République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Informatique

Département d'Intelligence Artificielle et Sciences des Données

Mini-project : Multi-agent systems, Mobile Agents

Module : Technologie des Agents

Réalisé par :

TAOURIRT HAMZA

SAADA Samir

Année Universitaire :

2023/2024

Table des matières

1 General Introduction

This report presents the mini-project undertaken as part of the Agents Technology module. The project focuses on the development and application of Multi-Agent Systems (MAS) through practical scenarios involving agent-based negotiation and multi-criteria decision-making.

Multi-Agent Negotiation: One Seller and Multiple Buyers

The first part of the project involves simulating an auction scenario where a single seller interacts with multiple buyers (a minimum of two). The auction process is structured as follows: the seller lists an item (product) for sale and sets an initial asking price (opening price). Buyers then place bids that exceed the initial asking price. The seller communicates the highest bid received to all buyers. These steps are repeated until all buyers cease bidding or the auction time expires. If the final bid surpasses a reserve price (known only to the seller), the item is sold to the highest bidder.

This simulation aims to demonstrate the competitive dynamics and negotiation strategies within a multi-agent environment, highlighting the interactions between autonomous agents in a bidding context.

Multi-Criteria Decision-Making and Mobile Agents: One Buyer and Multiple Sellers

The second part of the project involves implementing exercise 5 from SW4, focusing on multi-criteria decision-making. In this scenario, a buyer agent interacts with multiple sellers, with the unique requirement that the buyer agent operates as a mobile agent capable of inter-container and inter-platform migration. This component of the project emphasizes the flexibility and adaptability of agents in a distributed system, showcasing their ability to move across different environments to achieve their objectives.

2 Multi-Agent System (Part 1)

2.1 The JADE Platform

In this part of our project, we chose to use the JADE platform in Java for managing agents in the multi-agent auction system.

JADE (Java Agent DEvelopment Framework) provides a robust and flexible infrastructure for creating, deploying, and communicating between autonomous agents. This platform allowed us to effectively and reliably model the agents and implement the auction mechanisms.

The system enables the seller and buyer agents to participate in an auction for the purchase of a specific product. We developed three main classes: "Seller" and "Buyer", which represent the seller agent and the buyer agents, respectively, and the main class MultiAgentAuction.

2.2 The Seller Class

The Seller class, denoted as SellerAgent, orchestrates the auction process within the multi-agent system. Initially, upon setup, it defines essential parameters such as the opening price, reserve price, and initializes data structures to track bids and buyer information. Through its StartAuctionBehaviour, the agent manages the auction rounds, informing buyers of the current price and processing bids within a defined timeout. It dynamically updates the auction price based on received bids, ensuring transparency and competitiveness throughout the auction process.

Throughout the auction, the "Seller" class maintains communication with buyer agents, providing updates on the auction's progress, current price, and informing them of the auction's conclusion. By enforcing termination conditions, such as maximum rounds or meeting the reserve price, the seller agent ensures a structured and fair auction environment. Ultimately, it concludes the auction by determining the winner and the final price, facilitating a seamless interaction between seller and buyer agents in the multi-agent auction system.

2.3 The Buyer Class

The BuyerAgent class encapsulates the behavior of buyer agents in the auction system. Upon initialization, each agent is endowed with a randomly assigned budget, defining the financial scope of their bidding activities. The class employs a cyclic behavior named "BidBehaviour" to manage the bidding process iteratively. Within this behavior, agents dynamically formulate bids based on the current auction price and their budget constraints. These bids are communicated to the seller agent for consideration. The "BidBehaviour" cyclic behavior facilitates continual adaptation of bidding strategies, ensuring agents remain competitive while adhering to their financial limitations.

Throughout the auction, buyer agents actively engage in the bidding process, striving to secure desired items within their budgetary constraints. If the auction concludes or the agent's budget is exceeded, graceful termination ensures adherence to predefined constraints and fosters a structured auction environment.

2.4 The Main Class

The "Main" class, denoted as MultiAgentAuction, acts as the control center for the auction system, orchestrating the creation and coordination of seller and buyer agents. It dynamically determines the number of buyer agents to be involved in the auction, typically ranging from four to six. Using the JADE platform, it instantiates a main container to host the auction agents and sequentially creates and starts the seller and buyer agents within this container. Utilizing synchronization mechanisms like countdown latch, it ensures all agents are properly initialized before the auction begins. Additionally, it provides methods to retrieve crucial information about the auction environment, such as the number of buyer agents and their AIDs, while also facilitating the maintenance of consistency during auction execution. Overall, the "MultiAgentAuction" class serves as the vital nexus, ensuring the smooth functioning and coordination of agents in the auction system.

3 Functioning of the Multi-Agent Auction System

The multi-agent auction system operates through an iterative process involving several stages. These stages are executed sequentially until a termination condition is met.

3.1. Agent Initialization and Registration

In the MultiAgentAuction class, the auction system is initialized, and the agents, including both buyers and sellers, are created. Using JADE, a Java-based agent development framework, agents are instantiated within a ContainerController, facilitating agent management. The number of buyers is randomly determined, and each agent is registered in the Directory Facilitator (DF), a key component enabling agent discovery and communication. DF serves as a registry where agents advertise their services and locate other agents, crucial for efficient coordination during the auction process.

