



Lab Reports

Course: Agent Technology

Instructor: Mrs. B. Dellal-Hedjazi

Submitted by: SAYOUD Maissa

ID: 191931040670

Dated: May 25th, 2024

Department of Artificial Intelligence and Data Science

Masters in Intelligent Computer Systems

Faculty of Computer Science

USTHB

Table of content

Lab#01: Forward Chaining and Backward Chaining.....	2
Lab#02: Jade Introduction.....	10
Lab#03: JADE Communication	18
Lab#04: Behaviours	26

Lab Report #01

Lab#01: Forward Chaining and Backward Chaining

Introduction

In this lab, we implemented the backward chaining and forward chaining algorithms in Java. Additionally, we integrated a graphical user interface (GUI) for a better visualization.

Note: We have already presented the Forward Chaining Algorithm in a google meet session.

1. Forward chaining

Artificial intelligence and expert systems employ the forward chaining technique to draw inferences from a starting collection of facts. In this approach, the system begins with the known facts and works its way up to the desired outcome by repeatedly applying inference rules to infer new data.

1.1. Basic Steps of the Forward Chaining Algorithm

1. Initialization:

We start with an initial set of known facts which is our knowledge base and we define after that a set of rules.

2. Rule Application:

In the loop, for each rule, we check if its premises are satisfied by the current set of known facts. If the premises are satisfied, we apply the rule to get the conclusions.

3. Updating Knowledge Base:

We add the conclusions derived from each rule to the knowledge base.

4. Goal Checking:

We Repeat the process of rule application and knowledge base update until we reach our goal or no other deductions can be applied. If the goal is reached, this is the end of the algorithm with success. If the goal is not reached yet, it continues until no more rules can be applied.

5. In the end, we indicate whether the goal is achievable based on the available knowledge and rules.

1.2. Implementation Details

In this section, we detail the implementation of each class involved in the Java code:

Main Class: It displays the graphical user interface (GUI), sets its components up, initializes the knowledge base rules and handles the interaction between the GUI and the forward chaining algorithm.

Rule Class: The Rule class represents individual rules in the knowledge base. Each rule consists of premises (P) and a conclusion (C). This class stores them of a rule, keeps track of the state of the rule.

ChooseRule Class: It selects a rule from a list of candidate rules.

ForwardChaining Class: It simply implements the forward chaining algorithm. It returns a boolean value for whether the goal has been proven or not.

1.3. Example Scenario

We have this starting Rule base where all of the rules are previously applicable:

Rule 1: Premises: A B - Conclusion: F

Rule 2: Premises: F H - Conclusion: I

Rule 3: Premises: D H G - Conclusion: A

Rule 4: Premises: O G - Conclusion: H

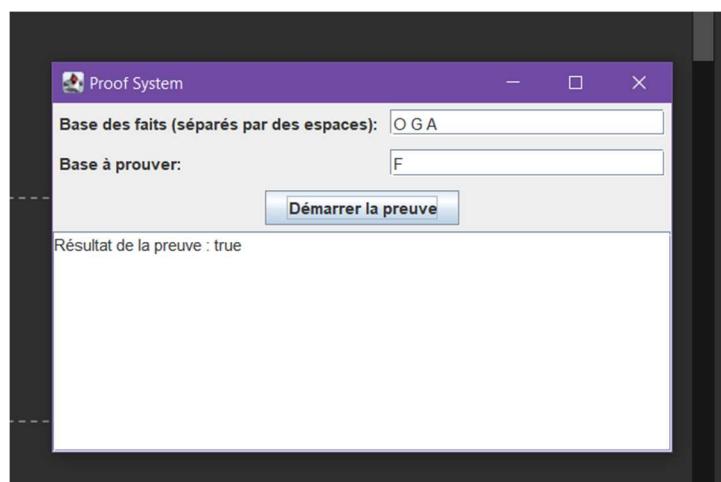
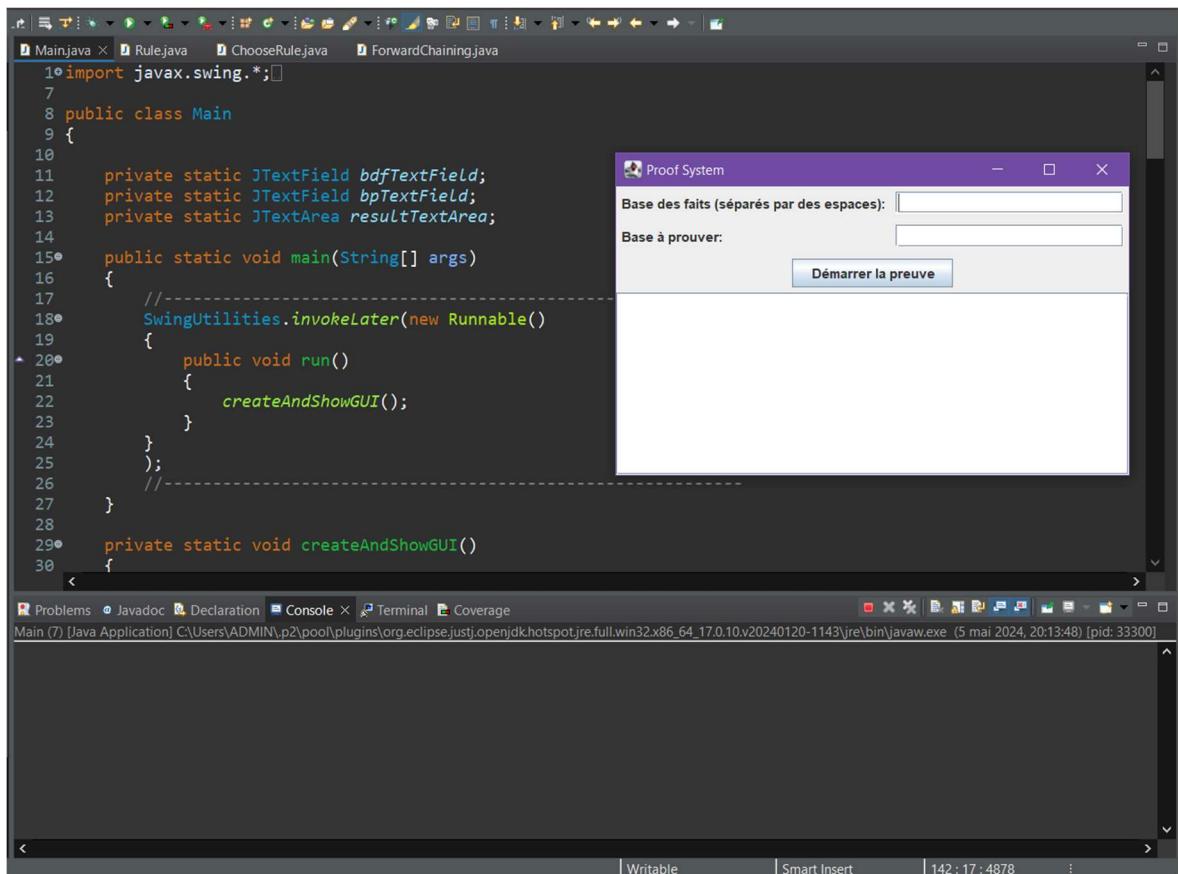
Rule 5: Premises: E H - Conclusion: B

Rule 6: Premises: G A - Conclusion: B

Rule 7: Premises: G H - Conclusion: P

Rule 8: Premises: G H - Conclusion: O

Rule 9: Premises: D O G - Conclusion: J



The image shows two side-by-side screenshots of a Java application's console window. Both windows have tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The title bar of both windows indicates 'Main (7) [Java Application] C:\Users\ADMIN\p2\pool\plugins'.
 The left window displays the following text:
 je remove la regle: 3
 regles non declenchees :
 0
 1
 2
 4
 5
 6
 7
 8
 je remove la regle: 5
 regles non declenchees :
 0
 1
 ?
 (suite)
 The right window displays:
 1
 2
 4
 6
 7
 8
 je remove la regle: 0
 regles non declenchees :
 1
 2
 4
 6
 7
 8
 SUCCESS !
 <

2. Backward Chaining

Another technique for reasoning in artificial intelligence and expert systems is backward chaining, which simply starts with the goal and moves backward to the known facts in order to demonstrate the given goal.

