Homework 2: Smart Museum

Weherage, Pradeep Peiris

November 29, 2015

1 Introduction

Smart Museum is a simple Agent framework build on top of JADE. In homework 1 the main Agents and their Behaviors was identified. The focus on homework 2 is to improve the communication in between agents. Also a new use case is introduced in this homework, that is an Auction in between an Artist Agent and Curator Agents.

2 Agents and Communications

Profiler Agent, Tour Guide Agent and Curator Agents are the main agents focused in this exercise. The basic functionality of Profiler Agent is to maintain the profile of the user and interact with Tour Guide Agent to get recommended Tours. Tour Guide Agent maintains a catalog of artifact on different Genre based on information retrieved from Curator Agent. Curator Agent manages the Artifacts and publish their information.

2.1 Profiler Agent

The profiler Agent interacts directly with Tour Guide Agents to get personalized virtual tours.

The communication with TourGuide Agent is implmented within **TourGuideManager**, which is a general Behavior. It starts a communication with TourGuide Agents for recommended tours. The communication is handled in two steps. First, **ACLMessage.CFP** message is sent to all TourGuide Agents, and reply from TourAgents are captured in second step and display recommended tours.

```
public void action() {
   switch (step) {
   case 0:
      log.info("Step 0. Propose for Tour Recommendation");
      ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
```

```
      for(AID tourAgent : tourGuideAgents) {
         cfp.addReceiver(tourAgent);
      }
      cfp.setContent(userProfile.getInterests());
      cfp.setConversationId("tour-recommend");
      cfp.setReplyWith("cfp" + System.currentTimeMillis());
      myAgent.send(cfp);

      step = 1;
      break;
   case 1:
      log.info("Step 1. Receive Tour Recommendation");
      MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.PROPOSE);
      ACLMessage reply = myAgent.receive(mt);
      if (reply != null) {
         gui.updateTourSuggestions(reply.getContent());
      } else {
         block();
      }
      step = 2;
      break;
  }
}
.
```

2.2 Tour Guide Agent

Tour Guide Agent registers itself in DF at Agent setup (*Agent.setup()*) and start searching for Curator Agents for every 5 seconds with a **TickerBehaviour**. It sends a request for artifact information for each found Curator Agents.

```
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("Publish-curator");
template.addServices(sd);

DFAgentDescription[] result = DFService.search(myAgent, template);
curatorAgents.clear();
artifactGenre.clear();

for (int i = 0; i < result.length; ++i) {
   curatorAgents.add(result[i].getName());
   myAgent.addBehaviour(new OneShotBehaviour() {
```

```
      public void action() {
        ACLMessage cfp = new ACLMessage(ACLMessage.REQUEST);
        for(AID curator : curatorAgents) {
           cfp.addReceiver(curator);
        }

        cfp.setConversationId("curator-artifact");
        cfp.setReplyWith("cfp" + System.currentTimeMillis());
        myAgent.send(cfp);
      }
   });
}
```

It has two other main behaviors, CuratorManager and RecommendationManager which both are **CyclicBehaviours**. CuratorManager wait for the reply message from Curator Agent which was requested for Artifacts information. The resply from Curator Agent contains all its artifacts details with genre. CuratorManager takes the reply and updates its Artifacts catalog against genre. RecommendationManager waits for tour recommendation requests from Profiler Agents. The request from Profiler agent contains its interest, which will be compared with Artifact genre catalog for recommendation. The reply to Profiler Agents contains all the information of Artifacts.

2.3 Curator Agent

The **CyclicBehaviour** of ArtifactRequest listens on requests from Tour-Guide Agents and reply with all artifacts infomration contains in Artifact Catalog.

```
private class ArtifactRequest extends CyclicBehaviour {
   private MessageTemplate mt =
      MessageTemplate.MatchPerformative(ACLMessage.REQUEST);

   public void action() {
      ACLMessage msg = myAgent.receive(mt);
      if (msg != null) {
         log.info("Receive Artifact Rrequest");

         ACLMessage reply = msg.createReply();
         reply.setContent(createArtifactsList());
         myAgent.send(reply);
      } else {
         block();
```

```
        }
    }

    private String createArtifactsList() {
        JSONArray list = new JSONArray();
        for(Artifact a : artifacts.values()) {
            list.add(a.getAttributes());
        }
        return list.toJSONString();
    }
}
```

3 Dutch Auction Interaction Protocol

The Auction between Artist Agent and Curator Agents is implemented using Dutch Auction Interaction Protocol.

