

# RAPPORT DÉVELOPPEMENT D'UNE PLATEFORME E- COMMERCE

Projet Développement d'une Plateforme e-Commerce

Realisé par:

AMGHAR HIND

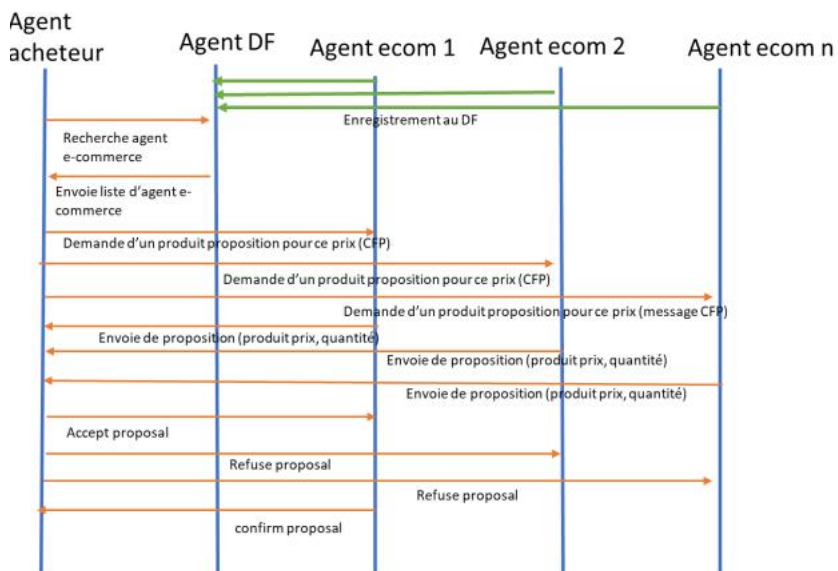
Encadré par:

Pr. BENABBOU FAOUZIA

# Introduction

Le but de ce projet est de réaliser un comparateur de prix dans le cadre d'une Plateforme de e-commerce à base de système multi-agent.

Le fonctionnement simplifié du process est montré dans la figure suivante :



---

Dans la réalisation de cet atelier j'ai utilise plusieurs concepts:

- **Java** est un langage de programmation et une plate-forme informatique qui ont été créés par Sun Microsystems en 1995.

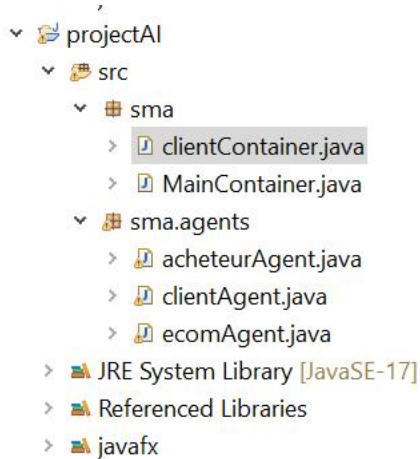
Beaucoup d'applications et de sites Web ne fonctionnent pas si Java n'est pas installé et leur nombre ne cesse de croître chaque jour. Java est rapide, sécurisé et fiable.

- **Jade** est une plateforme de programmation multi-agent implémentée en **Java**. Les agents qui tournent sous JADE communiquent via le langage *Agent Communication Language* ou ACL.
- **JavaFX** définit l'API relative au langage **FXML** qui permet de décrire une interface utilisateur d'une manière alternative à l'écriture \*\*de lignes de code.
- **Eclipse** est un environnement de développement intégré (IDE) populaire, utilisé principalement pour le développement logiciel en Java et d'autres langages. Il offre une interface extensible, des fonctionnalités avancées telles que le débogage et la gestion de versions, et permet l'ajout de plugins pour personnaliser ses capacités. Sa communauté active et son caractère open source en font un outil polyvalent et largement adopté par les développeurs.
- **Système multi-agent (SMA)** est un système composé d'un ensemble d'agents (un processus, un robot, un être humain, etc.), situés dans un certain environnement et interagissant selon certaines relations. Un agent est une entité caractérisée par le fait qu'elle est, au moins partiellement, autonome.

# Solution

On a réalisé notre solution en utilisant le **Eclipse** et la bibliothèque **JavaFX** pour les interfaces graphiques.

Structure du projet:



---

Le projet est constitué de plusieurs fichiers et bibliothèques: Jre, JavaFX, Jade.

