SMA : Projet EDF

Yoann Estepa et Jean Ogier du Terrail

7 janvier 2015

Introduction

Il s'agit dans ce projet de modéliser le comportement de tous les acteurs du marché de l'electricité. A première vue il en existe deux grandes catégories : les clients et les producteurs, mais nous verront que ce modèle même très simple nécessite plusieurs modifications. Il sera aussi l'occasion d'utiliser le framework Jade qui contient toute la structure d'un modèle multi-agents simple ainsi que de tester le logiciel de versioning git très populaire chez les développeurs.

Chapitre 1

Prise en main de Jade et mise en place des éléments de base du marché

1.1 Les Agents et leur contenu

Tous les agents possèdent une méthode setup dans laquelle sont codés des Behaviours qui peuvent être OneShot, Cyclic ou Ticker. En définissant ces behaviours nous determinons complétement le comportement de ces agents et Jade se chargera du reste. Pour l'instant nous avons définis ensemble une structure de conversations et d'interactions entre nos agents sans réellement qu'il y ait de véritable transaction d'argent et de problèmes de dettes ou d'impayés.

1.1.1 ClientAgent

Ce client a besoin de consommer en moyenne meanProduction mais sa consommation dépend beaucoup de ses activités qui ne sont pas toujours les mêmes. A chaque tick d'horloge chaque consommateur tire donc une quantité d'electricité a consommer selon une gaussienne centrée de variance définie. Cette action est définie dans un TickerBehaviour. Pour trouver cette electricité cet agent va regarder dans le DF (et il regarde jusqu'à en trouver un (CyclicBehaviour), sinon il peut y avoir un problème s'il regarde avant l'inscription de tous les producteurs) pour trouver les producteurs disponibles. Il demande les prix des producteurs disponibles (envoie d'un CFP) et reçoit des propositions (voir fournisseur). Comme il est intelligent il prend le moins cher et s'abonne quand il a complété son dialogue. Chaque Client possède en attribut un producteur qu'il initialise avec les bonnes valeurs quand la proposition est acceptée. Une fois qu'il est abonné il reçoit tous les mois (voir horloge) des REQUESTS pour effectuer la transaction électricité contre argent.

1.1.2 Le fournisseur

Le fournisseur possède un prix de vente au kilo qui est public et que tous les clients peuvent voir. Ils ont aussi, heureusement pour eux, une ArrayList de clients qui se remplit au fur et à mesure des souscriptions. Le fournisseur s'inscrit dans le DF (OneShotBehaviour) pour se faire connaitre. Quand il reçoit le CFP le producteur PROPOSE son prix aux clients qui peuvent faire le choix de l'accepter ou non. Justement le producteur attend les performatifs REFUSE ou ACCEPTE. Dans le cas ou sa proposition est acceptée il INFORME le client qu'il est bien abonné. Pour collecter l'argent de ses clients il envoie tous les mois des REQUESTS à l'ensemble de sa liste de clients.

1.1.3 Le dernier agent : l'horloge

Pour rappeler aux autres agents la notion du temps qui passe nous avons définis un agent horloge possédant un comportement Ticker. Cet agent INFORME seulement tous les producteurs qu'il est temps de récolter leur du et donc d'envoyer les REQUESTS. Ces messages aux producteurs se différencient de ceux des clients par l'identifiant de conversation "'top"'. Il a été utile de créer des messageTemplate pour différencier les messages et ne pas dépiler automatiquement des messages. Cela a aussi justifié l'utilisation d'ID de conversations.

1.1.4 Une interface graphique

Nous avons créé une interface graphique sommaire comportant quelques boutons (producteurs, Nouveau Producteur, Nouveau Client). Notre but et de la faire afficher la liste des clients de chaque producteur en cliquant sur le bouton approprié. On remarque que Jade possède déjà une GUI de ce type. Elle est complétée par des logs qui s'affiche lors du processus d'éveil et de souscription. Dans la suite il peut être intéressant de la faire évoluer en lui rajoutant les messages consoles voir de lui donner un aspect plus ludique (barres de consommation dŠélectricité, état des comptes clients, etc.).