3.1 Artist Agent

The main functionality of Artist Agent is to play the Auctioneer role in Dutch Auction Protocol. As it is in Figure1, Artist Agent initiate an auction providing detail of the item with Start price and Reserved Price.

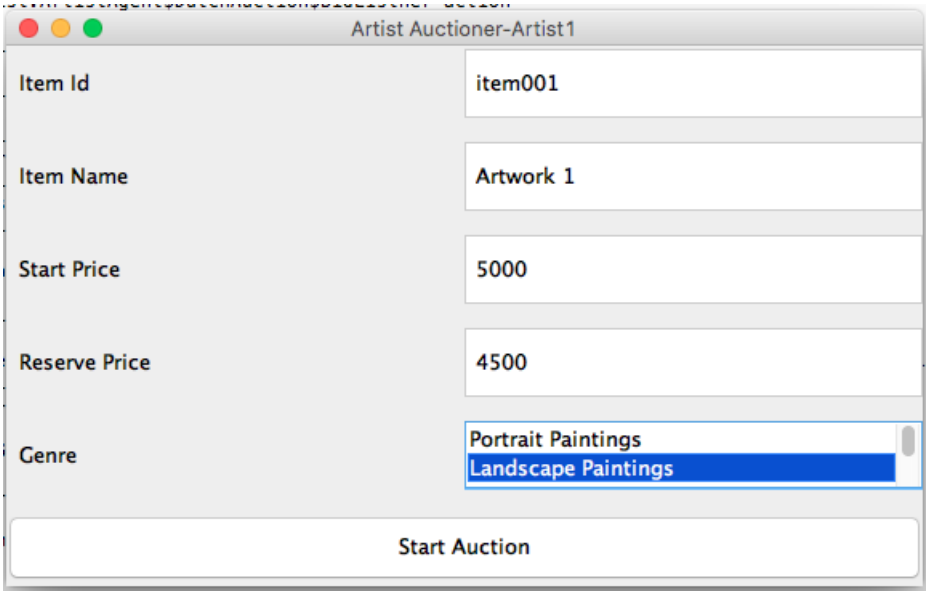

Figure 1: Artist Agent GUI

The Auction flow is managed in JADE FSMBehaviour, where the aution is considered as a state machine.

```
registerFirstState(new AuctionStart(auctionItem), STATE_START);
registerState(new PriceOffer(auctionItem), STATE_OFFER);
registerState(new BidListner(), STATE_LISTEN);
registerLastState(new CloseAuction(), STATE_CLOSED);

registerDefaultTransition(STATE_START, STATE_OFFER);
registerTransition(STATE_OFFER, STATE_LISTEN, TRANS_OFFER_TO_LISTEN);
registerTransition(STATE_OFFER, STATE_CLOSED, TRANS_OFFER_TO_CLOSE);
registerTransition(STATE_LISTEN, STATE_OFFER, TRANS_LISTEN_TO_OFFER);
registerTransition(STATE_LISTEN, STATE_CLOSED, TRANS_LISTEN_TO_CLOSE);
```

That is, it has 3 main states, AuctionStart, PriceOffer, BidListner and CloseAuction.

AuctionStart is a OneShotBehaviour which inform about the auction item and price level to all registered Curator Agents.

Then it moves to PriceOffer state, which is also a OneShotBehaviour. PriceOffer state informs the current price to all the participants (Curator Agents), reducing price from the previous price. The current price is always checked against the reserved price, and change state to CloseAuction when it reaches the reserved price level.

BidListner state listens on the reply from all participants. Then it checks for the first "Accept" reply from a participant, if no "Accept" reply found it changes its state again to PriceOffer state, otherwise Auction will be closed and send the "Winner" message to who accept the auction price. All the other participant send with "Auction Closed" message.