3.2. Auction Initiation by Seller

The auction begins with the seller agent initiating the auction process. Utilizing a cyclic behavior, the seller announces the item to be auctioned, along with the starting price and any relevant details, to all registered buyers. This announcement is disseminated via ACL (Agent Communication Language) messages, ensuring communication among agents in the system. By leveraging JADE's messaging capabilities, the seller effectively broadcasts auction details, initiating the bidding phase.

3.3. Bidding Phase

During the bidding phase, buyer agents engage in iterative rounds of bidding. Each buyer agent employs a cyclic behavior, continuously listening for bid announcements from the seller and responding accordingly. The cyclic behavior enables agents to repetitively execute bidding actions, ensuring timely participation in the auction process. Upon receiving bid announcements, buyer agents calculate their bids based on predefined formulas :

```
double bid = currentPrice + Math.random() * (budget - currentPrice)*threshold;
```

incorporating factors such as the current price set by the seller and the buyer's budget constraints.

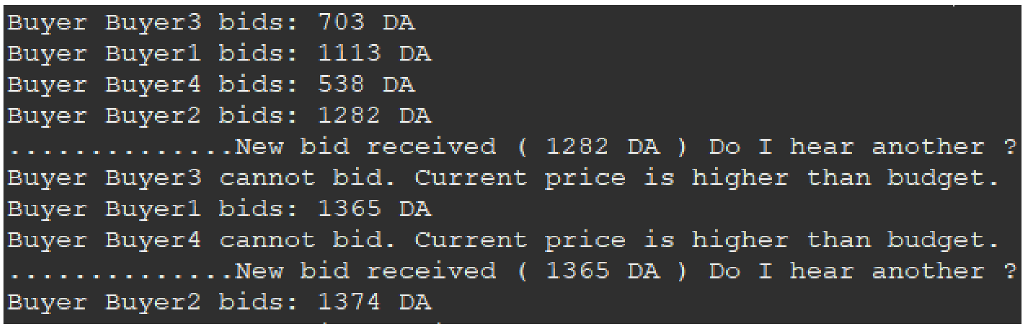
3.4. Bid Calculation and Submission

Buyer agents calculate their bids programmatically, employing mathematical formulas to determine bid amounts dynamically. Once calculated, bids are encapsulated in ACL messages and sent to the seller agent for consideration. To ensure bid integrity, ACL messages contain bid information, including bid amounts and buyer identifiers, facilitating bid processing by the seller agent. Bids are submitted asynchronously, allowing agents to continue participating in the auction while awaiting bid outcomes.

```
// Create and send bid message to the seller agent
ACLMessage bidMessage = new ACLMessage(ACLMessage.INFORM);
bidMessage.addReceiver(new jade.core.AID("Seller",
jade.core.AID.ISLOCALNAME));
bidMessage.setContent(String.valueOf(bid));
send(bidMessage);
```

3.5. Bid Handling by Seller

The seller agent employs JADE's messaging functionalities to handle bid reception and parsing. When a bid message is received, the seller extracts the bid amount and the buyer's AID (Agent Identifier). The seller then updates the current price based on the highest bid received. Crucially, the seller does not send the updated price to the buyer who placed the highest bid. This approach prevents the highest bidder from outbidding themselves in their next immediate round.



```
Buyer Buyer3 bids: 703 DA
Buyer Buyer1 bids: 1113 DA
Buyer Buyer4 bids: 538 DA
Buyer Buyer2 bids: 1282 DA
..............New bid received ( 1282 DA ) Do I hear another ?
Buyer Buyer3 cannot bid. Current price is higher than budget.
Buyer Buyer1 bids: 1365 DA
Buyer Buyer4 cannot bid. Current price is higher than budget.
..............New bid received ( 1365 DA ) Do I hear another ?
Buyer Buyer2 bids: 1374 DA
```

