Homework 3: Smart Museum and N-Queens Problem

Weherage, Pradeep Peiris

December 6, 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, and communication between Agents was improved in homework 2. The objective of homework 3 is to apply Agent Mobility in Dutch Auction Scenario.

2 Agent Mobility

Dutch Auction from previous home work is improved to run multiple Auctions in separate JADE containers. The JADE container in this sense represents different participant type of Auctions, each which can be function separately. Within a container, the participants of Auction will be clone of Curator Agents.

The Auctioneer, Artist Agent is resided in a septate container, and will be cloned and moved into Curator containers to run auctions in parallel.

Finally, Artist Agents are move back to the original container to evaluate the final result.

2.1 Curator Agent

Curator Agent UI is provided with an option to clone another Curator Agent.

It calls Curator Agent's *cloneCurator()* method when Clone button is pressed with provided container and name. (see Figure 1)

```
public void cloneCurator(String containerName, String curatorName) {
   ContainerID destination = new ContainerID();
   destination.setName(containerName);
   doClone(destination, curatorName);
}
.
```

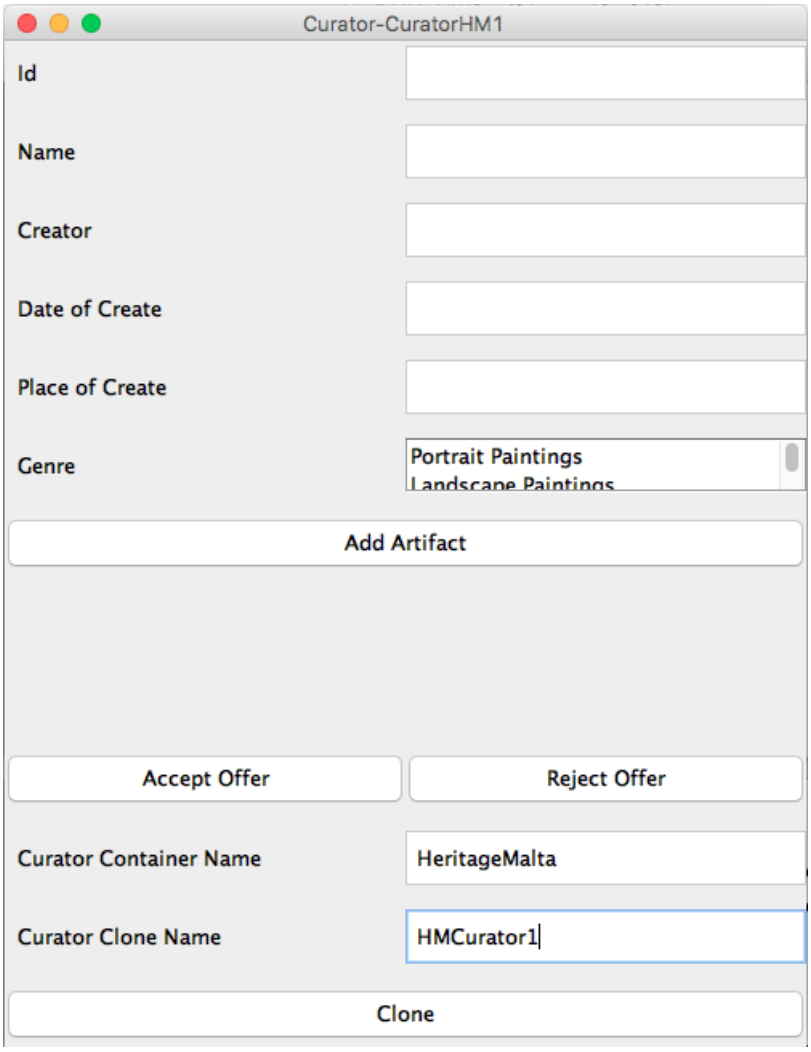

Figure 1: Curator Agent GUI

When *doClone(destination, curatorName)* is executed, initiation of behaviors and curator registration are handled overriding *afterClone()* in Curator Agent.

2.2 Artist Agent

The Auctioneer in our scenario is Artist Agent. It is improved to move in between containers. In this exercise, Artist Agent is provided with an UI to clone multiple Artist Agent and move them around containers (see Figure 2)

