

# 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'électricité. À 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 déterminons complètement 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’électricité à consommer selon une gaussienne centrée de variance définie. Cette action est définie dans un `TickerBehaviour`. Pour trouver cette électricité 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.

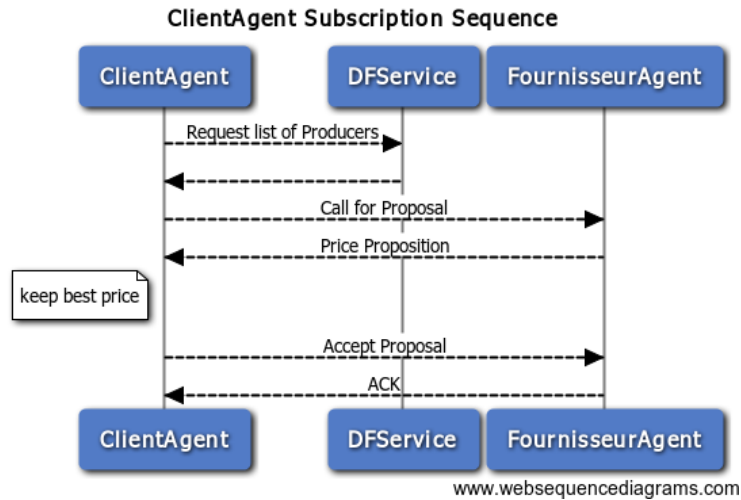


FIGURE 1.1 – Diagramme de séquence simplifié de souscription

### 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 connaître. 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 où 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 défini 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.

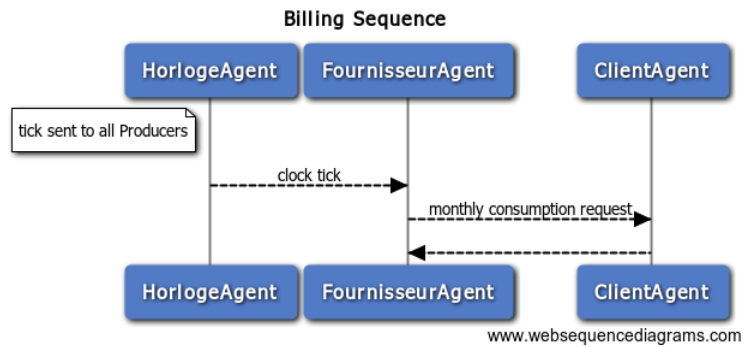


FIGURE 1.2 – Diagramme de séquence simplifié de facturation



#### 1.1.4 Une interface graphique

Nous avons créé une interface graphique sommaire comportant quelques boutons (producteurs, Nouveau Producteur, Nouveau Client). Notre but est de 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 d'électricité, 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'électricité 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 réception du message de `REQUEST` par les clients ils doivent tous répondre

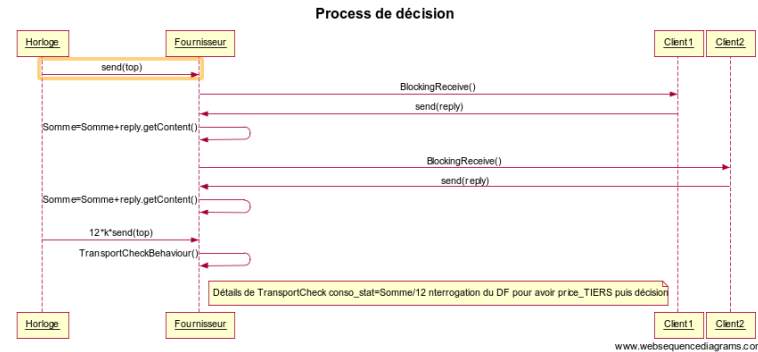


FIGURE 2.1 – Diagramme de séquence simplifié

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.