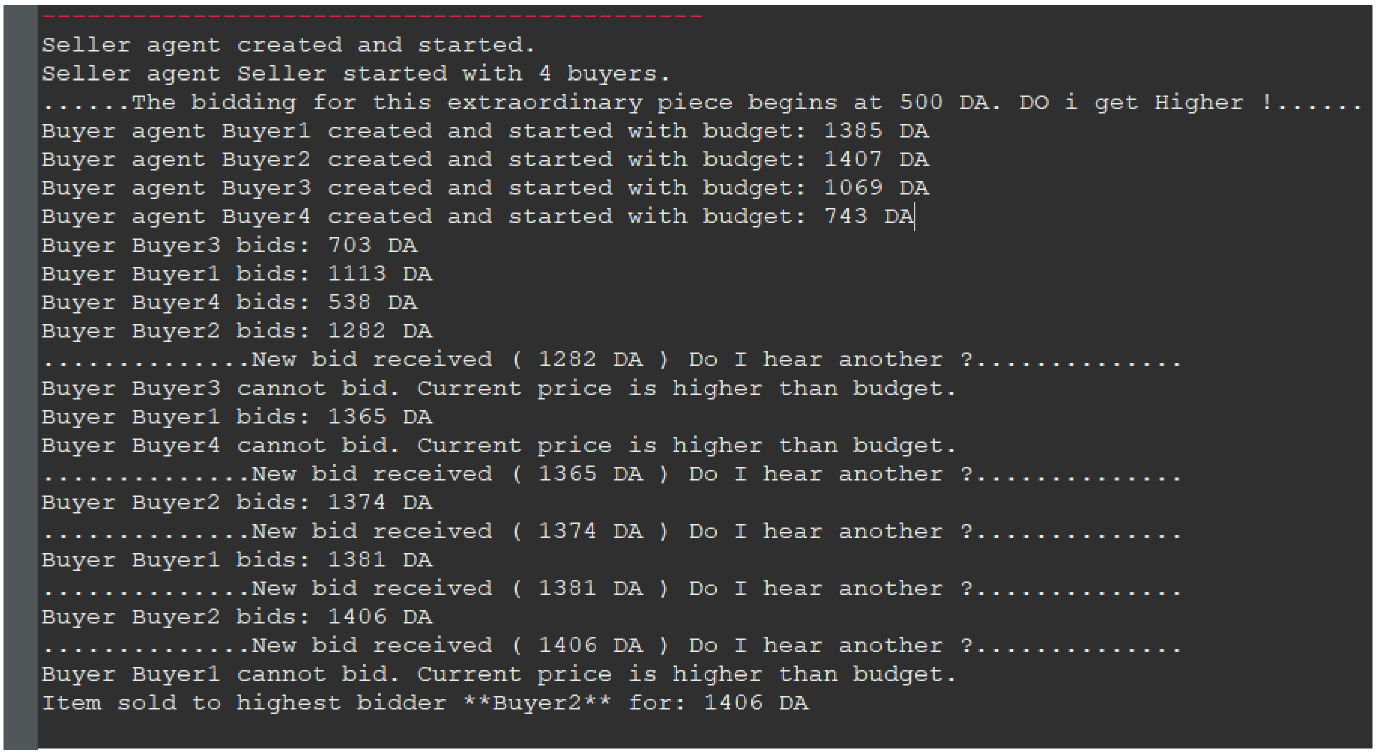
We can clearly see that the agent 2 got the highest bid, in his next immediate round he was excluded (didn’t receive the updated price), so he goes in block mode, till a new message from the seller, but ensuring that he comes for the new round.

3.6. Auction Termination and Outcome Determination

The auction continues through multiple bidding rounds until termination conditions are met. Termination conditions, enforced using blocking mechanisms, define criteria for auction conclusion, including maximum round limits or bid cessation. The seller agent orchestrates termination logic, monitoring bid reception and evaluating termination criteria iteratively. Once termination conditions are satisfied, the seller announces the auction outcome via ACL messages, conveying whether the item was sold and to whom. This conclusive communication marks the end of the auction, providing closure to participating agents and concluding auction proceedings.

3.7. Execution of the code

The figure below illustrates the execution scenario of the multi-agent auction system using JADE, implemented in Eclipse and initiated by running the main class:



```
-------------------------------------------
Seller agent created and started.
Seller agent Seller started with 4 buyers.
......The bidding for this extraordinary piece begins at 500 DA. DO i get Higher !......
Buyer agent Buyer1 created and started with budget: 1385 DA
Buyer agent Buyer2 created and started with budget: 1407 DA
Buyer agent Buyer3 created and started with budget: 1069 DA
Buyer agent Buyer4 created and started with budget: 743 DA
Buyer Buyer3 bids: 703 DA
Buyer Buyer1 bids: 1113 DA
Buyer Buyer4 bids: 538 DA
Buyer Buyer2 bids: 1282 DA
..............New bid received ( 1282 DA ) Do I hear another ?.............
Buyer Buyer3 cannot bid. Current price is higher than budget.
Buyer Buyer1 bids: 1365 DA
Buyer Buyer4 cannot bid. Current price is higher than budget.
..............New bid received ( 1365 DA ) Do I hear another ?.............
Buyer Buyer2 bids: 1374 DA
..............New bid received ( 1374 DA ) Do I hear another ?.............
Buyer Buyer1 bids: 1381 DA
..............New bid received ( 1381 DA ) Do I hear another ?.............
Buyer Buyer2 bids: 1406 DA
..............New bid received ( 1406 DA ) Do I hear another ?.............
Buyer Buyer1 cannot bid. Current price is higher than budget.
Item sold to highest bidder **Buyer2** for: 1406 DA
```

Figure 1 : Execution of the Multi-Agent Auction System.

4 Multi-Agent System - inter-container migration (Part 2)

To determine the best offer in a sales environment with multiple criteria, the buyer agent can use a weighted sum model. This model allows the buyer to assign different importance (preferences) to each criterion and to normalize the values to ensure comparability. The following formula f(x) can be used :

$$f(x) = \sum_i w_i \cdot N_i(C_i(x))$$

Where:

- $f(x)$ is the overall score for product x.
- $Ci(x)$ represents the value of criterion ii for product x.
- wi is the weight (preference) assigned to criterion i.
- $Ni(Ci(x))$ is the normalized value of criterion ii for product x.