2.1. Basic Steps of the Backward Chaining Algorithm

The steps of backward chaining were already explained and discussed in the lab session, so we recall the algorithm:

```

Fonction chaînageArrière
Paramètres : in BR, in BF, in listeButs.
if estVide(listeButs) then
    res ← SUCCES
else
    if demBut(premier(listeButs)) then
        res ← chaînageArrière(suite(listeButs))
    else
        res ← ECHEC
    end if
end if
retourner res

```

```

Fonction demBut
Paramètres : in BR, in BF, in but.
if but ∈ BF then
    res ← SUCCES
else
    regles ← BR; res ← ECHEC
    while regles ≠ ∅ et res ≠ SUCCES do
        r ← choix(regles); regles ← regles - {r}
        if But ∈ conclusion(r) then
            res ← chaînageArrière(BR, BF, premisses(r))
        end if
    end while
    retourner res
end if

```

2.2. Implementation Details

Even though the backward chaining approach was implemented during the lab session, I additionally integrated a graphical user interface (GUI) in the backward chaining too.

Main Class: It sets up the graphical user interface (GUI), takes input for the base of facts (bdf) and the goal to prove (bp).

Backward Chaining: It contains the Back Chaining algorithm implementation. We recursively apply the backward chaining to try to prove the goal, starting from this latter to finding the base facts.

Rest Class: It excludes the first element of the given list and return the rest of it.

TakeGoal Class: It takes a goal and determines if this goal can be derived from the knowledge base, it finds also the conflict set (RA) which is consisted of rules with the goal as a conclusion and applies backward chaining to each rule in the conflict set.

2.3. Example Scenario

We use the same rules shown earlier with the forward chaining example, same collection of starting facts and the same goal.

The screenshot shows a Java development environment with several tabs at the top: BackChainingJava, Mainjava (selected), RestJava, RuleJava, and TakeGoal.java. The code editor displays Java code for a proof system, specifically a backtracking algorithm. A modal dialog titled "Proof System" is open in the center. It has two input fields: "Base des faits (séparés par des espaces):" containing "O G A" and "Base à prouver:" containing "F". Below these fields is a button labeled "Démarrer la preuve". Underneath the dialog, the status bar shows the message "Résultat de la preuve : true". The bottom of the screen shows the Eclipse interface with toolbars and status bars.

```
131         bdr[7] = new Rule(7,p,c);
132
133         // -- 8
134         p = new ArrayList <> (Arrays.asList("D","O","G"));
135         c = new ArrayList <> (Arrays.asList("J"));
136
137         bdr[8] = new Rule(8,p,c);
138
139         // Appel de la fonction Backchain
140         boolean test = BackChaining.BackChain(bdr, bdf, bp);
141
142         // Mise à jour du JTextArea avec le résultat
143         resultTextArea.setText("Résultat de la preuve : " + test);
144     }
145 }
146
147 //-----
148 frame.pack();
149 frame.setVisible(true);
150 //-----
151 }
152 }
153
154
155
156 }
```

Another example:

The screenshot shows a Java development environment with several tabs at the top: BackChaining.java, Mainjava, Restjava, Rule.java, and TakeGoaljava. The code editor displays Java code for implementing a backtracking algorithm. A modal dialog titled "Proof System" is open, containing two input fields: "Base des faits (séparés par des espaces):" with the value "A B" and "Base à prouver:" with an empty input field. Below these fields is a button labeled "Démarrer la preuve". In the bottom right corner of the dialog, there is a message box with the text "Résultat de la preuve : false". In the background, a terminal window shows the command "javaw.exe" running with the process ID 27840.

```
131         bdr[7] = new Rule(7,p,c);
132
133         // -- 8
134         p = new ArrayList <> (Arrays.asList("D","O","G"));
135         c = new ArrayList <> (Arrays.asList("J"));
136
137         bdr[8] = new Rule(8,p,c);
138
139         // Appel de la fonction Backchain
140         boolean test = BackChaining.BackChain(bdr, bdf, bp);
141
142         // Mise à jour du JTextArea avec le résultat
143         resultTextArea.setText("Résultat de la preuve : " + test);
144     });
145 }
146
147 //-----
148 frame.pack();
149 frame.setVisible(true);
150 //-----
151 }
152 }
```

Conclusion

In this lab, we implemented both forward chaining and backward chaining algorithms in Java, we gained a deeper understanding of how they operate and their respective strengths and weaknesses.

Lab Report #02

Lab#02: Jade Introduction

Introduction

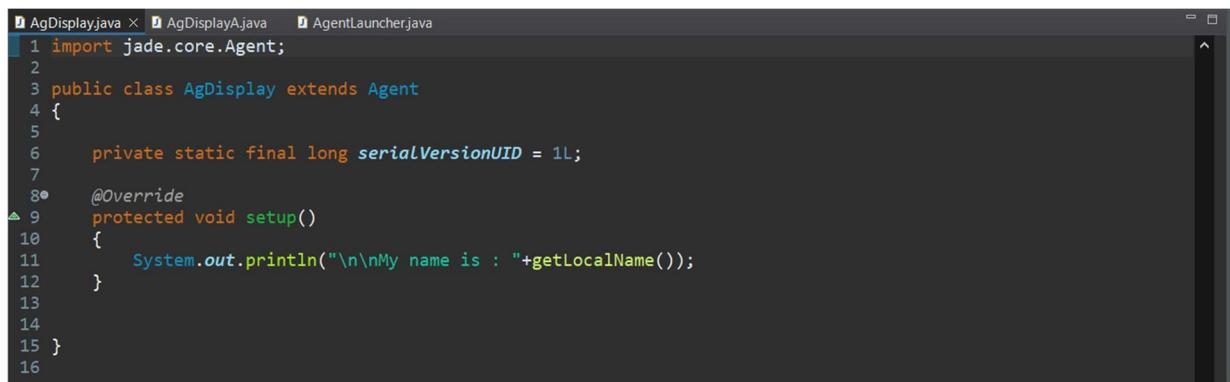
JADE provides a comprehensive platform for developing multi-agent systems (MAS) in Java.

In this lab, we explore various aspects of it through many exercises.

Please find the attached codes of the exercises in the **TechAgent_TP3_Agents.zip** file.

Exercise 01

1/ Create an agent in JADE platform that displays the message « Hello world » followed by its name.

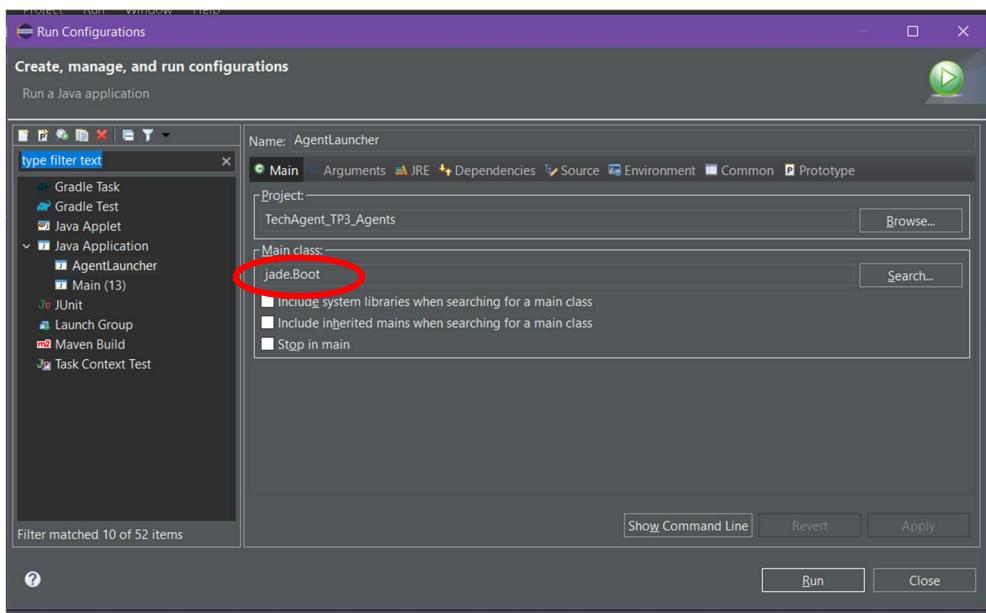


```
AgDisplay.java X AgDisplayAjax AgentLauncher.java
1 import jade.core.Agent;
2
3 public class AgDisplay extends Agent
4 {
5
6     private static final long serialVersionUID = 1L;
7
8     @Override
9     protected void setup()
10    {
11        System.out.println("\n\nMy name is : "+getLocalName());
12    }
13
14
15 }
16
```

