Hochschule Bonn-Rhein-Sieg, Autonomous Systems

Multi Agent Systems Exercise Sheet 1

Alexander Hagg, Obada Alaqtash

November 3, 2013

1 Exercise 2

1.1 Introduction

As part of the course in Multi Agent Systems of the BRSU University, an agent network is developed to solve the problem of job scheduling using agents as functional units. Later on this dynamic environment will allow a system to be rapidly reconfigurable and offer a degree of adaptivity needed in robotics as well as many other areas in computer science.

Jade offers a development and runtime framework enabling quick development of multiagent systems within the Java framework. Development was done using the Eclipse IDE, which enables the integration of the Jade library. Github is used as a versioning and cooperation tool.

The system is in a highly developmental state. The communication behaviours are not optimal and might not be implemented in the same way everywhere, as we are experimenting with different behaviours.

1.2 Structure

Three agents were developed:

- JobSupplierAgent, supplying a set of jobs and durations to the scheduler. Multiple job suppliers could be running on multiple systems, but for now just one agent is instantiated.
- SchedulerAgent, using a simple scheduling algorithm (ordering by ascending duration). The scheduler initiates a small conversation with the job supplier asking for a list of jobs.

- SchedulingVisualizerAgent. It asks the scheduler to send the current schedule and presents a Java Swing GUI representing schedule in a Gantt diagram.

Other classes:

- Job class, wrapping jobs as objects (with a name and a duration). This class might be redundant but it implements the Jade Leap Serializable interface that allows the ArrayList<Job> joblist to be sorted.
- SchedulingVisualizerGui class, containing the Java Swing elements.
- myReceiver class, containing a behaviour for receiving messages. This class will be removed later on as Jade already offers lots of standard behaviours that allow this functionality.
- RunJade class, containing the main() method that initiates the entire system. Rather than running the system from the command line, this class enables multi-platform compatibility.

1.3 Communication

The JobSupplierAgent's setup contains the instantiation of a Jade CyclicBehaviour that will run forever. The behaviour contains a call to the block() method. As such, the action() method will always be called on a new event (such as reception of a message from another agent). The behaviour reads the contents of the message and checks whether the type is "QUERY_REF", which is a Jade constant representing a query message. After this it will check whether the query message contains the "joblist" string, which ensures that the agent received a query asking for the job list. Now it will create a reply message containing the ArrayList<Job> of jobs.

The SchedulingVisualizerAgent uses a TickerBehaviour, which is also cyclic. The difference to CyclicBehaviour is the fact that TickerBehaviour is always running at a certain tick rate and does not need to be awakened by an event. This behaviour was chosen because the GUI needs to be updated in regular intervals, ensuring a live view of the schedules. The behaviour only sends messages to other agents and does not handle their replies, as we cannot be sure that any schedulers are active and reachable and do not want to create too much overhead. A CyclicBehaviour was implemented to allow handling of received messages in a similar way as the JobSupplierAgent.

The core of the system lies within the SchedulerAgent, which involves several behaviours. In the current version, a standard query message is sent to all other agents in the system, containing a joblist query. This is done as part of a SequentialBehaviour. The second SubBehaviour of this SequentialBehaviour is the scheduling algorithm itself. A CyclicBehaviour is added as well to enable handling of Visualizer queries.

1.4 Test results

Figures 1.1, 1.2 and 1.3 show the results using three different sets of jobs. We have not been able to test the complete functionality yet and we are not sure whether updates of

the joblist in a live system will be received in the GUI.

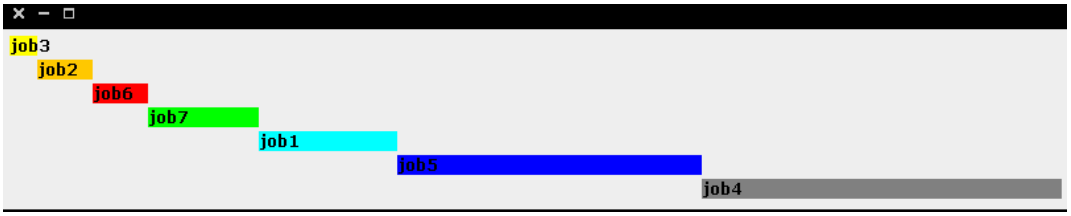

Figure 1.1: Output of the first joblist dataset

Figure 1.2: Output of the second joblist dataset

Figure 1.3: Output of the third joblist dataset

1.5 Bugs and things to do

- The schedule was implemented in a non-standard way, this will be corrected.
- Tests need to be done to check whether data flow behaves as expected.