Normalization of Criteria

Normalization is necessary to ensure that all criteria are comparable on the same scale (typically 0 to 1). Different normalization methods are used based on whether the criterion is to be maximized or minimized :

- For criteria to be maximized (e.g., quality, volume):

$$N_i(C_i(x)) = \frac{C_i(x) - C_{i,min}}{C_{i,max} - C_{i,min}}$$

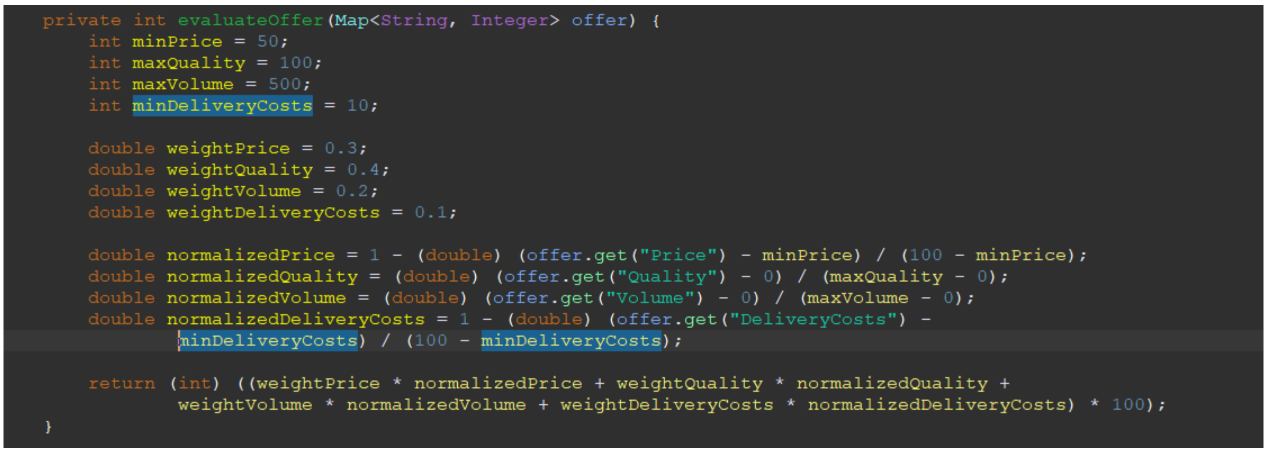
- For criteria to be minimized (e.g., price, delivery costs):

$$N_i(C_i(x)) = \frac{C_{i,max} - C_i(x)}{C_{i,max} - C_{i,min}}$$

Consider a product with the following criteria and weights:

- Price: Minimize, weight $w1 = 0.3$
- Quality: Maximize, weight $w2 = 0.4$
- Volume: Maximize, weight $w3 = 0.2$
- Delivery Costs: Minimize, weight $w4 = 0.1$

Here's the Java implementation of this formula in the context of the provided code:



```java
private int evaluateOffer(Map<String, Integer> offer) {
    int minPrice = 50;
    int maxQuality = 100;
    int maxVolume = 500;
    int minDeliveryCosts = 10;

    double weightPrice = 0.3;
    double weightQuality = 0.4;
    double weightVolume = 0.2;
    double weightDeliveryCosts = 0.1;

    double normalizedPrice = 1 - (double) (offer.get("Price") - minPrice) / (100 - minPrice);
    double normalizedQuality = (double) (offer.get("Quality") - 0) / (maxQuality - 0);
    double normalizedVolume = (double) (offer.get("Volume") - 0) / (maxVolume - 0);
    double normalizedDeliveryCosts = 1 - (double) (offer.get("DeliveryCosts") -
            minDeliveryCosts) / (100 - minDeliveryCosts);

    return (int) ((weightPrice * normalizedPrice + weightQuality * normalizedQuality +
            weightVolume * normalizedVolume + weightDeliveryCosts * normalizedDeliveryCosts) * 100);
}
```

Figure 2 : Java implementation of the used formula.

4.1. The Main class

The MainContainer class is the orchestrator of the multi-agent system (MAS) setup. It initializes the JADE runtime environment, configures the main container (which contains the buyer agent), and dynamically creates containers for seller agents. It then spawns seller agents within these containers before creating the buyer agent. This class ensures the smooth establishment and coordination of the MAS components, facilitating effective agent interaction and negotiation.

4.2. The BuyerAgent class

The BuyerAgent class is a JADE-based agent responsible for interacting with multiple seller agents to solicit, evaluate, and select the best offer. It registers itself with the Directory Facilitator (DF), searches for seller agents, and migrates to their respective containers to request and receive offers. The agent evaluates these offers using a weighted scoring system based on criteria such as price, quality, volume, and delivery costs. After assessing all received offers, it selects the most advantageous one and finalizes the transaction. This ensures a thorough and balanced decision-making process, leveraging JADE's agent mobility and communication capabilities.