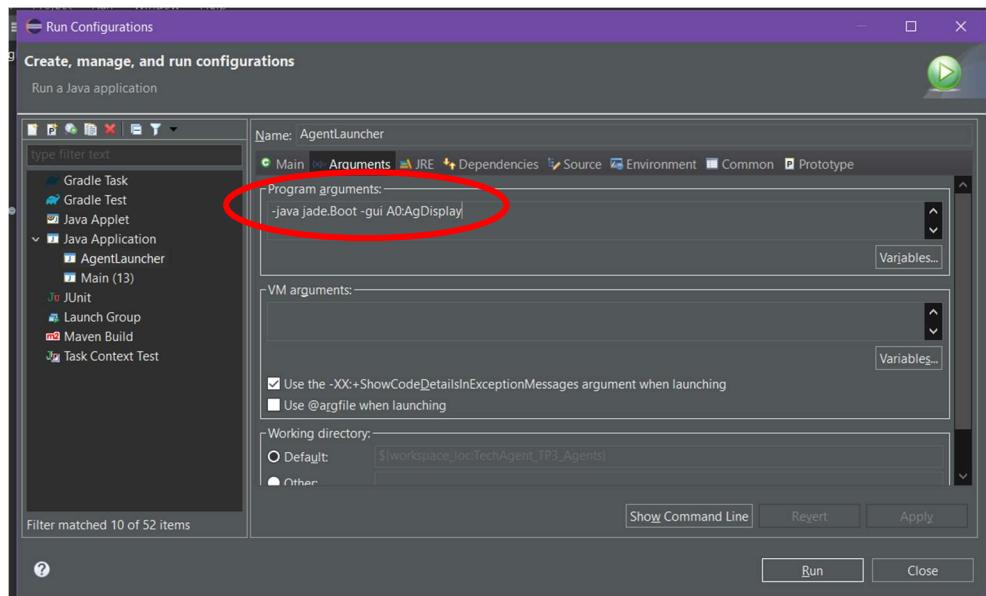
A screenshot of a code editor window showing a Java file named 'AgDisplay.java'. The code defines a class 'AgDisplay' that extends 'Agent'. It contains a single method 'setup()' which prints the agent's local name to the console. The code editor interface shows tabs for 'AgDisplay.java', 'AgDisplayAjax', and 'AgentLauncher.java'. The code is numbered from 1 to 16.

2/ Give the steps for launching one agent.

To launch one agent in JADE, we choose the **run as run configurations** option.



We specify the jade.Boot in the Main class field and then we type the following command in Arguments/Program arguments : **-java jade.Boot -gui A0:AgDisplay** in which we give a name to the agent and specify clearly the the class containing it.



```

1 import jade.core.Agent;
2

INFO: -----
This is JADE 4.6.0 - revision 6869 of 30-11-2022 14:47:03
downloaded in Open Source, under LGPL license
at http://jade.tilab.com/
-----
mai 06, 2024 1:03:41 AM jade.imtp.leap.LEAPIMTP
INFO: Listening for intra-platform commands
- jicp://192.168.1.11:1099

mai 06, 2024 1:03:42 AM jade.core.BaseService
INFO: Service jade.core.management.AgentManager
mai 06, 2024 1:03:42 AM jade.core.BaseService
INFO: Service jade.core.messaging.Messaging
mai 06, 2024 1:03:42 AM jade.core.BaseService
INFO: Service jade.core.resource.ResourceManager
mai 06, 2024 1:03:42 AM jade.core.BaseService
INFO: Service jade.core.mobility.AgentMobility
mai 06, 2024 1:03:42 AM jade.core.BaseService
INFO: Service jade.core.event.Notification
mai 06, 2024 1:03:42 AM jade.mtp.http.HTTPServer
INFO: HTTP-MTP Using XML parser com.sun.org.apache.xml.resolver
mai 06, 2024 1:03:42 AM jade.core.messaging.Messaging
INFO: MTP addresses:
http://DESKTOP-ITR9E5M:7778/acc

My name is : A0
mai 06, 2024 1:03:42 AM jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Main-Container@192.168.1.11 is ready.

```

3/ Give the steps for launching two agents.

To launch two agents we just change the command in Program Arguments to: **-java jade.Boot -gui A0:AgDisplay;A1:AgDisplay**

```

1 import jade.core.Agent;
2

INFO: -----
This is JADE 4.6.0 - revision 6869 of 30-11-2022 14:47:03
downloaded in Open Source, under LGPL license
at http://jade.tilab.com/
-----
mai 06, 2024 1:05:22 AM jade.imtp.leap.LEAPIMTP
INFO: Listening for intra-platform commands
- jicp://192.168.1.11:1099

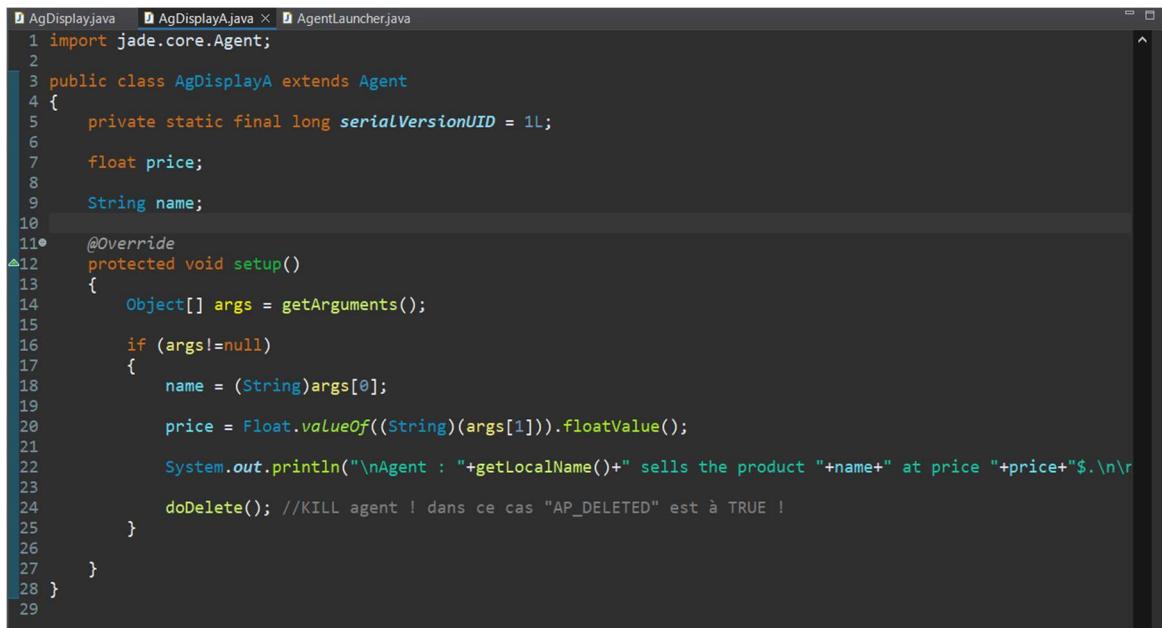
mai 06, 2024 1:05:22 AM jade.core.BaseService
INFO: Service jade.core.management.AgentManager
mai 06, 2024 1:05:22 AM jade.core.BaseService
INFO: Service jade.core.messaging.Messaging
mai 06, 2024 1:05:22 AM jade.core.BaseService
INFO: Service jade.core.resource.ResourceManager
mai 06, 2024 1:05:22 AM jade.core.BaseService
INFO: Service jade.core.mobility.AgentMobility
mai 06, 2024 1:05:22 AM jade.core.BaseService
INFO: Service jade.core.event.Notification
mai 06, 2024 1:05:22 AM jade.mtp.http.HTTPServer
INFO: HTTP-MTP Using XML parser com.sun.org.apache.xml.resolver
mai 06, 2024 1:05:22 AM jade.core.messaging.Messaging
INFO: MTP addresses:
http://DESKTOP-ITR9E5M:7778/acc

My name is : A0
My name is : A1
mai 06, 2024 1:05:22 AM jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Main-Container@192.168.1.11 is ready.

