KTH- ID 2209 – Distributed Artificial Intelligence and Autonomous Agents

Course Coordinator: Mihhail Matskin

Teacher Assistants: Shatha Jaradat & Lorenzo Corneo

HOMEWORK 1 – Group : Alessio Russo, Mumtaz Fatima

To deal with the assigned problem first we had to define how to store/save the user profile and the various artifacts of the different types of museum.

To do so we used as storage type flat files, which are formatted in the following way:

For the user (each field is a string, without brackets. fields are separated by a space):
[AGE] [OCCUPATION] [GENDER] newline
[INTEREST 1] [INTEREST 2]....[INTEREST N] newline
[ARTIFACT SEEN 1]... [ARTIFACT SEEN M]

For a museum each line corresponds to an artifact (except the first line, the name of the museum) The line is formatted in the following way:
[ID][ARTIFACT NAME][ART. CREATOR][DATE][PLACE][TYPE][SUBTYPE][DESCRIPTION]

When the profiler starts it loads the user profile. When a curator starts, based on the argument passed to the curator, loads the specified museum file.

PROFILER

The profiler first loads the user profile (line 28). The user info is displayed (line 29).
In line 30 and 50 we have a ticker behaviour, that every 10 seconds check for new curators or new tour guides registered at the DF.
After it subscribes to the DF whenever a tour guide registers at the DF.

```
DFAgentDescription dfd = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType("Virtual-Tour");
    dfd.addServices(sd);
    SearchConstraints sc = new SearchConstraints();
    sc.setMaxResults(new Long(1));
```

```
send(DFService.createSubscriptionMessage(this, getDefaultDF(),
                                         dfd, sc));
addBehaviour(new RegistrationNotification());
```

When that happens, RegistrationNotification (line 86) is called, which is a Cyclic behaviour, that handles the notification. In our case it sends a message to the Tour guide agent.
Ticker,Wacked,Oneshot,Cyclic behaviour were used.

CURATOR

The curator first loads the museum file, the register its service to the DF. Then, thanks to a Cyclic Behaviour, waits for messages. It can wait for two kinds of messages, one where details about a single artifact is requested, and the other one where all the artifacts of the museum are requested.
To request for a single artifact a message, with content “ARTIFACT INFO NAME” has to be sent, where NAME is the name of the artifact.

For all the artifacts a message with content “MUSEUM INFO” has to be sent.

TOUR AGENT GUIDE

The agent firsts register its service to the DF. Then, thanks to a TickerBehaviour, every 10 seconds updates the list of the available curators. Every time a curator is added, a request for a NULL artifact and for the artifacts of the museum is sent.
Next the agent loads a MSGReceiver behaviour, that is used to receive messages from the profiler. Whenever a message is received, a sequential behaviour is started: first a parallel behaviour is executed, where for each subbehaviour a list of artifacts from a certain curator is requested. Once all the artifacts are received, based on the user profile (not implemented yet) a tour is sent, which is a oneshotbehaviour.

To run the program use the following arguments:
-gui -agents
"user:homework1.Profiler;cur1:homework1.Curator(museum1.txt);ag1:homework1.TourGuideAg
ent"

OLD CODE

1.Implementation of Agent Behaviour

1. Simple Behaviour

- **One Shot Behaviour**
  Displaying welcome message to user.

  ```
  class MyOneShotBehaviour extends OneShotBehaviour {
  public void action() {
  System.out.println("Welcome Message.");     }
  }
  ```

- **Generic Behaviour**
  Display user information for e.g. Name, age, gender, occupation and interest. Works like a cyclic behaviour which continues execution until last condition is not false.

  ```
  class FiveStep extends SimpleBehaviour
  {
  public void action()
  {
  block(1000);
  System.out.println("Name:");
  block(1000);
  System.out.println("Age:");
  block(1000);
  System.out.println("Gender:");
  block(1000);
  System.out.println("Occupation:");
  block(1000);
  System.out.println("Interest:");
  finished = true;
  ```

```
}
 private boolean finished = false;
public boolean done() { return finished;}
}
```

- **Ticker Behaviour**
  A remainder which displays message after every 10 sec.

```
addBehaviour(new TickerBehaviour(this, 10000) {
        protected void onTick()
        {
           System.out.println("Agent will terminate after 50 second.");
              }
        });
```

- **Waker Behaviour**
  Agent will be terminated after 50 sec after displaying termination message.