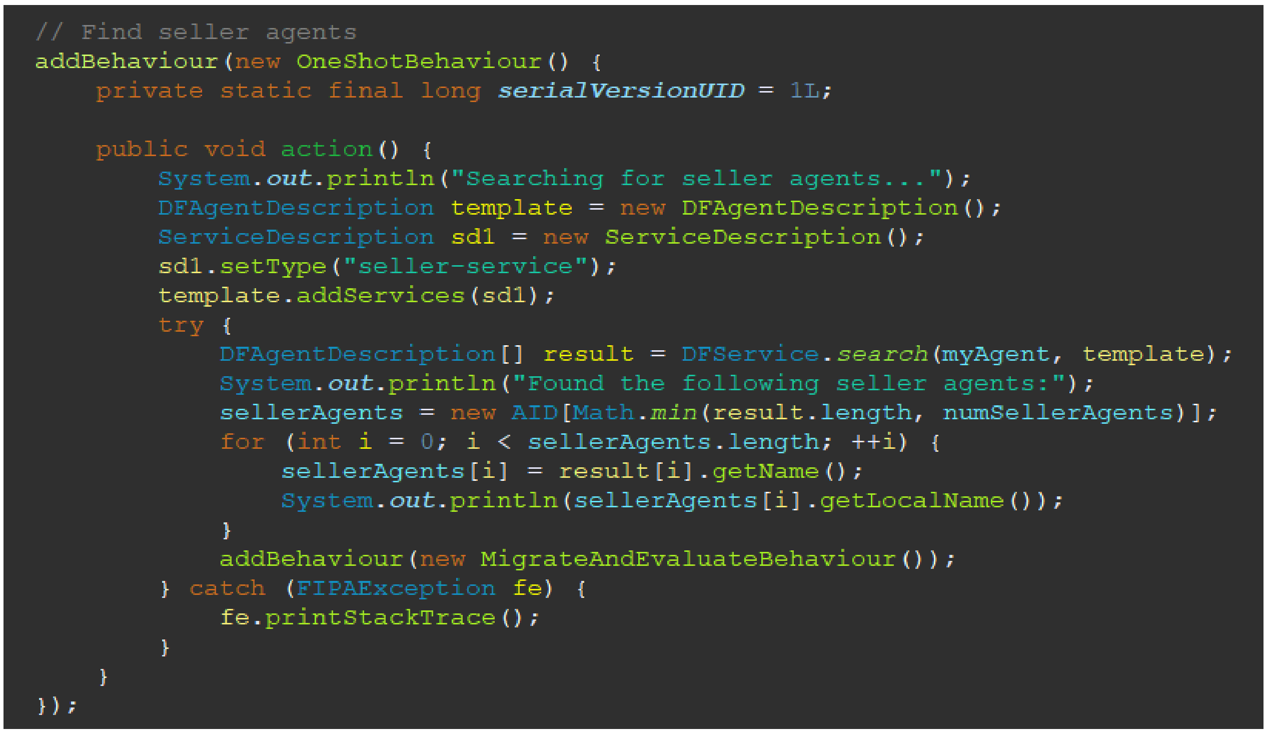
4.3. The SellerAgent class

The SellerAgent class in JADE represents a seller agent that registers itself with the Directory Facilitator (DF) to advertise its selling services. It continuously listens for incoming messages from buyer agents, responding with randomly generated offers that include criteria such as price, quality, volume, and delivery costs. When a buyer agent sends a greeting, the SellerAgent creates a proposal with these details and sends it back to the buyer. The agent's behavior is managed by a cyclic behavior to handle ongoing interactions, and it deregisters itself from the DF when it is taken down. This class encapsulates the seller's functionalities, enabling dynamic and automated offer generation in response to buyer inquiries.

5 Functioning of the - inter-container migration

5.1. Buyer Agent Initialization

The buyer agent initializes by retrieving the number of seller agents it needs to interact with from the provided arguments. If no arguments are given, it defaults to three seller agents. The agent then registers itself with the Directory Facilitator (DF) using a DFAgentDescription and ServiceDescription for the "buyer-service" type, making it discoverable by other agents. The buyer agent uses a OneShotBehaviour to search for seller agents that provide a "seller-service". It creates a template DFAgentDescription and ServiceDescription to find the relevant seller agents registered with the DF. The search results are stored in an array, limited to the specified number of seller agents. As it shown in the next figure :