```

Exercise 02

1/ Develop a Jade agent class, which receives data as arguments and displays it.

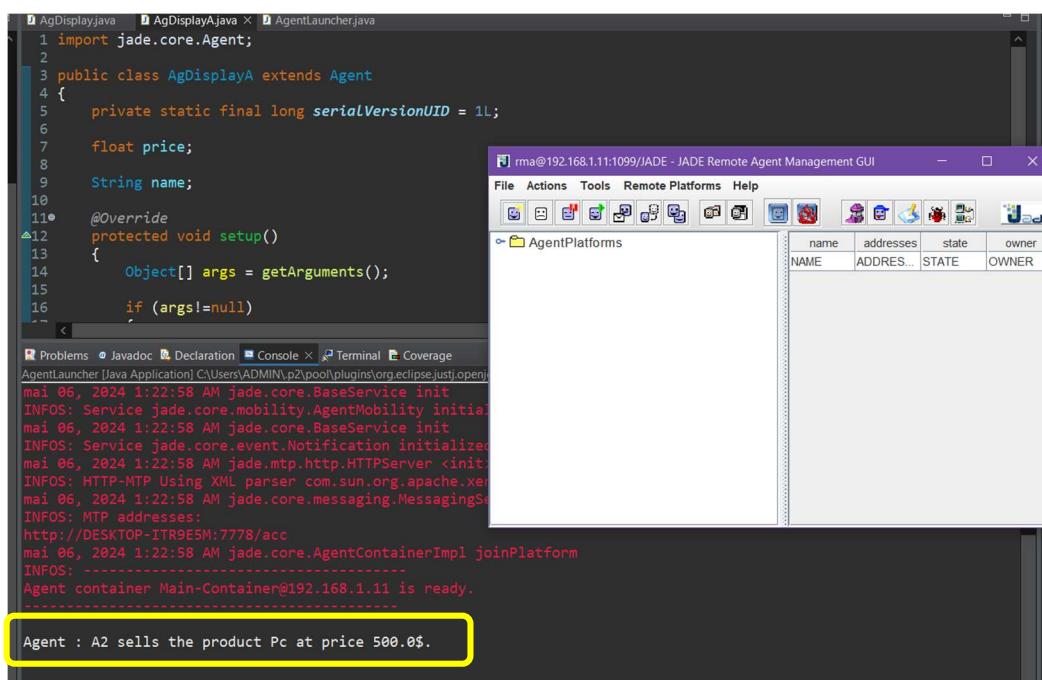


```
AgDisplay.java  AgDisplayAjava X AgentLauncher.java
1 import jade.core.Agent;
2
3 public class AgDisplayA extends Agent
4 {
5     private static final long serialVersionUID = 1L;
6
7     float price;
8
9     String name;
10
11    @Override
12    protected void setup()
13    {
14        Object[] args = getArguments();
15
16        if (args!=null)
17        {
18            name = (String)args[0];
19
20            price = Float.valueOf((String)(args[1])).floatValue();
21
22            System.out.println("\nAgent : "+getLocalName()+" sells the product "+name+" at price "+price+"\$.\n");
23
24            doDelete(); //KILL agent ! dans ce cas "AP_DELETED" est à TRUE !
25        }
26    }
27 }
28 }
```

2/ Give the steps for launching an agent with passing arguments.

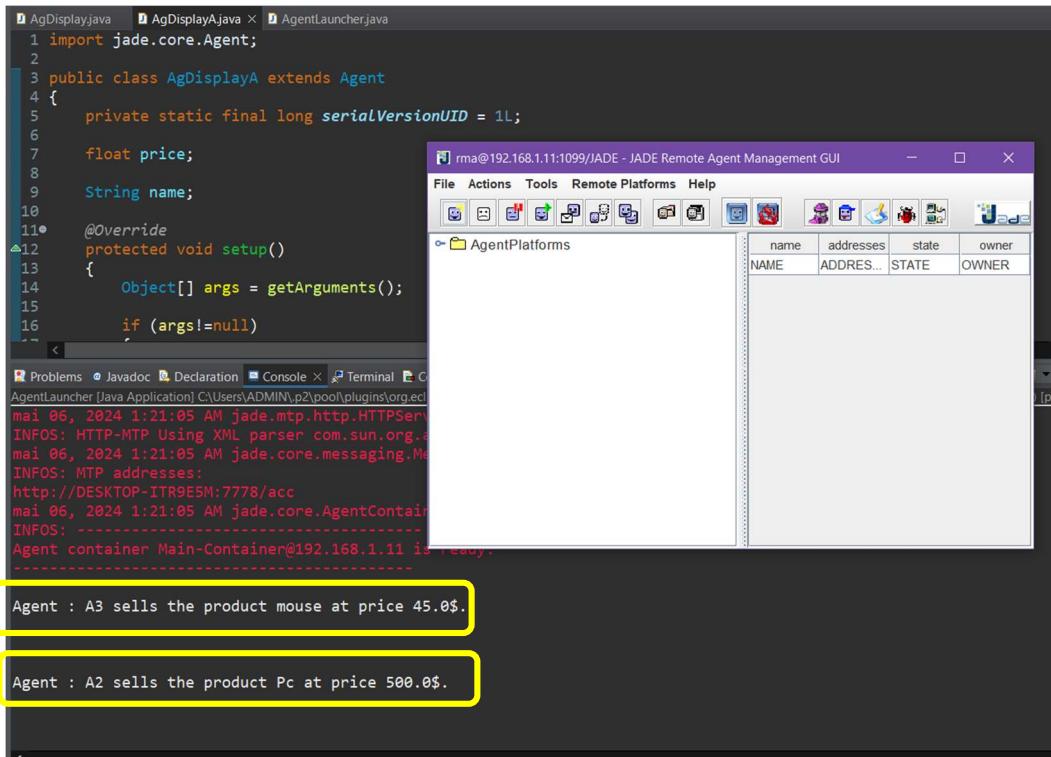
We follow the exact same steps as in the second question of the first exercise, with a slight change in the command:

-java jade.Boot -gui A2:AgDisplayA(Pc,500)



3/ Give the steps for launching two agents with passing arguments.