## Chapitre 3

# Mise en marche de la GUI et introduction de l'argent

### 3.1 La GUI, un agent comme un autre

La GUI possède peu d'attributs et se comprend simplement. Elle possède une hashtable d'un nouveau type `DataProducer`, qui est défini dans la classe ainsi que les clés correspondantes, à savoir les AID des producteurs. Son setup est défini comme les autres, elle s'enregistre dans le DF pour que les agents puisse l'identifier et lui envoyer des messages. En dehors de son setup elle possède un `CyclicBehaviour`, qui remplit la table avec un nombre de `DataProducer` représentant les fournisseurs, qui ont répondu à l'appel. Ces `DataProducers` ont le nombre d'attributs que l'on veut afficher sur l'interface graphique. Nous avons choisi d'afficher pour chaque fournisseur :

- son nom
- son nombre de clients
- sa production mensuelle et totale
- son capital
- son nombre de transports personnels

Ces valeurs sont updatées tous les mois grâce à l'utilisation du comportement `EnvoiGUI` par les Fournisseurs eux-mêmes.

```
public class EnvoiGUI extends Behaviour{
private String champ;
private double valeur;
private boolean foundGUI;

public EnvoiGUI(String champ, double valeur){
this.champ = champ;
this.valeur = valeur;
}
```

```

@Override
public void action() {
    //contacter le DFService pour obtenir l'AID de la GUI
    DFAgentDescription template = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType("gui");
    template.addServices(sd);
    try{
        DFAgentDescription[] results = DFService.search(myAgent, template);

        //on garde ce comportement tant que l'on obtient pas de GUI (ie elle n'est pas encore souscrite)
        if(results.length != 0){
            foundGUI = true;
            AID gui = results[0].getName();

            //envoi de la nouvelle production mensuelle Ã la GUI
            ACLMessage inf = new ACLMessage(ACLMessage.INFORM);
            inf.addReceiver(gui);
            //on se sert du conversationID pour passer le champ Ã MaJ
            inf.setConversationId(champ);
            inf.setContent(String.valueOf(valeur));

            myAgent.send(inf);

            //log
            System.out.println("Producteur " + myAgent.getLocalName() + " a envoyÃ une nouvelle valeur " + valeur);
        }
        catch(FIPAException e){
            e.printStackTrace();
        }
    }

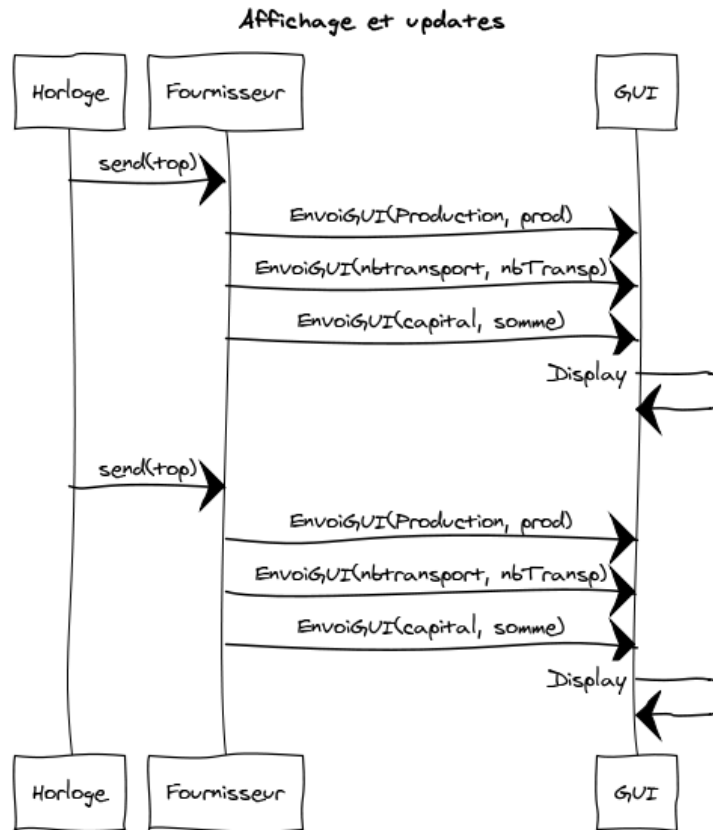
}

@Override
public boolean done() {
    // TODO Auto-generated method stub

    return foundGUI;
}
}

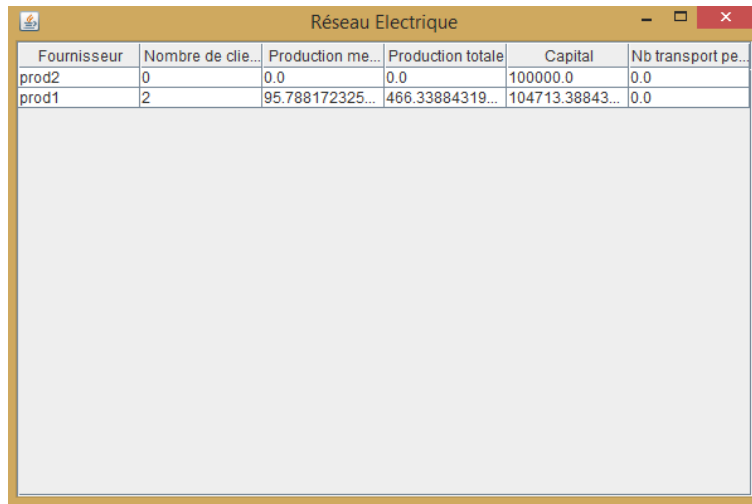
```

