20%
- **Offre « Hotel_3 » :** 2 personnes, 80 Euros/nuit, Mons, 3 étoiles, 35%

L'agent « **Agence** » devra donc refuser l'offre du l'«**Hôtel_1 »** en raison de la localisation de son hôtel. L'offre de l'« **Hôtel_2 »** sera **également refusée** car son prix total est de **168 Euros** qui supérieur à celui de l'« **Hôtel_3 »** ayant un prix total de **156 Euros.** La Figure 2 illustre un exemple d'exécution**.**

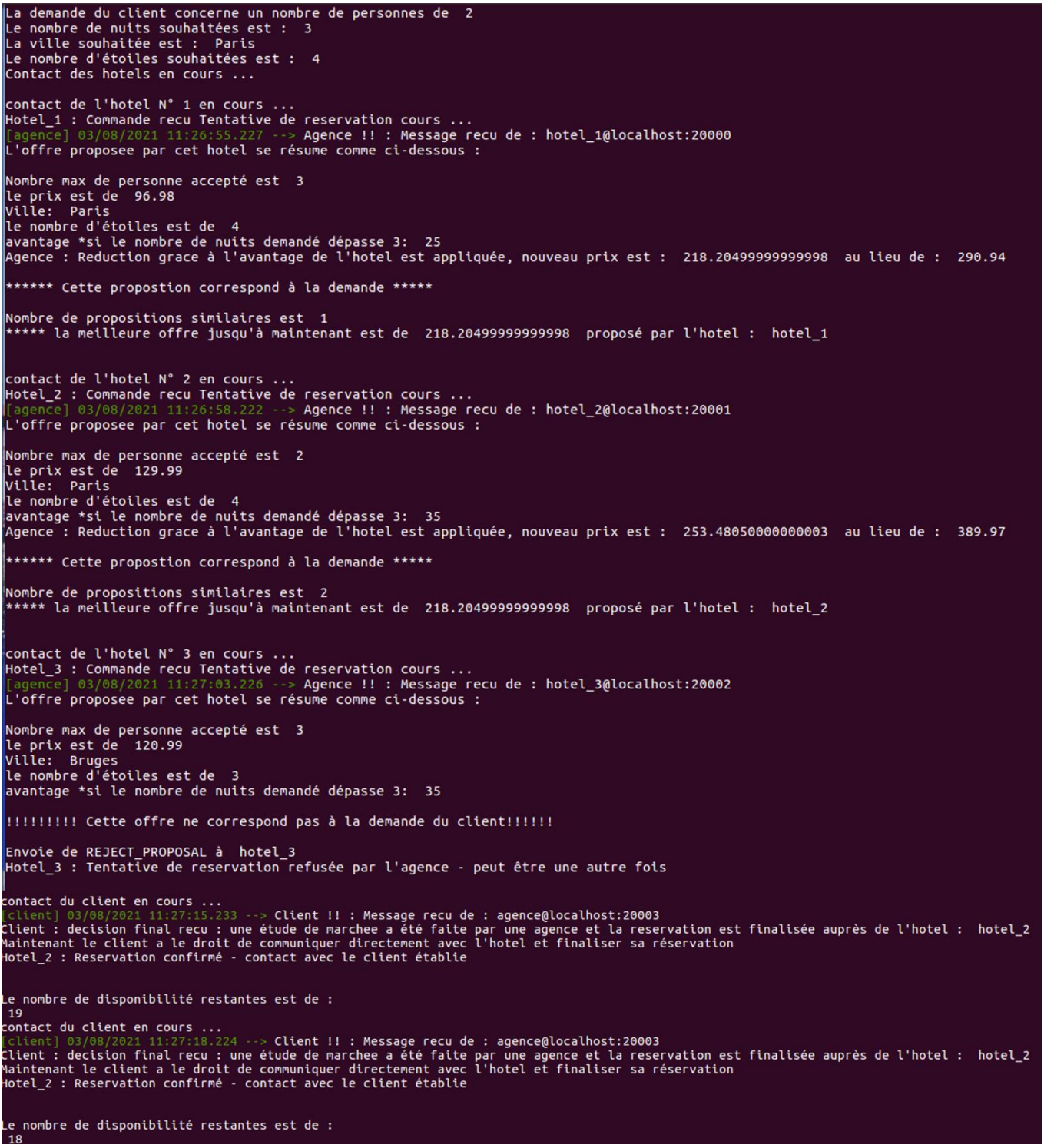

Figure 2 : exemple de mise en œuvre d'échange entre agents

- **Question 9 :** l'objectif ici est proposer un **système de scoring** qui serait proposé par l'agent « **Agence** » qui calculera un score pour chaque agent « **Hotêl** » en fonction de son offre sans devoir passer par la vérification de conditions successives (if else …). La figure 3 illustre la méthode de calcul du score en question ainsi que le tri des offres avant d'informer le client.

Figure 3 : système de Scoring

- Modifier/compléter le code de « **Agence.py** » en suivant les étapes suivantes :
  - **a.** Définir une liste de Scoring
  - **b.** Calculer le score de chaque agent « **Hôtel** » selon le nombre de caractéristiques demandées par le client avec la formule suivante :

    Score_Hôtel = (100/nombre_de_caractéristiques) * (nombre de critères satisfaits)
  - **c.** Trier la liste des agents de type « Hôtel » en ordre décroissant selon le score
  - **d.** Proposer et envoyer la meilleure offre au client.

  Exemple :

  - **Demande :** Pour la demande Client suivante « 2 personnes, 220 Euros, 3 nuitées, Mons, 3 étoiles », nous avons **5** caractéristiques.
  - **Offre Hotel_1 :** pour cette offre d'hôtel « 2 personnes, 60 Euros/nuit, Charleroi, 3 étoiles, 20% » le score de l'hôtel serait calculé comme suit :

    Score_Hôtel_1 = (100/5) * 4 = 80% (4 : nombre de critères satisfaits)

  Note : il serait intéressant d'utiliser la notion des dictionnaires de python afin de lier chaque hôtel à son score :

```
DictHotel = {}
# Ajouter Des élements
DictHotel [message.sender.localname] = UnScore