-java jade.Boot -gui A2:AgDisplayA(Pc,500);A3:AgDisplayA(mouse, 45)



The screenshot shows the Eclipse IDE interface with two open files: `AgDisplayJava` and `AgentLauncherJava`. The `AgDisplayJava` file contains Java code for an agent named `AgDisplayA` that overrides the `setup()` method to handle command-line arguments. The `AgentLauncherJava` file contains the main method for launching the JADE framework and creating agents. In the background, the `JADE Remote Agent Management GUI` window is open, showing a list of agents under the `AgentPlatforms` section. Two agents are listed: `A3` and `A2`. The console output in the Eclipse interface shows messages from the agents: `Agent : A3 sells the product mouse at price 45.0$.` and `Agent : A2 sells the product Pc at price 500.0$.`

Exercise 03 1/ Repeat exercise 1 to launch the agent by program. (Java)

```
12●  public static void main(String[] args)
13  {
14      try
15      {
16          Runtime rt = Runtime.instance();
17
18          ProfileImpl p = new ProfileImpl("localhost",1099,"JADE");
19
20          //pour afficher l'interface graphique il faudra rajouter les param:
21          p.setParameter(Profile.LOCAL_HOST, "localhost");
22          p.setParameter(Profile.LOCAL_PORT, "1099");
23          p.setParameter(Profile.GUI, "true");
24
25          //Object [] Arg1 = { "Pc", "500"};
26          //Object [] Arg2 = {"Mouse", "45"};
27
28          ContainerController mc = rt.createMainContainer(p);
29
30          AgentController ag1 = mc.createNewAgent("Buyer1","AgDisplay",null);
31          AgentController ag2 = mc.createNewAgent("Buyer2","AgDisplay",null) (Red circle around the second argument null)
32
33          //AgentController ag3 = mc.createNewAgent("Buyer3","AgDisplay",Arg1);
34          //AgentController ag4 = mc.createNewAgent("Buyer4","AgDisplayA",Arg2);
35
36          ag1.start();
37          ag2.start();
38          //ag3.start();
39          //ag4.start();
40      }
41      catch (Exception e)
42      [
43          System.out.println("Une erreur s'est produite : " + e.getMessage());
44          //e.printStackTrace();
45      }
46  }
47 }
```

The screenshot shows a Java application window with a code editor and a graphical user interface. The code editor contains Java code for creating agents in a JADE container. The graphical interface is the JADE Remote Agent Management GUI, showing a tree view of agent platforms and a list of agents under a 'Main-Container'. Two agents are highlighted with red boxes: 'Buyer1@JADE' and 'Buyer2@JADE'. The list also includes 'ams@JADE', 'df@JADE', and 'rma@JADE'. In the code editor's console, two messages are printed: 'My name is : Buyer1' and 'My name is : Buyer2', each enclosed in a yellow box.

```

24
25     //Object [] Arg1 = { "Pc", "500"};
26     //Object [] Arg2 = {"Mouse", "45"};
27
28     ContainerController mc = rt.createMainContainer();
29
30     AgentController ag1 = mc.createNewAgent("Buyer1", "AgDisplay", null);
31     AgentController ag2 = mc.createNewAgent("Buyer2", "AgDisplay", null);
32
33     ag3.start();
34     ag4.start();
35
36 }
37 catch (Exception e)
38 {
39     System.out.println("Une erreur s'est produite : " + e.getMessage());
40     e.printStackTrace();
41 }
42
43 }
44
45 }
46
47 }

```

Exercise 04

1/ Repeat exercise 2 to launch the agent programmatically. Launch two agents, each with different input data (arguments).

The screenshot shows a Java code editor with code for launching four agents. The code uses a try-catch block to handle exceptions. It creates a Runtime instance, sets up a ProfileImpl for JADE, and then creates four agents named 'Buyer1' through 'Buyer4'. The first two agents are created with arguments 'Pc' and '500', while the last two are created with 'Mouse' and '45'. The code also includes comments for setting up a graphical interface and starting the agents. A red box highlights the argument lines for the first two agents, and a red circle highlights the argument line for the fourth agent.

```

12*   public static void main(String[] args)
13  [
14      try
15      {
16          Runtime rt = Runtime.instance();
17
18          ProfileImpl p = new ProfileImpl("localhost",1099,"JADE");
19
20          //pour afficher l'interface graphique il faudra rajouter les param:
21          p.setParameter(Profile.LOCAL_HOST, "localhost");
22          p.setParameter(Profile.LOCAL_PORT, "1099");
23          p.setParameter(Profile.GUI, "true");
24
25          Object [] Arg1 = { "Pc", "500"};
26          Object [] Arg2 = {"Mouse", "45"};
27
28          ContainerController mc = rt.createMainContainer(p);
29
30          //AgentController ag1 = mc.createNewAgent("Buyer1", "AgDisplay",null);
31          //AgentController ag2 = mc.createNewAgent("Buyer2", "AgDisplay",null);
32
33          AgentController ag3 = mc.createNewAgent("Buyer3", "AgDisplayA", Arg1);
34          AgentController ag4 = mc.createNewAgent("Buyer4", "AgDisplayA", Arg2);
35
36          //ag1.start();
37          //ag2.start();
38          ag3.start();
39          ag4.start();
40      }
41      catch (Exception e)
42      {
43          System.out.println("Une erreur s'est produite : " + e.getMessage());
44          e.printStackTrace();
45      }
46  }
47 }

```

The screenshot shows a Java code editor with the following code:

```
36         //ag1.start();
37         //ag2.start();
38         ag3.start();
39         ag4.start();
40     }
41     catch (Exception e)
42     {
43         System.out.println("Une erreur s'est produite");
44         //e.printStackTrace();
45     }

```

Below the code is a terminal window showing logs:

```
AgentLauncher (1) [Java Application] C:\Users\ADMINI~1.p2\pool\plugins\org.eclipse.jst:jopenjdk.ho
mai 06, 2024 1:38:18 AM jade.core.AgentContainerImpl joinPla
INFOS: -----
Agent container Main-Container@192.168.1.11 is ready.
-----
Agent : Buyer4 sells the product Mouse at price 45.0$.
```

A yellow box highlights the line "Agent : Buyer4 sells the product Mouse at price 45.0\$.". To the right is a table titled "AgentPlatforms" with columns "name" and "addresses".

Conclusion

Through these four exercises, we learned and manipulated some Jade operations including agent creation, argument passing to the agents and programmatically launching agents in order to gain familiarity.

Lab Report #03

Lab#03: JADE Communication

Introduction

On JADE communication lab#03, we explore the agent-to-agent communication through a series of exercises, we'll explore point-to-point messaging, message filtering, topic-based broadcasting, and serialization.

Exercise 1: (Point-to-point communication)

Let there be two agents (agent1, agent2). Give the code of each agent in JADE to carry out their following behaviour:

- Agent 2 sends a “Hello” message to Agent 1
- Agent 1 responds with a “Thank you” message
- Each agent displays the message received from the other agent.

Solution

Please find the attached code in the **TechAgent_TP4_ACLMessages.zip** file.