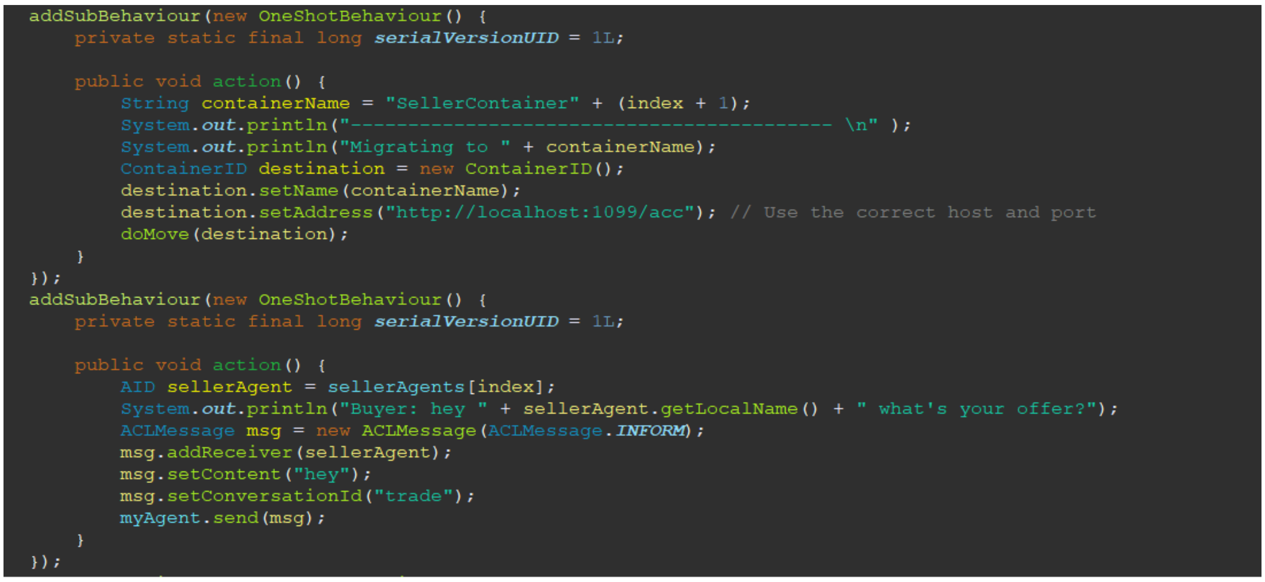
```java
// Find seller agents
addBehaviour(new OneShotBehaviour() {
    private static final long serialVersionUID = 1L;

    public void action() {
        System.out.println("Searching for seller agents...");
        DFAgentDescription template = new DFAgentDescription();
        ServiceDescription sd1 = new ServiceDescription();
        sd1.setType("seller-service");
        template.addServices(sd1);
        try {
            DFAgentDescription[] result = DFService.search(myAgent, template);
            System.out.println("Found the following seller agents:");
            sellerAgents = new AID[Math.min(result.length, numSellerAgents)];
            for (int i = 0; i < sellerAgents.length; ++i) {
                sellerAgents[i] = result[i].getName();
                System.out.println(sellerAgents[i].getLocalName());
            }
            addBehaviour(new MigrateAndEvaluateBehaviour());
        } catch (FIPAException fe) {
            fe.printStackTrace();
        }
    }
});
```

Figure 3 : the implemented search method for sellers.

5.2. Migrating and Sending Messages

The buyer agent adds a SequentialBehaviour called MigrateAndEvaluateBehaviour, which manages the process of migrating to each seller agent's container. For each seller agent, it performs the following sub-behaviours:

- Migration: Uses the doMove method to migrate to the seller agent's container.
- Sending Messages: Sends a greeting message using an ACLMessage with the INFORM performative.
- Receiving Offers: Receives the seller's response using blockingReceive to wait for an ACLMessage with the PROPOSE performative.



```java
addSubBehaviour(new OneShotBehaviour() {
    private static final long serialVersionUID = 1L;

    public void action() {
        String containerName = "SellerContainer" + (index + 1);
        System.out.println("------------------------------------------ \n" );
        System.out.println("Migrating to " + containerName);
        ContainerID destination = new ContainerID();
        destination.setName(containerName);
        destination.setAddress("http://localhost:1099/acc"); // Use the correct host and port
        doMove(destination);
    }
});
addSubBehaviour(new OneShotBehaviour() {
    private static final long serialVersionUID = 1L;

    public void action() {
        AID sellerAgent = sellerAgents[index];
        System.out.println("Buyer: hey " + sellerAgent.getLocalName() + " what's your offer?");
        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
        msg.addReceiver(sellerAgent);
        msg.setContent("hey");
        msg.setConversationId("trade");
        myAgent.send(msg);
    }
});
```

Figure 4 : Migration and sending messages SubBehaviours.