Ce comportement est appelé dans celui de Facturation. On peut résumer par :



[www.websequencediagrams.com](http://www.websequencediagrams.com) Voici une

photo de notre interface graphique haute en couleur :



Fournisseur	Nombre de client	Production moyenne	Production totale	Capital	Nb transport personnel
prod2	0	0.0	0.0	100000.0	0.0
prod1	2	95.788172325...	466.33884319...	104713.38843...	0.0

### 3.2 Introduction du capital

On a déjà vu le processus de décision qui amène à faire l'acquisition ou non d'un transport personnel. Il est intéressant de voir enfin son effet sur le portefeuille des fournisseurs. C'est une lourde dépense mais qui sera bien remboursée par la suite si les clients continuent à affluer et à consommer. C'est pourquoi en partant d'un capital de 100 000. On peut voir à toutes les étapes de facturation ce qui se passe pour les fournisseurs :

- le paiement de la mise en marche des centrales
- le paiement des transporteurs tiers pour l'acheminement de l'électricité aux clients, dont le prix est recherché par l'appel au DF
- la réception du paiement des clients pour se rembourser

Sachant que le prix des transporteurs est proportionnel au volume circulant il est intéressant d'en faire passer une partie dans les transports personnels des fournisseurs.



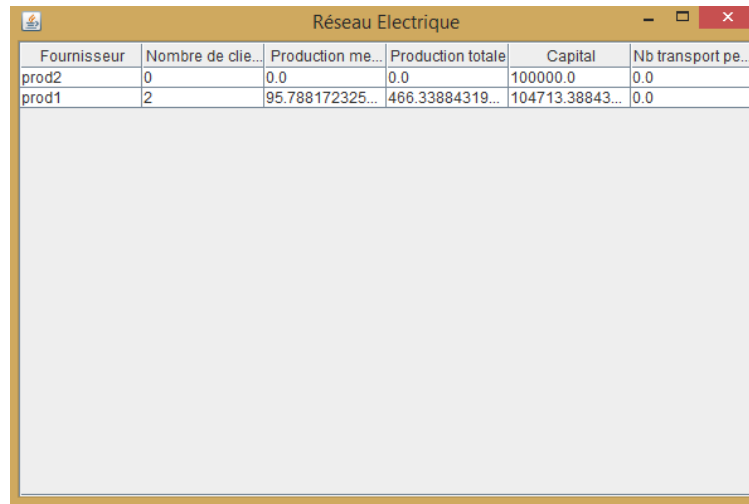
### 3.3 Comparaison des comportements des agents

On peut voir l'effet d'un comportement intelligent comme le notre sur le long terme, vis à vis de plusieurs autres comportements moins optimaux.