## Les classes java:

- **MainContainer**: contient le traitement pour lancer le conteneur principale ( rma, df et ams).

```

package sma;

import jade.core.Profile;

public class MainContainer {
    public static void main(String[] args) {
        try {
            Runtime runtime = Runtime.instance();
            Properties properties = new ExtendedProperties();
            properties.setProperty(Profile.GUI, "true");
            Profile profile = new ProfileImpl(properties);
            AgentContainer mainContainer = runtime.createAgentContainer(profile);
            mainContainer.start();
        } catch (ControllerException e) {
            e.printStackTrace();
        }
    }
}

```

- **clientContainer** : lancer les agents **client**, **ecom**, **acheteur** et contient le code pour l'interface graphique qui herite de la classe **Application** pour utilise les methodes de **JavaFX**, pour la creation d'un **GUI**.

Agent Interface

Produit :

Prix :

Quantite :

Date de debut :

Date de fin :

```
private static String[] products = {  
    "{nomDeProduit=Lipstick, quantite=4, startDate=05/01/2024, endDate=07/01/2024, prix=500}",  
    "{nomDeProduit=Lipstick, quantite=6, startDate=09/01/2024, endDate=13/01/2024, prix=600}",  
    "{nomDeProduit=Lipstick, quantite=7, startDate=05/01/2024, endDate=07/01/2024, prix=700}"  
};
```

**products** est un table de type String de tous les produits ( les offres) **ecomAgent**.

Puis on fait une iteration sur la variable **products** pour la creation des agents **ecomAgent** different.

```
int i = 0;
for (String argument : products) {
    AgentController controller = agentContainer.createNewAgent("ecom[" + i++ + "]", ecomAgent.class.getName(),
        new Object[]{
            argument
        });
    controller.start();
}
```

- **ecomAgent**: contient le traitement de l'agent **ecom**,

Dans la fonction setup() :

- Registre **Ecom** dans **df**

```
public void registerEcomInDF() {
    DFAgentDescription description = new DFAgentDescription();
    description.setName(getAID());
    ServiceDescription serviceDescription = new ServiceDescription();
    //serviceDescription.setType("ecom");
    //serviceDescription.setName("ecom");
    serviceDescription.setType("Commerce-en-Ligne");
    serviceDescription.setName("Commerce en Ligne");
    description.addServices(serviceDescription);
    try {
        DFService.register(this, description);
    } catch (FIPAException e) {
        e.printStackTrace();
    }
}
```

- Ajouter un comportement cyclique pour envoyer et recevoir les messages des agents **acheteurAgent**.

```

    this.registerEcomInDF();
    this.addBehaviour(new CyclicBehaviour() {
        @Override
        public void action() {
            MessageTemplate messageTemplate = MessageTemplate.or(MessageTemplate.MatchPerformative(ACLMessage.CFP),
                MessageTemplate.or(MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL),
                    MessageTemplate.MatchPerformative(ACLMessage.REJECT_PROPOSAL)));
            ACLMessage message = this.myAgent.receive(messageTemplate);

            if (message != null) {
                switch (message.getPerformative()) {
                    case ACLMessage.CFP:
                        if (message.getContent().equals("Demande des offres disponible")) {
                            ACLMessage reply = message.createReply();
                            reply.setPerformative(ACLMessage.PROPOSE);
                            reply.setContent(offre);
                            this.myAgent.send(reply); }
                        break;
                    case ACLMessage.ACCEPT_PROPOSAL:
                        ACLMessage reply = message.createReply();
                        reply.setPerformative(ACLMessage.CONFIRM);
                        this.myAgent.send(reply);
                        break;
                    case ACLMessage.REJECT_PROPOSAL:
                        break;
                }
            } else {
                this.block();
            }
        }
    });
}

```

```

@Override
public void takeDown() {
    try {
        DFService.deregister(this);
        System.out.println("Agent est fini: " + this.getAID().getName());
    } catch (FIPAException e) {
        e.printStackTrace();
    }
}
}

```

- **clientAgent** : agent gui ou le client, herite de la classe **GuiAgent** pour s'interagir avec l'interface graphique on utilisant la methode **onGuiEvent**.



```

@Override
public void onGuiEvent(GuiEvent guiEvent) {
    if (guiEvent.getType() == 1) {
        Map<String, String> offreCherche = new HashMap<>();
        offreCherche.put("produit", guiEvent.getParameter(0).toString());
        offreCherche.put("prix", guiEvent.getParameter(1).toString());
        offreCherche.put("quantite", guiEvent.getParameter(2).toString());
        offreCherche.put("startDate", guiEvent.getParameter(3).toString());
        offreCherche.put("endDate", guiEvent.getParameter(4).toString());

        // Envoyer le message de type REQUEST
        this.sendRequestMessage(offreCherche);
    }
}
}