5.3. Generating and Sending Offers

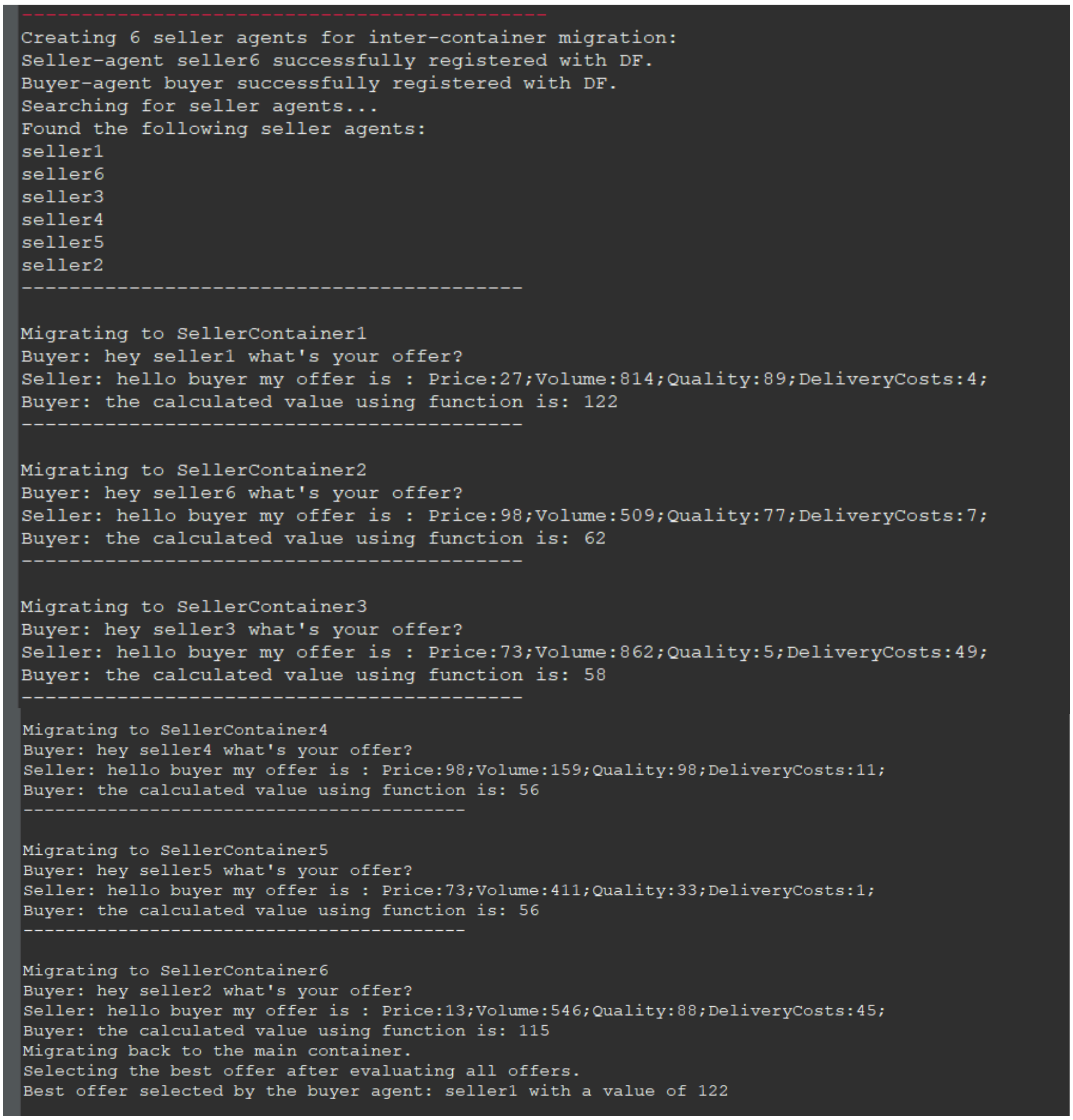
The seller agent initializes by registering itself with the DF using a DFAgentDescription and ServiceDescription for the "seller-service" type, making it discoverable by buyer agents. It then adds a CyclicBehaviour to continuously listen for incoming messages. The seller agent's CyclicBehaviour waits for messages from buyer agents. Upon receiving a greeting message with the INFORM performative, it generates a random offer. This offer, containing criteria such as price, quality, volume, and delivery costs, is sent back to the buyer as a proposal using an ACLMessage with the PROPOSE performative.

5.4. Selecting the Best Offer

The buyer agent evaluates each received offer using a weighted scoring function. This function takes into account various criteria such as price, quality, volume, and delivery costs, each with specific weights and normalization. This ensures a consistent evaluation across all offers.After evaluating all received offers, the buyer agent selects the best one based on the highest calculated value. The details of the best offer are printed out.

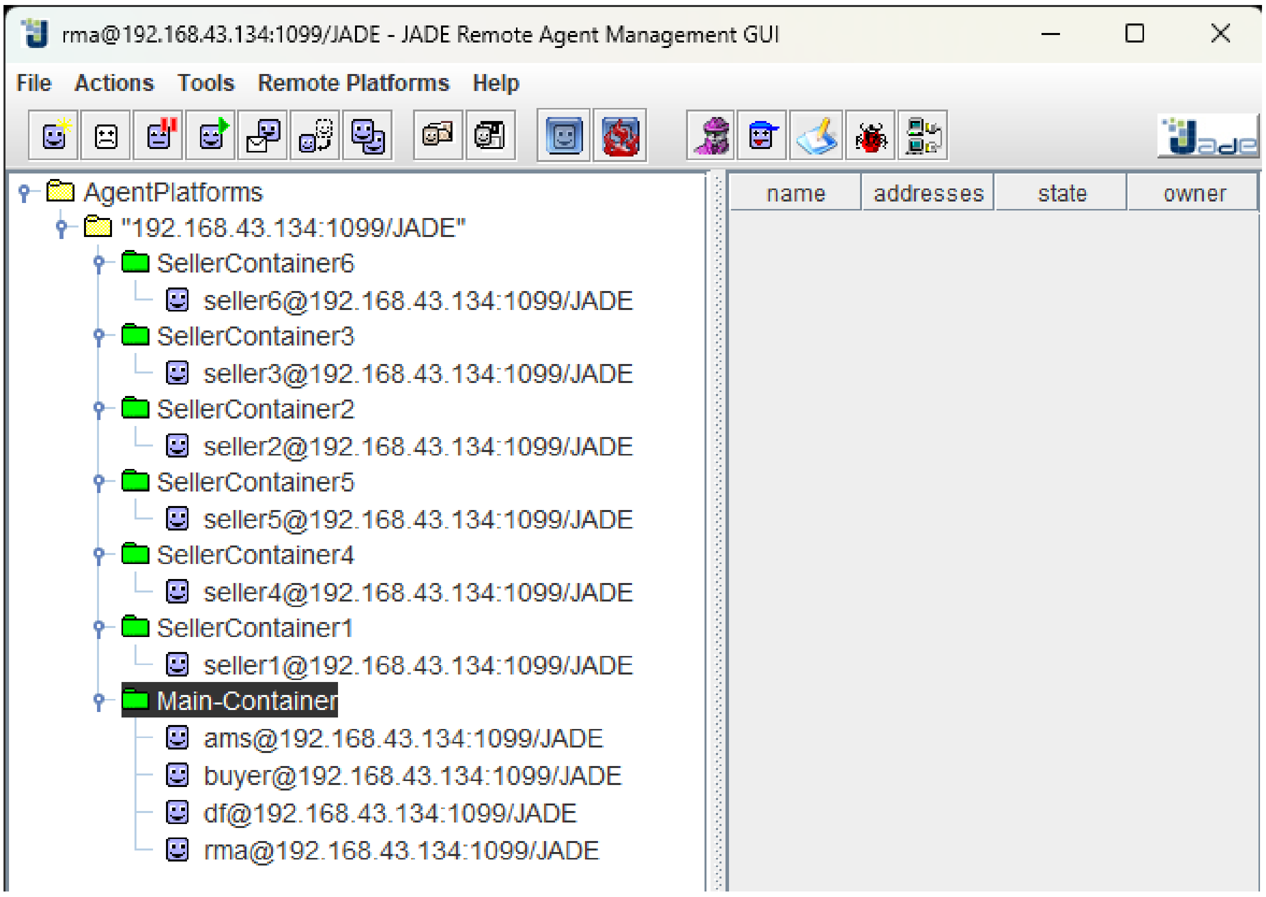
3.8. Execution of the code - inter-container migration (Part 2)