Il y a le fournisseur borné qui n'a pas envie de s'encombrer de transport personnels : son bénéfice ne s'envole pas car même si le nombre de ses clients augmentent il aura toujours plus à payer en transport, on peut même imaginer de le facturer de manière supplémentaire au delà d'un certain nombre d'électricité à transporter dans une démarche de régulation des flux.

Il y a au contraire le fournisseur voulant trop bien faire qui s'endette en achetant à tous les tours des transports personnels alors qu'il n'en a pas besoin, celui là fait faillite très vite si il n'a pas accès à énormément de clients.

Il y a le fournisseur, qui s'en tient à la chance et sa bonne étoile. Tous les ans il lance un dé si le résultat est compris entre 1 et 4 il achète un transporteur sinon non. Son comportement étant difficile à prédire la différence ne se voit que sur

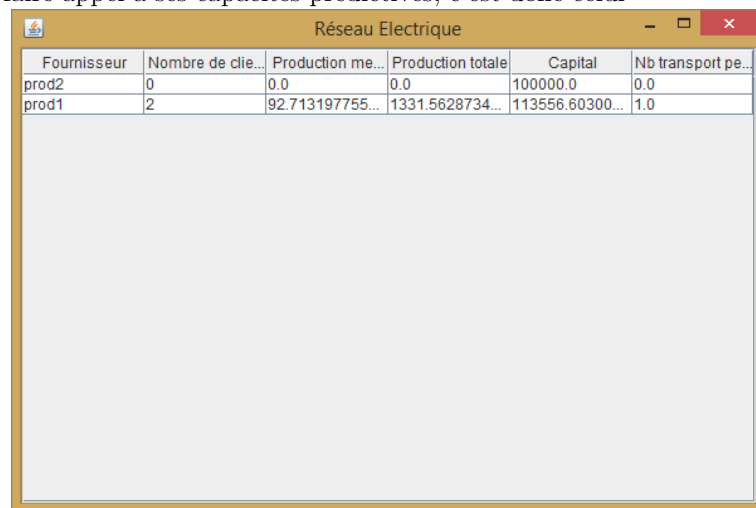


Fournisseur	Nombre de cli...	Production me...	Production totale	Capital	Nb transport pe...
prod2	0	0.0	0.0	100000.0	0.0
prod1	2	95.788172325...	466.33884319...	104713.38843...	0.0

le long terme.

Enfin il y a notre fournisseur intelligent qui sait qu'il ne vaut le coup d'acheter un transport personnel que s'il a suffisamment de clients pour rentabiliser.

Ce fournisseur a juste su faire appel à ses capacités prédictives, c'est donc celui



Fournisseur	Nombre de cli...	Production me...	Production totale	Capital	Nb transport pe...
prod2	0	0.0	0.0	100000.0	0.0
prod1	2	92.713197755...	1331.5628734...	113556.60300...	1.0

qui gagne le plus d'argent.

Cependant on peut se demander ce qui arriverait si des avaries se développaient de manière aléatoire sur ses transports personnels.

## Conclusion

Maintenant que la structure de base est posée il serait intéressant d'introduire du hasard ainsi qu'un marché plus large, pour voir véritablement de la concurrence ou de la coopération les fournisseurs pourraient modifier légèrement leurs prix, quitte à ramasser plus de clients à perte pour les ferrer puis de monter ses prix. Ce petit projet aura quand même été l'occasion de pas mal de

prise de tête et de travail pour aboutir à ce résultat. L'interface aurait bien eut besoin d'un petit lifting pour rendre ce marché plus accessible.