MULTI-AGENT SYSTEMS COURSEWORK

Giovanni Paolini - 40276003

Paolini, Giovanni
40276003@napier.ac.uk

Introduction

The project aims to solve a supply chain problem that includes multiple entities communicating with each other. A manufacturing company is trying to maximise the profit assembling PCs for customers, the company has a storage for parts and needs to restock frequently to meet the customers demand. The parts to restock are bought from two different suppliers, the first one with higher price and a fast delivery time, the second one with lower price but slower delivery time.

With the implementation of Agents is possible to handle multiple asynchronous messages and coordinate all the tasks required to reach the aim of the problem. Being each Agent autonomous, once implemented the program can be run multiple times with different criteria. As (Hakansson 2013) states, multi agent programming is a very reliable and powerful way to run simulations, agents may be started on multiple machines increasing the dimension of the experiment exponentially, allowing to simulate scenarios that may not be possible to test with another approaches. (Sharma, Deepak, et al 2013) highlight that the proactivity of agents can be fundamental in simulations that aim to replicate real world scenarios and the autonomy that an agent can have allows it to perform complex tasks without any intervention from the programmer, saving time and resources. Considering that in the context of this experiment, there are multiple tasks to be performed, multiple communication channels to be setup and multiple entities that interact with each other, multi agent programming is a suitable solution to implement this simulation.

Model Design

The project includes 4 main Agents:

ClockAgent: The Clock Agent is used to keep track of the days in the experiment, its main task is to send a new day message to all the other agents so they know when to start a new day cycle. The clock agent also wait for end of day messages from the others agents, this is needed so that a new day is started only when all the other agents are done with their main tasks.

CustomerAgent: The Customer Agent simulates a customer that wants to place a PC order in the manufacturing shop. Its main task is to generate a random PC build that will be sent to the manufacturer with a desired quantity.

ShopAgent: The shop agent represents the manufacturing shop, it includes functions to handle the warehouse storage, calculate profit keeping track of the penalties, build PCs and ordering new required PC parts from the suppliers.

Supplier: There are two supplier agents, Supplier and Supplier2, both have the same functionalities, but different prices and delivery times. Their main task is answering to the ShopAgent orders and responding back with a completed restock order when the delivery wait is finished.

In [Appendix 1: figure1], the ontology diagram for the project is displayed.
From the top, we have Component. Component includes a serial number and it represents a PC part. Motherboard, CPU, HD (Hard Drive), and RAM all extend Component, having new attributes, specifically Motherboard and CPU will have a model number that can either be “Mintel” or “IMD”, while RAM and HD have a size attribute which identifies the capacity of the memory. All the new attributes for motherboard, CPU, HD and RAM are set as required.
Note that Component was implemented but not used in the code. This decision was taken thinking about further development of the simulation. A serial number could be used to identify specific pieces, keeping track of them during the experiment where a task would need this option. In the context of this experiment, keeping track of a specific component was not necessary, so a Component ontology was implemented for formality, serial number was set as not required and not used.

The concept PC includes one of each component and it represents the final product for a customer order. As the Component concept, PC has an ID for formality, but it was not necessary for this experiment.

The AgentAction SuppOrder includes a PC and has price and quantity as attributes. A SuppOrder action is generated when a Supplier received the parts ordered by the manufacturer, then sent it to the manufacturer as content of a message.

The AgentAction CustOrder includes a Motherboard, CPU, HD and RAM, while having price, quantity and days as additional attributes. Price indicates the amount of money a customer is willing to spend for that order, quantity is the quantity of machines with the same specifications wanted and days is the time frame the customer expects the order to arrive after placing an order. CustOrder also includes an AID, which is also used from the supplier when the manufacturer forwards the order to restock the components the customer AID is replaced by the manufacturer AID.

In [Appendix 2: figure 1] a Sequence Diagram that displays the communication between the agents is shown.
Communication between Agents:

1. The *Clock* sends an INFORM message to all the agents, informing them that a new day is starting.
2. The *Customer* Agent creates a CustOrder object, which includes a randomised PC and a quantity. The order is the sent via a REQUEST message to the *Manufacturer*.
3. The *Manufacturer* receives the REQUEST message from the customer, the order is analysed and if the profit from that specific order is considered good, the shop will either ACCEPT the order or REFUSE if the profit is too low.
4. If the *Manufacturer* accepted the order, it will proceed to order the required parts from the *Supplier*. In this scenario there are two different suppliers, so the order is sent only after considering the time frame and costs of the two. The order is sent as a CustOrder set as the content of the message.
5. When receiving the order, the *Supplier* will accept it and store it. In the graph we see that REFUSE is also an option, this is because in a real world scenario, the supplier won’t have an infinite stock of parts, so it could be possible for an order to be refused.

6. After the *Supplier* receives the parts, he will send them through a SuppOrder object to the *Manufacturer*
7. The *Manufacturer* receives the parts and fulfil the *Customer* order, sending the machines.
8. All the agents send an INFORM message to the clock once they finish all their tasks. When the clock receives a message from all the agents, a new day message will be forwarded.

