Homework 1: Smart Museum

Weherage, Pradeep Peiris

November 22, 2015

1 Introduction

Smart Museum is a simple Agent framework build on top of JADE. The main task of this homework is to identify the main Agents and their Behaviors. Also to use Directory Facilitator (DF) Agent in JADE to register and discover Agent's services.

2 Agents and Behaviors

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

There are two main functionalities in Profiler Agent. Managing user's profile and Tour Requests.

User is provided with a simple UI to get and update user's information. The UI is initiated at Profiler Agent setup phase (*Agent.setup()*). (See Figure 1). Also at Agent setup, Profiler Agent add a **TickerBehaviour** which search for TourGuide Agents in Directory Facilitator (DF) at each 5 seconds. Any newly identified TourGuide Agents are added into its TourGuide agent list.

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

DFAgentDescription[] result = DFService.search(myAgent, template);
```

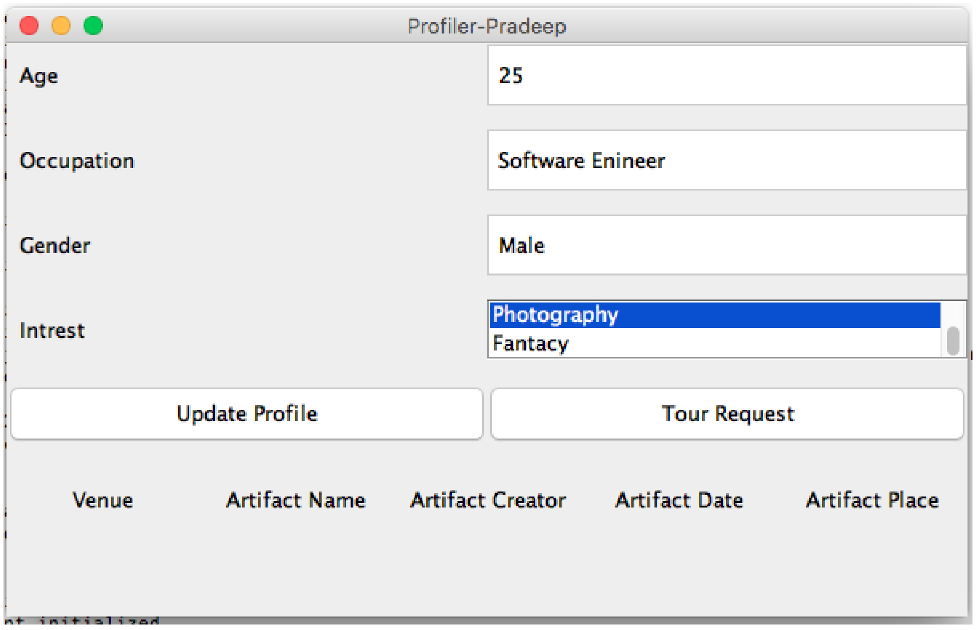


Figure 1: Profiler Agent GUI

```
for (int i = 0; i < result.length; ++i) {
   if(tourGuideAgents.contains(result[i].getName())) {
      log.info("TourGuideAgent " + result[i].getName() + " is already in list");
   } else {
      log.info("TourGuideAgent " + result[i].getName() + " is added to list");
      tourGuideAgents.add(result[i].getName());
   }
}
```

Chnages on the user profile are handled in ProfileManager Behavior. ProfileManager is a **OneShotBehaviour** which is invoked upon UI update ("Update Profile" button).

TourGuideManager is a general Behavior in Profiler Agent which start 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.

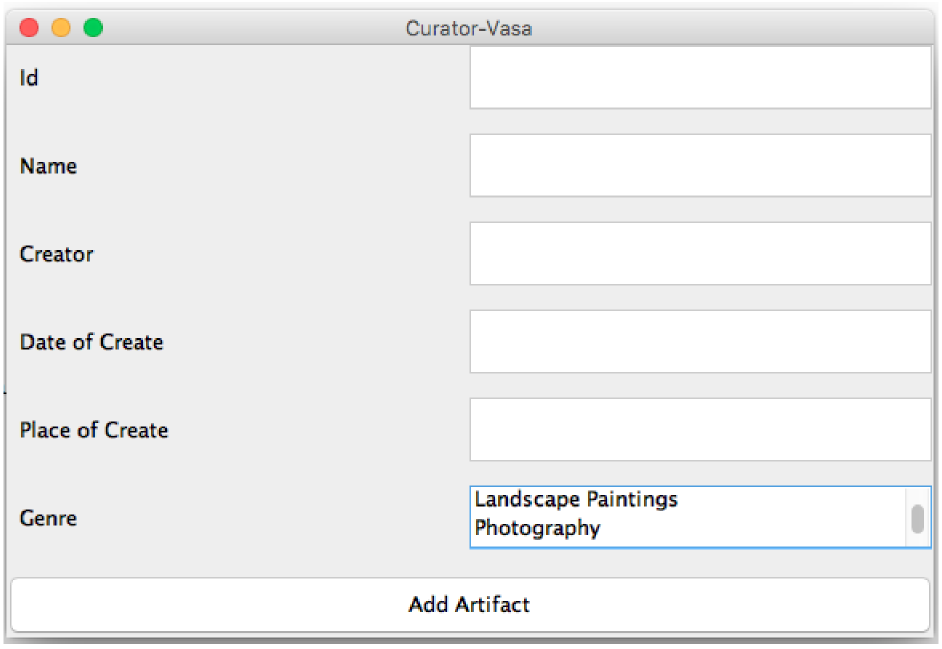
2.3 Curator Agent

The main functionality of Curator Agents is to maintain its artifacts with detail information. Curator Agent is provided with a simple GUI to maintain artifact detail. (See Figure 2).

ArtifactManager in Curator Agent is a **OneShotBehaviour**, which updates its Artifact Catalog. The **CyclicBehaviour** of ArtifactRequest listens on requests from TourGuide 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");
```



Id	
Name	
Creator	
Date of Create	
Place of Create	
Genre	Landscape Paintings Photography

Figure 2: Curator Agent GUI

```
            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 Implementation

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

3.1 Technologies

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

3.2 Prerequisite

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

3.3 Build the Application

1. mvn package

3.4 Start Main Container

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

3.5 Start Profiler Container

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

3.6 Start Curator Container

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

3.7 Start TourGuide Container

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