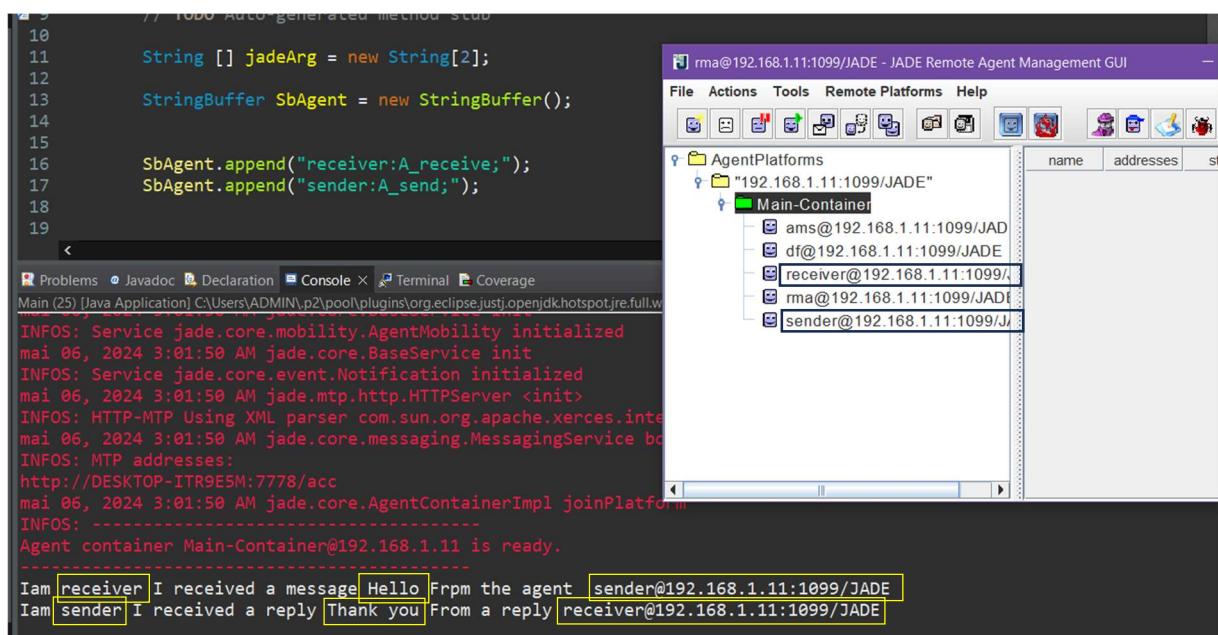
```
7 public class A_receive extends Agent
8 {
9     private static final long serialVersionUID = 1L;
10    protected void setup()
11    {
12        addBehaviour(new CyclicBehaviour(this)
13        {
14            private static final long serialVersionUID = 1L;
15            @Override
16            public void action()
17            {
18                // TODO Auto-generated method stub
19                MessageTemplate mt= MessageTemplate.MatchPerformativ(ACLMessage.INFORM);
20
21                ACLMessage msg= receive(mt);
22
23                if(msg!=null)
24                {
25                    System.out.println("Iam "+ getLocalName() +" I received a message " + msg.getContent() +" F rpm the agent " + msg.getSender().getName());
26                    ACLMessage reply=msg.createReply();
27
28                    reply.setPerformative(ACLMessage.INFORM);
29                    reply.setContent("Thank you");
30
31                    send(reply);
32                }
33            }
34        });
35    }
36}
```

```

6 public class A_send extends Agent
7 {
8     private static final long serialVersionUID = 1L;
9
10    protected void setup()
11    {
12        ACLMessage msg=new ACLMessage(ACLMessage.INFORM);
13
14        msg.setContent("Hello");
15        msg.addReceiver(new AID("receiver",AID.ISLOCALNAME));
16
17        send(msg);
18
19        addBehaviour(new CyclicBehaviour(this)
20        {
21
22            private static final long serialVersionUID = 1L;
23
24            @Override
25            public void action()
26            {
27                // TODO Auto-generated method stub
28
29                ACLMessage msg=new ACLMessage(ACLMessage.INFORM);
30
31                msg=receive();
32
33                if(msg!=null)
34                {
35                    System.out.println("Iam "+ getLocalName() +" I received a reply " + msg.getContent() +" From a reply "+ msg.getSender().getName());
36                }
37                block();
38            }
39        });
40    }
41 }
42

```

As a result:



Exercise 2

- Create the same receiver agent in main container.

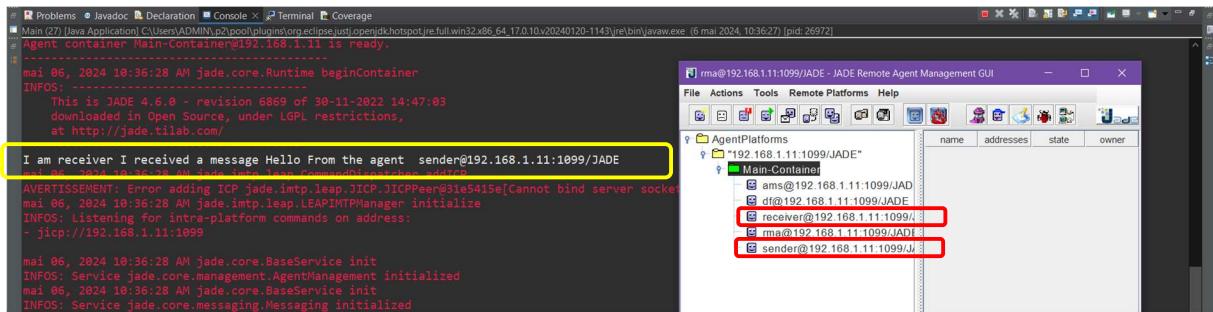
Please find the attached code in the **TechAgent_TP3_ContainerA.zip** file.

```

2 public class Main
3 {
4
5     public static void main(String[] args)
6     {
7         String[] jadeArg = new String[2];
8         StringBuffer SbAgent = new StringBuffer();
9
10        SbAgent.append("receiver:a_receive;");
11        SbAgent.append("sender:a_sender;");
12
13        jadeArg[0] = "-gui";
14        jadeArg[1] = SbAgent.toString();
15
16        jade.Boot.main(jadeArg);
17
18        // Creation of the receiver agent in the main container
19        jade.core.Runtime rt = jade.core.Runtime.instance();
20        jade.core.ProfileImpl p = new jade.core.ProfileImpl(false);
21
22        p.setParameter(jade.core.Profile.MAIN_HOST, "localhost");
23
24        jade.wrapper.AgentContainer mainContainer = rt.createMainContainer(p);
25        try
26        {
27            jade.wrapper.AgentController receiver = mainContainer.createNewAgent("receiver", "a_receive", null);
28
29            receiver.start();
30        }
31        catch (jade.wrapper.ControllerException e)
32        {
33            e.printStackTrace();
34        }
35    }
36}

```

After starting the agents, it creates a new instance of a_receive agent and starts it in the main container. Like this, both sender and receiver agents are created and run in the main container.



- b) Create a simple container and the same sender agent in it (by program).

Please find the attached code in the **TechAgent_TP3_ContainerB.zip** file.

We add a new class called simplecontainer in which we define the necessary steps for creating a simple container and the same sender agent in it.

```

8 public class simplecontainer
9 {
10     public static void main(String[] args)
11     {
12         try
13         {
14             Runtime rt=Runtime.instance();
15
16             ProfileImpl p= new ProfileImpl(false);
17
18             p.setParameter(Profile.MAIN_HOST, "localhost");
19
20             AgentContainer container=rt.createAgentContainer(p);
21
22             //container.start();
23
24             AgentController ag2=container.createNewAgent("agent2","a_sender", null);
25
26             ag2.start();
27         }
28         catch (ControllerException e)
29         {
30             // TODO Auto-generated catch block
31             e.printStackTrace();
32         }
33     }
34 }
35

```

It creates a simple container, and within it, it instantiates and runs an agent of the class a_sender.

Exercise 4+5: (Broadcast Communication & Serialization)

Realise Topic-based communication between a sender agent that creates a topic « JADE » and a set of receiver agents that register to that topic.

Consider a product defined by a name and a price. Write the code of two agents (Seller and Buyer) who communicate as follows: the Seller agent creates a new product and sends it to the Buyer agent.

The Buyer agent displays the product received.

(Buyer and seller instead of sender and receiver)

Solution

We implemented the following java classes and run them as run configurations with the command:

-gui -services jade.core.messaging.TopicManagementService A0:A2Topic;A1:A1Topic;A2:A1Topic;A3:A1Topic

```
1 public class AITopic extends Agent
2 {
3     private static final long serialVersionUID = 1L;
4
5     @Override
6     protected void setup()
7     {
8         try
9         {
10             TopicManagementHelper TopicHelper = (TopicManagementHelper) getHelper(TopicManagementHelper.SERVICE_NAME);
11
12             final AID topic = TopicHelper.createTopic("JADE");
13
14             TopicHelper.register(topic);
15
16             addBehaviour(new CyclicBehaviour(this)
17             {
18
19                 private static final long serialVersionUID = 1L;
20
21                 public void action ()
22                 {
23                     ACLMessage msg = receive(MessageTemplate.MatchTopic(topic));
24
25                     if (msg!=null)
26                     {
27                         System.out.println("\nAgent "+getLocalName()+" : Message about topic "+topic.getLocalName()+" received content is : "+msg.getContent());
28                     }
29                     else
30                     {
31                         block();
32                     }
33                 }
34             });
35         }
36         catch (ServiceException e)
37         {
38             e.printStackTrace();
39         }
40     }
41 }
```

```
5
6 public class Buyer extends Agent
7 {
8
9     private static final long serialVersionUID = 1L ;
10
11●    @Override
12●    protected void setup()
13{
14    ACLMessage m;
15
16    Product P;
17
18    m = receive();
19
20    if(m!=null)
21    {
22        try
23        {
24            P= (Product)m.getContentObject();
25
26            System.out.println("I am "+getLocalName()+" I received the product "+P.name+" with the price : "+P.price);
27        }
28        catch (Exception e)
29        {
30
31        }
32    }
33}
34
```

```
10 public class Seller extends Agent
11 {
12     private static final long serialVersionUID = 1L ;
13
14
15     Product P = new Product() ;
16
17     ACLMessage m;
18
19•     @Override
20     protected void setup()
21     {
22         P.price = 2500 ;
23         P.name = "Keyboard" ;
24
25         m = new ACLMessage(ACLMessage.INFORM);
26
27         m.addReceiver ( new AID("Buyer",AID.ISLOCALNAME));
28
29
30
31         try
32         {
33             m.setContentObject(P);
34
35             m.setLanguage("JavaSerialization");
36             //le msg s'envoie comme une chaîne de caractères puis il se reconstitue lors de son arrivée en forme d'objet Produit !
37
38             send(m);
39         }
40         catch (Exception e)
41     }
```

```
1 import java.io.*;  
2  
3  
4 public class Product implements Serializable  
5 {  
6  
7     //private static final long serialVersionUID = 1L;  
8  
9     double price;  
10    String name;  
11 }  
12  
13
```

```
Problems Javadoc Declaration Console Terminal Coverage  
Main (18) [Java Application] C:\Users\ADMIN\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86_64.17.0.10.v20240120-1143\jre\bin\  
-----  
mai 06, 2024 3:48:13 AM jade.tools.ToolAgent init  
AVERTISSEMENT: NotificationService not installed. Some tool may not work properly.  
  
Agent A0 : Sending message about topic JADE  
  
Agent A2 : Message about topic JADE received contentent is : 1  
  
Agent A3 : Message about topic JADE received contentent is : 1  
  
Agent A1 : Message about topic JADE received contentent is : 1  
  
Agent A0 : Sending message about topic JADE  
  
Agent A2 : Message about topic JADE received contentent is : 2  
  
Agent A3 : Message about topic JADE received contentent is : 2  
  
Agent A1 : Message about topic JADE received contentent is : 2
```

The screenshot shows two windows side-by-side. The left window is the Eclipse IDE's Console tab, displaying the same JADE tool initialization and agent message exchange as the previous screenshot. The right window is the "JADE Remote Agent Management GUI" (RMA), titled "rma@192.168.1.11:1099/JADE - JADE Remote Agent Management GUI". It features a toolbar at the top with various icons for file operations, actions, and tools. Below the toolbar is a menu bar with File, Actions, Tools, Remote Platforms, and Help. The main interface consists of a tree view on the left and a table on the right. The tree view shows an "AgentPlatforms" node expanded to show a "Main-Container" node under "192.168.1.11:1099/JADE". The table on the right has columns for "name", "addresses", and "state". Under the "Main-Container" node, there are eight entries listed in the table:

name	addresses	state
A0@192.168.1.11:1099/JADE		
A1@192.168.1.11:1099/JADE		
A2@192.168.1.11:1099/JADE		
A3@192.168.1.11:1099/JADE		
ams@192.168.1.11:1099/JAD		
df@192.168.1.11:1099/JADE		
rma@192.168.1.11:1099/JADE		

Please find the attached code in the **TechAgent_TP5_ACLMessages.zip** file.

We combined in this code both broadcast communication and serialization.

As we can notice, in one hand, the agents A1Topic and A2Topic are participating in topic-based communication: A1Topic receives messages published to that topic by other agents while A2Topic publishes messages about the topic using a TickerBehaviour which allows the broadcasting.

In the other hand, the Product class indicates that elements of this class can be serialized, for example: the Seller agent which is an instance of the Product class (P) is serialized and sent as the content of an ACLMessage to the Buyer agent. Upon receiving the message, the Buyer agent deserializes the content back into a Product object and processes it.

Conclusion

In this lab#03 work, we learned the ACLMessage mechanism and some functions related to it like: Point-to-point communication and Broadcast communication through topic-based messaging, we explored also the Container concept and Serialization for transmitting complex data objects between agents.

Lab Report #04

Lab#04: Behaviours

Introduction

In this lab we focused on understanding and implementing various types of behaviors in JADE agents. we learned about agent migration, primitive behaviors, and composite behaviors to understand how agents can perform actions autonomously and interact with their environment.

Exercise 1: (Agent Migration: Mobile Agent)

Create an agent under a container (container_2). Migrate it through JADE GUI (Interface) to MainContainer by displaying a message before, during and after moving.

Solution

```
public class simplecontainer
{
    public static void main(String[] args)
    {
        // Get a hold of the JADE runtime
        Runtime rt = Runtime.instance();

        // Create a default profile for the main container
        Profile pMain = new ProfileImpl(true); // true means it's the main container
        pMain.setParameter(Profile.MAIN_HOST, "localhost");

        // Create and start the main container
        AgentContainer mainContainer = rt.createMainContainer(pMain);

        try
        {
            // Start the main container
            mainContainer.start();

            // Create a profile for the additional container
            ProfileImpl pContainer = new ProfileImpl(false); // false means it's not the main container
            pContainer.setParameter(Profile.MAIN_HOST, "localhost");

            // Create the additional container
            AgentContainer container = rt.createAgentContainer(pContainer);

            // Create and start agents in the additional container
            AgentController agent1 = container.createNewAgent("agent1", "BehaviourComplex", null);
            AgentController agent2 = container.createNewAgent("agent2", "BehaviourComplex", null);

            agent1.start();
            agent2.start();
        }
        catch (ControllerException e)
        {
            e.printStackTrace();
        }
    }
}
```

```

protected void takeDown()
{
    System.out.println("agent terminated");
}
public void DoMove(Location L) {
    System.out.println("Agent migration"+L.getName());
}
protected void beforeMove() {
    System.out.println("before migration"+ this.getAID().getName());

    try
    {
        System.out.println("from container"+this.getContainerController().getContainerName());
    }
    catch (ControllerException e)
    {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

```

protected void afterMove()
{
    System.out.println("after migration"+ this.getAID().getName());
    try
    {
        System.out.println("to the container "+ this.getContainerController().getContainerName());
    }
    catch (ControllerException e)
    {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

Exercise 2: (Primitive Behaviour)

Create an agent running several simple behaviours. Test these behaviours each alone: Behavior, OneshotBehaviour, CyclicBehaviour, WakerBehaviour and TickerBehaviour. Each behaviour displays a simple message.

Solution

1. OneshotBehaviour

```

//-----OneShotBehaviour-----//

addBehaviour(new OneShotBehaviour()
{

    private static final long serialVersionUID = 1L;

    @Override
    public void action()
    {
        // TODO Auto-generated method stub
        System.out.println("this action runs 1 time");
    }
});

```

This behavior runs its action method only once. In this case, it prints a message.

2. CyclicBehaviour

```
//-----cyclicBehaviour-----
addBehaviour(new CyclicBehaviour()
{
    private static final long serialVersionUID = 1L;

    @Override
    public void action()
    {
        // TODO Auto-generated method stub
        System.out.println("this action runs an infinity times");
    }
});
```

This behavior runs indefinitely, continuously executing its action method and printing a message.

3. WakerBehaviour

```
//-----WakerBehaviour-----//
//ce reveille apres certain temps et fait une action
addBehaviour(new WakerBehaviour(this,6000)
{

    private static final long serialVersionUID = 1L;

    public void onWake() {
        System.out.println(getLocalName()+"i'm daying");
        doDelete();
    }
});
```

This behavior wakes up after a specified delay (6000 milliseconds) and performs an action. Here, it prints a message and then deletes the agent using doDelete().

4. TickerBehaviour

```
//-----TickerBehaviour-----
addBehaviour(new TickerBehaviour(this,30000)
{

    private static final long serialVersionUID = 1L;

    @Override
    protected void onTick() {
        // TODO Auto-generated method stub
        System.out.println("this action runs each 30s");

    }
});
```

This behavior executes its action method at regular intervals (every 30 seconds in this case), printing a message each time.