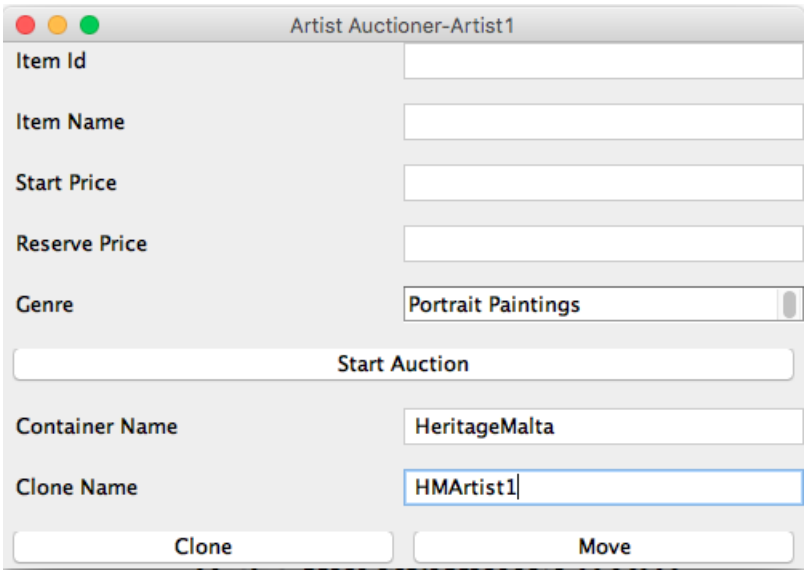

Figure 2: Artist Agent GUI

For each UI option *Clone* and *Move*, methods *cloneArtist()* and *moveArtist()* are called.

```
public void cloneArtist(String containerName, String curatorName) {
   ContainerID destination = new ContainerID();
   destination.setName(containerName);
   doClone(destination, curatorName);
}

public void moveArtist(String containerName) {
   ContainerID destination = new ContainerID();
   destination.setName(containerName);
   doMove(destination);
}
```

2.3 Experiment

As an experiment for the use case, two container *HeritageMalta* and *MuseGalileo* was initiated with one Curator Agent within it. Then Curator Agents were increased cloning them.

The Auctioneer, Artist Agent is created in another container, and a cloned of it moved into *HeritageMalta* and *MuseGalileo* container separately to run parallel auctions.

The setup can be viewed in JADE Managment GUI as in Figure 3.

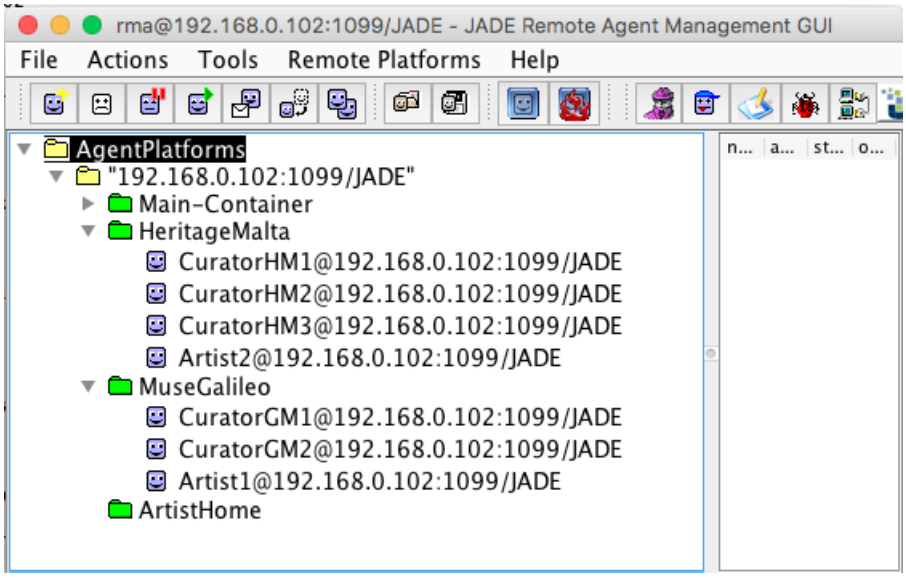

Figure 3: JADE Management GUI

3 N-Queen Problem

The aim of this task is to place N number of Queens on N by N chessboard, so that no two queens threatens each other.

This is archived in JADE considering Queens and Chessboard as Agents, which communicate with each other to achieve the following goal, to place queens on the chessboard so that no conflicts with queens.

3.1 Chessboard Agent

Chessboard Agent is the entry point to the solution. There is only one Chessboard Agent and it is setup with given dimension. That is, if N is the given input value it setups a Chessboard Agent with N dimension and create N number of Queen Agents.

Each Queen is assigned to a column and allow only moving on it. Queens Agent sends a message to Chessboard Agent to update the board when it finds a stable position.