4 Strategies of Agents

Artist agent's strategy is to sell high or low quality items. As selling low quality iteam has better payoff for Artist, the preference value 2 is given for selling high quality items, whereas 1 is given for selling low quality items.

Curator Agent's strategy is to increase or decrease quote price for items based on the demand. Increase quote price better payoff for Curator agent. Therefore, preferance value 2 is given for increasing quote price, and 1 for decreasing quote price.

Profiler Agent's strategy is to View or Buy an item. Also 'Doesn't view' is also considered as a strategy for Profiler agent. Buying an item has better payoff for Profiler agent than Viewing. Preference value 4 is given for Buying high quality product at decrease of quote price. (It was assume after buying an item, profiler knows quality of the item). Value 3 is given for buying an

item at increase of quote price. Viewing a high quality item has preference over viewing an low quality item.

Figure 2 represents the Scenario Matrix of different strategies for Agents. Figure 3 shows the Payoff matrix. There is a Nash Equilibrium highlighted in Payoff matrix. That is, when Profiler Agent is buying a low quality item at increase quote price.

		Artist Agent			
		Sell High Quality		Sell Low Quality	
		Curator Agent		**Curator Agent**	
		Increase Quote price	Decrease Quote price	Increase Quote price	Decrease Quote price
Profiler Agent	View	*Artist has decided to sell high quality product, and Curator quotes high price. Profiler views it.*	*Artist has decided to sell high quality product, and Curator quotes low price. Profiler views it.*	*Artist has decided to sell low quality product, and Curator quotes high price. Profiler views it.*	*Artist has decided to sell low quality product, and Curator quotes low price. Profiler views it.*
	Not View	*Artist has decided to sell high quality product, and Curator quotes high price. Profiler doesn't views it.*	*Artist has decided to sell high quality product, and Curator quotes low price. Profiler doesn't views it.*	*Artist has decided to sell low quality product, and Curator quotes high price. Profiler doesn't views it.*	*Artist has decided to sell low quality product, and Curator quotes low price. Profiler doesn't views it.*
	Buy	*Artist has decided to sell high quality product, and Curator quotes high price. Profiler buy it.*	*Artist has decided to sell high quality product, and Curator quotes low price. Profiler buy it.*	*Artist has decided to sell low quality product, and Curator quotes high price. Profiler buy it.*	*Artist has decided to sell low quality product, and Curator quotes low price. Profiler buy it.*

Figure 2: Scenario Matrix

		Artist Agent			
		Sell High Quality		Sell Low Quality	
		Curator Agent		**Curator Agent**	
		Increase Quote price	Decrease Quote price	Increase Quote price	Decrease Quote price
Profiler Agent	View	*2, 2, 1*	*2, 1,1*	*1, 2, 2*	*1, 1, 2*
	Not View	*0, 2, 1*	*0, 1,1*	*0, 2, 2*	*0, 1, 2*
	Buy	*3, 2, 1*	*4, 1,1*	*1.5, 2, 2*	*2, 1, 2*

Figure 3: Payoff Matrix

5 Implementation

The implementation of Smart Museum framework including all its Agents with GUI are available at https://github.com/wpnpeiris/smartmuseum

5.1 Technologies

1. JADE (http://jade.tilab.com/)
2. JSON Simple (https://code.google.com/p/json-simple/)
3. Maven (https://maven.apache.org/)

5.2 Prerequisite

1. Maven 3.x as the build tool
2. Java 7

5.3 Build the Application

1. mvn package

5.4 Start Main Container

1. java -jar target/smartmuseum-1.0.0.jar -gui

5.5 Start Profiler Container

1. java -jar target/smartmuseum-1.0.0.jar -container -host localhost Profiler1:kth.id2209.profiler.ProfilerAgent

5.6 Start Curator Container

1. java -jar target/smartmuseum-1.0.0.jar -container -host localhost Curator1:kth.id2209.curator.CuratorAgent

5.7 Start TourGuide Container

1. java -jar target/smartmuseum-1.0.0.jar -container -host localhost TourGuide:kth.id2209.tourguide.TourGuideAgent