Chapitre 2

Ajout d'un agent transporteur et d'un processus de décision chez les fournisseurs

2.1 Fournisseur : modifications

On ajoute chez le fournisseur les attributs suivants : ajout d'un nombre de fournisseurs en activité : nb_transport_perso, le coût fixe associé au transporteur CF, la capacité moyenne d'une telle machine capamoy, le prix de la consommation par kilo associé au passage de l'electricité par un transporteur externe price_TIERS, ainsi que LT une durée en année dont nous verrons l'utilité plus tard.

```
private int LT=3;  //Durée long terme
private int CF=50000; //Cout Fixe de créer une installation
private int capamoy=10; //capacité moyenne d'une telle installation
private int price_TIERS;
private int nb_transport_perso=0;
```

Ces attributs vont lui permettre le moment venu (tous les ans) de prendre une décision quant à l'installation ou non d'un nouveau transport. Décision fondée sur une estimation des demandes de productions mensuelles à venir sur la durée LT à partir d'une statistique sur la consommation moyenne par mois en un an.

2.2 Prise de décision

Il va donc falloir ajouter un nouveau Behaviour au fournisseur. Lors de la reception du message de REQUEST par les clients ils doivent tous répondre

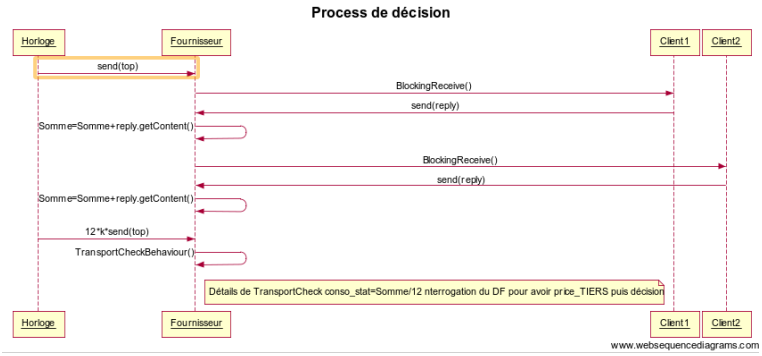

FIGURE 2.1 – Diagramme de séquence simplifié

leurs consommations mensuelles sous la forme d'un message informatif dont on a changé l'ID de conversation à conso et dont le contenu est égal à la consommation mensuelle du client en question (monthlytotal). Pour réaliser une statistique il faut donc à chaque top d'horloge faire la somme des consommations mensuelles de tous les clients abonnés au fournisseur pour connaître la production totale à réaliser. La reception des messages de chaque client du fournisseur est bloquante (BlockingReceive(mt)) car normalement tous les clients devraient répondre à la REQUEST assez vite et nous aimerions avoir la somme réelle et non une somme partielle (message null casté en 0). Puis tous les ans (tous les tops d'horloge dont le contenu est divisible par 12) diviser cette somme par 12 on obtient conso_stat. On a créé une classe TransportCheckBehaviour dans laquelle ce comportement est implémenté et qui est invoqué tous les ans. Dans cette méthode nous avons aussi réalisé un appel au DF pour connaître le prix du transporteur externe et du coup donner sa valeur à notre price_TIERS. Tous les éléments sont maintenant prêts pour permettre la décision. Cela est résumé dans le diagramme suivant :

Il s'agit de comparer la valeur :

```
deltat=(myFournisseur.getCF()/
(Math.min(myFournisseur.getCapamoy(),conso_stat)*
(myFournisseur.getPrice_TIERS())));
```

à LT c'est à dire si la perte sur LT année d'utiliser le transporteur extérieur au prix tiers est supérieure au coût fixe. Dans ce cas il l'achète. Nous avons commencé à gérer le capital du transporteur afin de prendre en compte ses finances. En créant le capital et en connaissant la somme consommée par mois il est facile de connaître ses dépenses en faisant passer le maximum de consommation par le transporteur car c'est gratuit. Nous attendions la prochaine séance pour finaliser la gestion du portefeuille.

2.3 Avancement de la GUI

Nous nous sommes demandé comment stocker les fournisseurs et clients dans la GUI de manière à les updater de manière régulière sans non plus bloquer le processeur par des communications bloquantes incessantes. Nous avons hésité entre des hashtable ou des ArrayList. La construction de la GUI est en cours.

Conclusion

Après avoir passé beaucoup de temps à comprendre le système de Jade et à mettre en place le processus d'abonnement, il est maintenant plus facile de complexifier les comportements et de rajouter un côté financier à notre marché jusque là fondé sur la confiance.