```

➢ **Question 10 :** pénaliser sur les scores

- Modifier votre programme de manière à pénaliser le score (avec une pénalité de votre choix) si deux hôtels proposent une offre avec le même score mais les prix proposés sont différents. Dans ce cas, l'agent « **Agence** » doit pénaliser l'agent « **Hôtel** » ayant le prix le plus élevé afin de favoriser le meilleur prix.

QUESTIONS facultatives :

- **Question 11 :** après la confirmation de chaque réservation, l'agent « **Agence »** doit enregistrer le nom de l'hôtel avec l'offre proposée. Si un autre client souhaiterait faire une réservation similaire, l'Agent « **Agence** » doit réagir intelligemment et contacter directement le bon agant « **Hôte**l » sans demander aux autres.

- **Question 12 :** dans cette partie, il faudra modifier votre programme pour prendre en considération plus de caractéristiques demandées par le client (type de vue, avis d'utilisateurs, etc.) pour répondre de manière plus précise à la demande des clients. L'agent « Agence » devra prendre en considération ses nouveaux éléments dans le choix d'hôtel.

- **Question 13 :** proposer une interface graphique avec l'outil de votre choix (tkinter, PyQt …) qui permettra à chaque client d'encoder les caractéristiques de sa demande et visualiser la réponse de l'agent sans passer par le terminal.

- **Question 14 :** proposer une amélioration du programme permettant d'offrir plus d'intelligence aux agents en intégrant vos idées personnelles.

Références :

[1] González-Briones, Alfonso, et al. "Multi-agent systems applications in energy optimization problems: A state of-the-art review." *Energies* 11.8 (2018): 1928.

[2] De Freitas, Bruna Kobay, et al. "Exploiting PADE to the Simulation of Multiagent Restoration Actions." *2019 11th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*. IEEE, 2019.

[3] Melo, Lucas Silveira, et al. "Python-based multi-agent platform for application on power grids." *International Transactions on Electrical Energy Systems* 29.6 (2019): e1 2012.