3.2 Queen Agent

Queen Agents are setup by Chessboard Agent and assigned to column on which only it allows to move. At its each move Queen comminicates with other Queens Agent to check the validity of the move.It is checked by send-

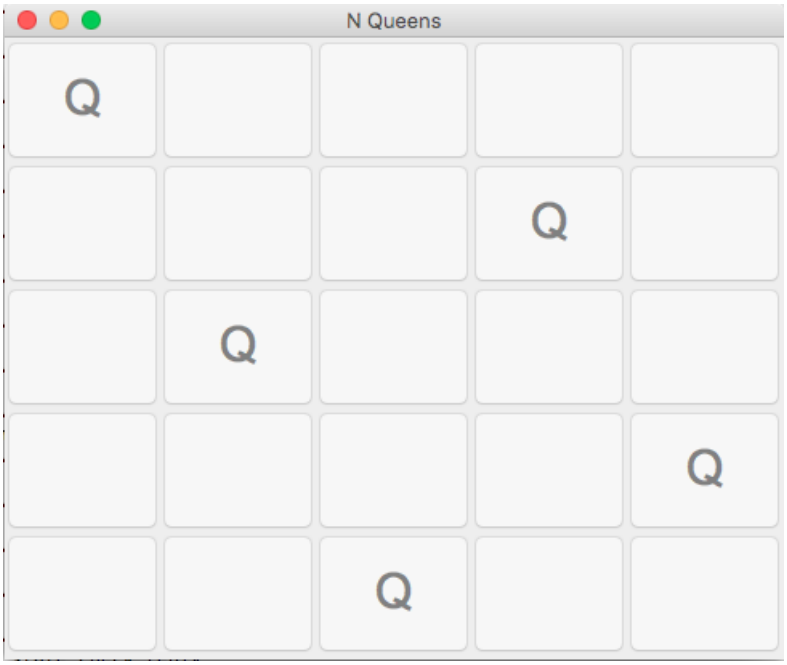

Figure 4: N-Queens Problem

ing *ACLMessage(ACLMessage.PROPOSE)* message. At received of this message, each Queen Agent check its validity with its position.

```
private boolean validCheck(CheckValue checkValue) {
   boolean valid;
   if(isPriorQueen(checkValue.col)) {
      valid = true;
   } else {
    valid = notInSameRow(checkValue.row)
              && notInSameDiagonal(checkValue);
    }

    return valid;
}
private boolean notInSameRow(int checkRow) {
   return (checkRow != currentRow);
}
private boolean notInSameDiagonal(CheckValue checkValue) {
   return (Math.abs(checkValue.row - currentRow) !=
   Math.abs(checkValue.col - queenId));
}
```

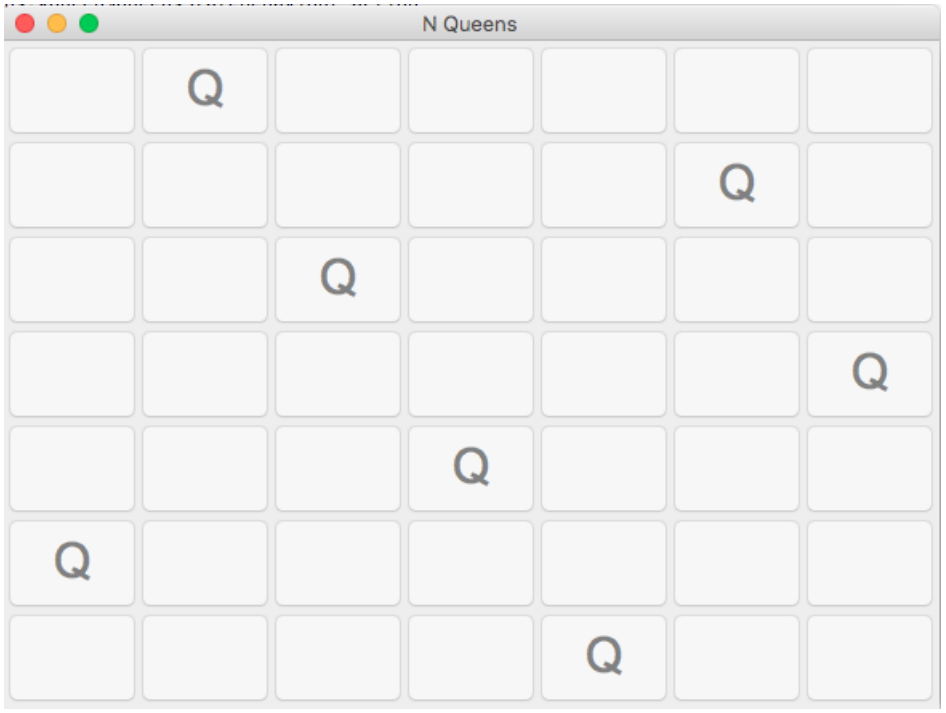

Figure 5: N-Queens Problem

Queen Agent mainly has used *PROPOSE* and *INFORM* message primitives. Messages they exchange mainly are Command, and with position information at message CHECK.

```
public interface ChessCommand {
   public static final String MOVE = "MOVE";
   public static final String CHECK = "CHECK";
   public static final String VERIFY = "VERIFY";
   public static final String RESET = "RESET";
}
```

4 Implementation

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

4.1 Technologies

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

4.2 Prerequisite

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

4.3 Build the Application

1. mvn package

4.4 Start Main Container

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

4.5 Start Curator Container

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

4.6 Start TourGuide Container

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