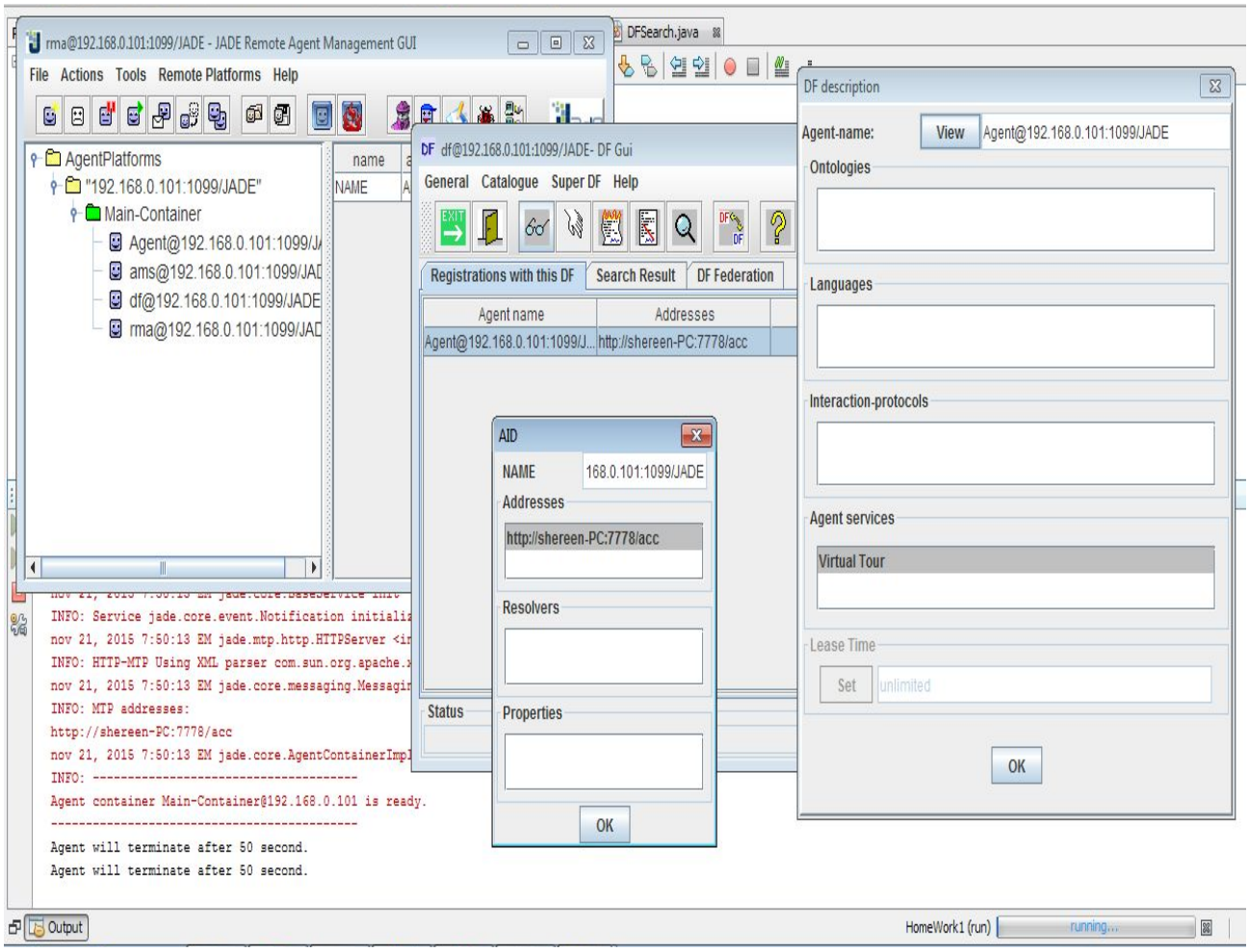
```
addBehaviour(new WakerBehaviour(this, 50000)
        {
           protected void handleElapsedTimeout() {
           System.out.println("Timeout! Agent has been terminated.");
           doDelete();
           }
        });
```

2.Implementing DF Agent

- **Agent service registration in DF**
  Agent want to register as Requester and provide Virtual Tour service.

```
ServiceDescription sd  = new ServiceDescription();
        sd.setType( "Requester" ); //Representing service type
        sd.setName("Virtual Tour");
        register( sd );

void register( ServiceDescription sd)
   {
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        dfd.addServices(sd);
        try {
           DFService.register(this, dfd );
        }
        catch (FIPAException fe) { fe.printStackTrace(); }
   }
```



- **Search Agent**

```
Search for Requester
ServiceDescription sd  = new ServiceDescription();
        sd.setType( "Request" );
        sd.setName("Virtual Tour");
        register( sd );

        try {
                        DFAgentDescription dfd = new DFAgentDescription();
                        DFAgentDescription[] result = DFService.search(this, dfd);

                        System.out.println("Search returns: " + result.length + " elements"
);
                        if (result.length>0)
                                System.out.println(" " + result[0].getName() );
                     }
```

```
    catch (FIPAException fe) { fe.printStackTrace(); }

                System.exit(0);
     }
void register( ServiceDescription sd)
{
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(getAID());
    dfd.addServices(sd);

    try {
       DFService.register(this, dfd );
    }
    catch (FIPAException fe) { fe.printStackTrace(); }
}

       AID getService( String service )
       {
              DFAgentDescription dfd = new DFAgentDescription();
              ServiceDescription sd = new ServiceDescription();
              sd.setType( service );
              dfd.addServices(sd);
              try
              {
                     DFAgentDescription[] result = DFService.search(this, dfd);
                     if (result.length>0)
                            return result[0].getName() ;
              }
              catch (Exception fe) {}
       return null;
       }
```

References

1. http://www.iro.umontreal.ca