```

Dans methode setup():

- registre **clientAgent** avec le conteneur client principale

```

protected void setup() {
    gui = (clientContainer) getArguments()[0];
    gui.setAgentinterface(this);
}

```

- registre **clientAgent** dans df

```

public void registreClientinDF() {
    DFAgentDescription description = new DFAgentDescription();

    description.setName(getAID());
    ServiceDescription serviceDescription = new ServiceDescription();
    serviceDescription.setType("transaction");
    serviceDescription.setName("client");
    description.addServices(serviceDescription);

    try {
        DFService.register(this, description);
    } catch (FIPAException e) {
        e.printStackTrace();
    }
}

```

- Commander ()

```
public DFAgentDescription[] commander() {
    DFAgentDescription ecomDescription = new DFAgentDescription();
    ServiceDescription ecomServiceDescription = new ServiceDescription();
    ecomServiceDescription.setType("acheteur");
    ecomServiceDescription.setName("acheteur");
    ecomDescription.addServices(ecomServiceDescription);

    DFAgentDescription[] result = null;
    try {
        result = DFService.search(this, ecomDescription);
    } catch (FIPAException e) {
        e.printStackTrace();
    }

    return result;
}
```

- lancement du comportement cyclique

```
// Interaction avec l'agent acheteur
this.addBehaviour(new CyclicBehaviour() {
    @Override
    public void action() {
        MessageTemplate messageTemplate = MessageTemplate.MatchPerformative(ACLMMessage.INFORM);

        ACLMessage message = this.myAgent.receive(messageTemplate);

        if (message != null) {
            if (message.getPerformative() == ACLMessage.INFORM) {
                String content = message.getContent();

                Platform.runLater(new Runnable() {
                    @Override
                    public void run() {
                        if (content.length() == 0) {
                            gui.show("Pas d'offre trouve!");
                        } else {
                            gui.show("Meilleur offre trouve : " + content);
                            String confirmedPrice = message.getContent();
                        }
                    }
                });
            }
        } else {
            this.block();
        }
    }
});
```

- **acheteurAgent**: contient le traitement de l'agent acheteurAgent, et dans la methode setup():
  - Registre dans **DF**
  - Sauvgarder le nombre des **ecomAgent** trouve
  - Lancer le comportement cyclique avec **clientAgent** et avec **ecomAgent** simultanement.

```

@Override
protected void setup() {
    this.registreAcheteurinDF();
    this.result = this.getEcomListFromDF();
    this.nombreEcom = this.result.length;
    this.addBehaviour(clientAgent);
    this.addBehaviour(ecomAgent);
    System.out.println("Agent est démarré: " + this.getAID().getName());
}

private void registreAcheteurinDF() {
    DFAgentDescription description = new DFAgentDescription();
    description.setName(getAID());
    ServiceDescription serviceDescription = new ServiceDescription();
    serviceDescription.setType("acheteur");
    serviceDescription.setName("acheteur");
    description.addServices(serviceDescription);
    try {
        DFService.register(this, description);
    } catch (FIPAException e) {
        e.printStackTrace();
    }
}