The figure below illustrates the execution scenario of the inter-container migration system using JADE, implemented in Eclipse and initiated by running the main class :



```
--------------------------------------------
Creating 6 seller agents for inter-container migration:
Seller-agent seller6 successfully registered with DF.
Buyer-agent buyer successfully registered with DF.
Searching for seller agents...
Found the following seller agents:
seller1
seller6
seller3
seller4
seller5
seller2
------------------------------------------

Migrating to SellerContainer1
Buyer: hey seller1 what's your offer?
Seller: hello buyer my offer is : Price:27;Volume:814;Quality:89;DeliveryCosts:4;
Buyer: the calculated value using function is: 122
------------------------------------------

Migrating to SellerContainer2
Buyer: hey seller6 what's your offer?
Seller: hello buyer my offer is : Price:98;Volume:509;Quality:77;DeliveryCosts:7;
Buyer: the calculated value using function is: 62
------------------------------------------

Migrating to SellerContainer3
Buyer: hey seller3 what's your offer?
Seller: hello buyer my offer is : Price:73;Volume:862;Quality:5;DeliveryCosts:49;
Buyer: the calculated value using function is: 58
------------------------------------------
Migrating to SellerContainer4
Buyer: hey seller4 what's your offer?
Seller: hello buyer my offer is : Price:98;Volume:159;Quality:98;DeliveryCosts:11;
Buyer: the calculated value using function is: 56
------------------------------------------

Migrating to SellerContainer5
Buyer: hey seller5 what's your offer?
Seller: hello buyer my offer is : Price:73;Volume:411;Quality:33;DeliveryCosts:1;
Buyer: the calculated value using function is: 56
------------------------------------------

Migrating to SellerContainer6
Buyer: hey seller2 what's your offer?
Seller: hello buyer my offer is : Price:13;Volume:546;Quality:88;DeliveryCosts:45;
Buyer: the calculated value using function is: 115
Migrating back to the main container.
Selecting the best offer after evaluating all offers.
Best offer selected by the buyer agent: seller1 with a value of 122
```

Figure 5 : Execution of the inter-container migration System.


rma@192.168.43.134:1099/JADE - JADE Remote Agent Management GUI

File Actions Tools Remote Platforms Help

```
AgentPlatforms
  "192.168.43.134:1099/JADE"
    SellerContainer6
      seller6@192.168.43.134:1099/JADE
    SellerContainer3
      seller3@192.168.43.134:1099/JADE
    SellerContainer2
      seller2@192.168.43.134:1099/JADE
    SellerContainer5
      seller5@192.168.43.134:1099/JADE
    SellerContainer4
      seller4@192.168.43.134:1099/JADE
    SellerContainer1
      seller1@192.168.43.134:1099/JADE
    Main-Container
      ams@192.168.43.134:1099/JADE
      buyer@192.168.43.134:1099/JADE
      df@192.168.43.134:1099/JADE
      rma@192.168.43.134:1099/JADE
```

name	addresses	state	owner

Figure 6 : JADE Platform of the inter-container migration System.

6. Conclusion

The integration of multi-auction MAS and mobile migration across containers represents a sophisticated approach to decentralized decision-making and resource allocation in dynamic environments. By leveraging mobile agent technology, agents can autonomously migrate between containers, enabling seamless communication and collaboration across distributed systems. This mobile migration capability enhances the scalability, flexibility, and fault tolerance of the MAS, allowing agents to adapt to changing conditions and optimize their operations in real-time. Moreover, the integration with multi-auction mechanisms empowers agents to engage in competitive bidding processes, facilitating efficient resource allocation and negotiation. Overall, this synergy between multi-auction MAS and mobile migration inter-container enhances the system's responsiveness, robustness, and effectiveness in complex, distributed scenarios.