Exercise 3: (Composite behaviour)

Create and launch 2 agents that owns behaviors executing in parallel. These behaviors can be activated 3 times and display greetings (Hello) in several languages.

Example of requested result :

I, Agent a1, my address is (agent-identifier :name a1@192.168.1.4:1099/JADE :addresses
(sequence http://Admin:7778/acc))

I, Agent a2, my address is (agent-identifier :name a2@192.168.1.4:1099/JADE :addresses
(sequence http://Admin:7778/acc))

I execute several behaviors in parallel

I execute several behaviors in parallel

(2 -> 1/3) a علیکم السلام

(1 -> 1/3) a علیکم السلام

a1 -> bonjour (1/3)

a1 -> hallo (1/3)

a1 -> buongiorno (1/3)

a1 -> buenos dias (1/3)

a1 -> Olá (1/3)

a2 -> bonjour (1/3)

a2 -> hallo (1/3)

a1 -> saluton (1/3)

(1 -> 2/3) a علیکم السلام

a1 -> bonjour (2/3)

a1 -> hallo (2/3)

a2 -> buongiorno (1/3)

a2 -> buenos dias (1/3)

a1 -> buongiorno (2/3)

a1 -> buenos dias (2/3)

a2 -> Olá (1/3)

a1 -> Olá (2/3)

a2 -> saluton (1/3)

(2 -> 2/3) **عليكم السلام** a

a1 -> saluton (2/3)

Solution

In this Java program we create two agents (agent1 and agent2). Each agent executes a set of behaviors at the same time, and each behavior is responsible for printing a greeting message in different languages. These behaviors are activated three times before they terminate.

```
3
4 public class Main
5 {
6
7•   public static void main(String[] args)
8   {
9       // TODO Auto-generated method stub
10
11
12
13
14     String [] jadeArg = new String[2];
15     StringBuffer SbAgent = new StringBuffer();
16
17
18     SbAgent.append("agent:BehaviourComplex;");
19     SbAgent.append("agent1:BehaviourComplex;");
20
21     jadeArg[0] = "-gui";
22
23     jadeArg[1] = SbAgent.toString();
24
25     jade.Boot.main(jadeArg);
26
27
28   }
29
30 }
```

Initialization: The main method initializes an array jadeArgs to hold JADE command-line arguments.

Agents Definition: A StringBuffer SbAgent is used to define two agents (agent and agent1), both of type BehaviourComplex.

Arguments Assignment: The first element of jadeArgs is set to -gui, which instructs JADE to start the GUI. The second element is the agent definition string.

JADE Boot: jade.Boot.main(jadeArgs) launches the JADE runtime with the specified arguments, starting the agents.

```
1• import jade.core.Agent;
2 import jade.core.behaviours.Behaviour;
3 import jade.core.behaviours.ParallelBehaviour;
4 public class BehaviourComplex extends Agent
5 {
6     private static final long serialVersionUID = 1L;
7
8•     protected void setup()
9     {
10         ParallelBehaviour parab=new ParallelBehaviour();
11         parab.addSubBehaviour(langhelloBehaviour(" السلام "));
12         parab.addSubBehaviour(langhelloBehaviour(" Bonjour "));
13         parab.addSubBehaviour(langhelloBehaviour(" buenos dias "));
14         parab.addSubBehaviour(langhelloBehaviour(" Maraba "));
15         System.out.println();
16         addBehaviour(parab);
17     }
18•     private Behaviour langhelloBehaviour(String msg)
19     {
20         Behaviour b= new Behaviour()
21         {
22
23             private static final long serialVersionUID = 1L;
24
25             int i=0;
26             @Override
27             public void action()
28             {
29                 // TODO Auto-generated method stub
30                 System.out.println("affichage du msg"+msg);
31
32                 i++;
33             }
34
35             @Override
36             public boolean done()
37             {
38                 // TODO Auto-generated method stub
39                 return i>=3;
40             }
41
42         };
43         return b;
44     }
45 }
```

Agent Setup: The setup method is called when the agent is initialized.

ParallelBehaviour: A ParallelBehaviour parab is created to allow multiple behaviors to run concurrently.

Adding Sub-Behaviours: The parab object has several sub-behaviours added to it, each created by the langhelloBehaviour method, which takes a greeting message as a parameter.

Print Agent Information: The agent prints its name and address, indicating it is executing multiple behaviors in parallel.

Adding Behaviour: The parab (which includes all the sub-behaviours) is added to the agent.

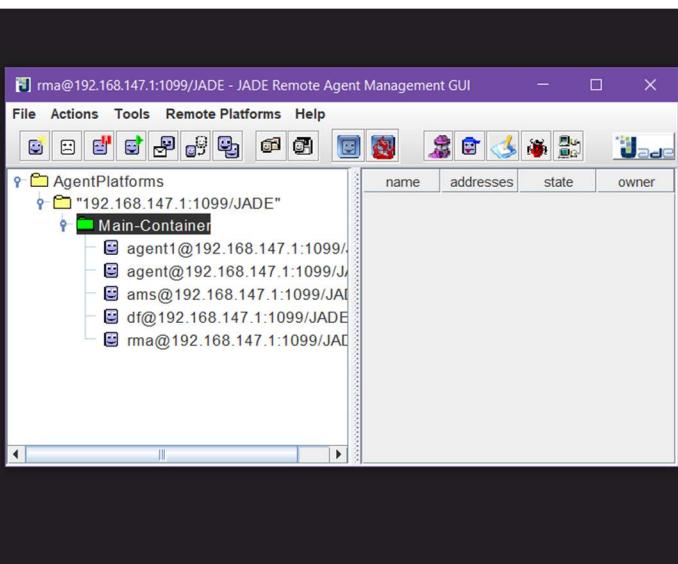
langhelloBehaviour Method

Behaviour Definition: The method returns a new Behaviour object.

Action Method: The action method prints the agent's name, the message, and the current iteration number (formatted as (i+1) to start from 1).

Done Method: The done method returns true when the message has been printed three times, stopping the behavior.

Output:



The screenshot shows the JADE Remote Agent Management GUI window. The title bar reads "rma@192.168.147.1:1099/JADE - JADE Remote Agent Management GUI". The menu bar includes File, Actions, Tools, Remote Platforms, and Help. The toolbar contains various icons for managing agents. The main pane displays a tree view of agents under "AgentPlatforms" and a table view of agents under "Main-Container". The tree view shows nodes for "192.168.147.1:1099/JADE" and "Main-Container", with "agent1@192.168.147.1:1099/JADE", "agent@192.168.147.1:1099/JADE", "ams@192.168.147.1:1099/JADE", "df@192.168.147.1:1099/JADE", and "rma@192.168.147.1:1099/JADE". The table view has columns for name, addresses, state, and owner. On the left, a terminal window shows repeated messages: "affichage du msg Maraba", "affichage du msg Bonjour", and "affichage du msg msgbuenos dias".

```
affichage du msg Maraba
affichage du msg Bonjour
affichage du msg msgbuenos dias
affichage du msg Maraba
affichage du msg Bonjour
affichage du msg msgbuenos dias
affichage du msg Maraba
affichage du msg Maraba
affichage du msg Bonjour
affichage du msg msgbuenos dias
affichage du msg Maraba
affichage du msg Maraba
affichage du msg Maraba
affichage du msg Bonjour
affichage du msg msgbuenos dias
affichage du msg Maraba
affichage du msg Bonjour
affichage du msg msgbuenos dias
affichage du msg Maraba
affichage du msg Maraba
```

In this code we created and run multiple agents in JADE, each with multiple concurrent behaviors. Each behavior repeatedly prints a hello message three times.

Conclusion

Through these exercises, we have explored different aspects of agent behaviors in JADE. We learned how to migrate agents between containers, implemented and tested various primitive behaviors, and created agents with composite behaviors that run concurrently.