```

```

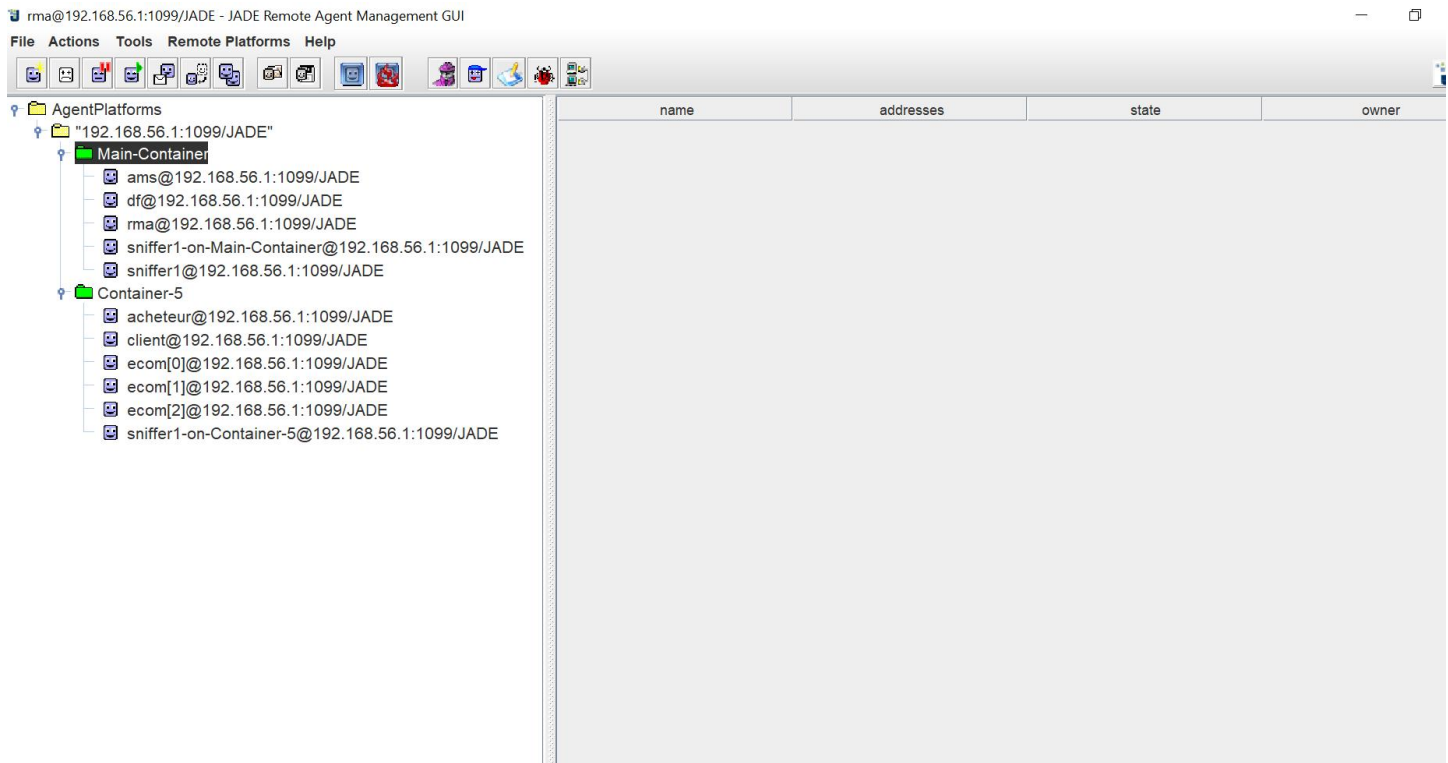
CyclicBehaviour clientAgent = new CyclicBehaviour() {

    @Override
    public void action() {
        MessageTemplate messageTemplate = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        ACLMessage message = this.myAgent.receive(messageTemplate);
        if (message != null) {
            if (message.getPerformative() == ACLMessage.REQUEST) {
                produitAChercher = stringToHashMap(message.getContent());
                client = message.getSender();
                sendCFPMMessage(result);
            }
        } else {
            this.block();
        }
    }
};

```

# Sniffer

- Lacement du conteneur principal et des agents:



- le cas de **ACCEPT\_PROPOSAL**

Agent Interface

Produit : lipstick

Prix : 500

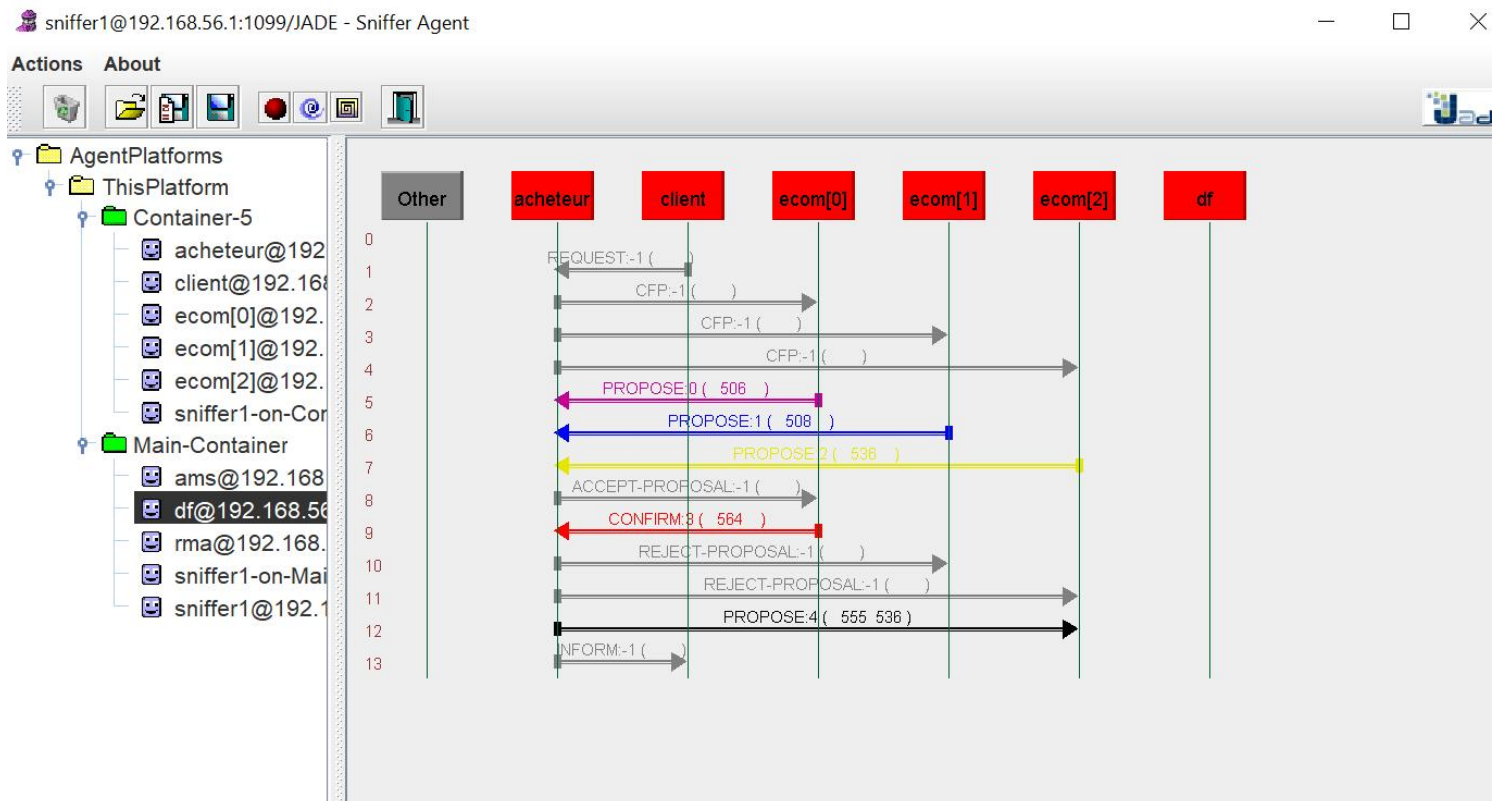
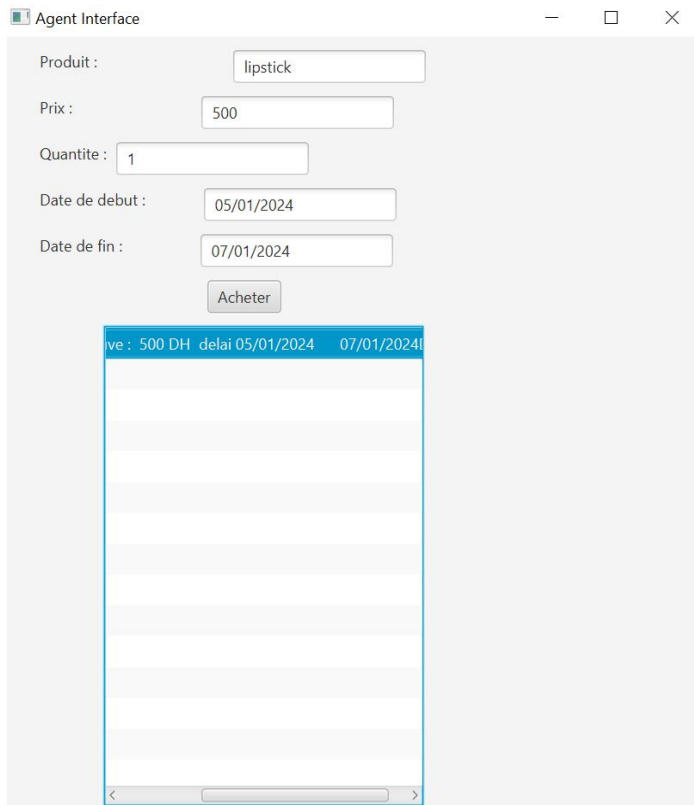
Quantite : 1

Date de debut : 05/01/2024

Date de fin : 07/01/2024

Acheter

ve : 500 DH delai 05/01/2024 07/01/2024





- le cas de **REJECT\_PROPOSAL**

Agent Interface

Produit :

Prix :

Quantite :

Date de debut :

Date de fin :

Pas d'offre trouve!

Meilleur offre trouve : 500 DH delai 05/01/2