Model Implementation

Behaviour description:

The Agent ClockAgent has a single custom behaviour SynchAgentsBehaviour(Line 69), It is composed by multiple steps switched by a switch case statement. The behaviour will start by sending a “New Day” INFORM message(Line 144) to all the agents registered in the yellow pages, this message is used to determine when a new day cycle is starting in the simulation. Then the behaviour will wait to receive an INFORM “done” message (Line 152) from all the Agents, this will indicate that all the Agents ended their tasks and a new day can be started. When reaching the last day, a “terminate” message will be sent to all the Agents to inform them that the simulation is over and they can call their delete method.

The Agent CustomerAgent has one Cyclic behaviour called TickerWaiter(Line 62), the Agent will listen for a new day message from the clock agent to start a new day routine. Inside the TickerWaiter behaviour there are two additional sub-behaviour: MakeOrder(Line 99) is a one shot behaviour that will generate and send a new order to the manufacturer, the method sends a REQUEST message containing the AgentAction CustOrder as content. EndDay(Line 179) is a one shot behaviour that sends a “done” inform message back to the clock agent, to inform it that the daily routine is concluded.

The Agent ShopAgent implements the same TickerWaiter behaviour and its sub-behaviours are EndDay and Daily(Line 199), a one shot behaviour that is used to call all the required daily tasks such as updating the delivery time on orders, manufacturing new PCs and calculating penalties from the warehouse storage. The method OrderComponents (Line 260) switches between the two suppliers and then send a REQUEST message with the AgentAction CustOrder as content to request parts from a specific supplier.

The Agents Supplier and Supplier2 implement the TickerWaiter behaviour and two sub-behaviours. The sub-behaviour SendParts(Line 139 and 139) is a one shot behaviour used to send a REQUEST message to the manufacture when a part order is ready to be delivered, using the AgentAction SuppOrder as content of the message.

Constraints:
1) A Mintel CPU must be paired with a Mintel motherboard, and an IMD CPU with an IMD motherboard.
This constraint is implemented in the Customer Agent (Line 114), a switch case with a random number generator will only generate these parts paired with the right combination.

2)The component delivery time from the two suppliers.
This constraint is implemented in both the suppliers(Line 98 and 98), when a new part order is received by the shop, the day counter is set to the right delivery time of each supplier.

3) The per-component per-day warehouse storage cost.
This constraint is implemented in the Agent ShopAgent in the Daily oneshot behaviour (Line 208). The function called to get the storage penalty cost is in the Warehouse Class(Line 17)

4) The maximum number of PCs that can be assembled on one day.
This constraint is implemented in the Agent ShopAgent in the Manufacture method (Line 401), the number of buildable PCs is checked every time the loops finds an order that could be built with the part in the warehouse.

5) An order can only be assembled if there are the components in the warehouse needed to build all of the PCs in the order
This constraint is implemented in the Agent ShopAgent Manufacture method (Line 398) and the function called canManufacture() is defined in the Warehouse class(Line 24)

6)Penalties for late delivery.
This constraint is implemented in the Agent ShopAgent Daily behaviour(Line 199) where the method OrderDayCounter (Line 387) is called. At the start of everyday, the method also checks if any of the order is late and applies the penalty for each late order.

7) Correct calculation of profit at the end of each day.
The penalties are detracted directly from the total profit, the option to display the daily profit is provided on Line 221 using the verbose variable to display more output in the console(Declared on top of each class). The total profit is all calculated in the ShopAgent class using the following values: Late delivery penalty (Line 386) , Warehouse storage cost (Line 208), supplies purchased (Line 146) and the earning from completing the order (Line 405)

Design of manufacturer agent control strategy

The experiment includes some constraints to try to maximise the profit of the manufacturer. Considering that penalties may be applied in different scenarios such as having too many orders in the order queue, having too many parts in the storage or not using the maximum building capacity of the facility, it is easy to understand that many things come in play when balancing the profit. The first approach in order to maximise the profit is for the manufacturer to accept only specific orders. (Claudia B. Gilbertson) states that the minimum acceptable net profit for a company should be between 19% and 21%. Given the constraints of the simulation, filtering for profits around that percentage will result in accepting more orders than what the manufacture can handle, ending in a negative profit caused by the penalties of late orders. To filter out some orders, the ideal percentage after trial and error was set to be around 40%. From 37% after many simulations it seems that the final profit is always positive. Another method implemented to maximise the profit is to use the maximum allowance of pc to be built in the most optimal way. All the orders are looped until no order which could be built with the parts in storage could fit in the quantity of buildable PCs.

Experimental Results

A total of 5 different experiments was conducted on the simulation.
As an overview, these are the criteria for each run:

1) Standard settings, all parameters according to the project specification
2) Increased the maximum amount of PCs that can be built to 100
3) Settings from experiment 2, with 4 Customers
4) New algorithm implemented in the order selection
5) Algorithm from experiment 4 with order limit increased from 50 to 80

The raw data is displayed in the following table:

Experiment 1	Experiment 2	Experiment 3	Experiment 4	Experiment 5		
461044	698669	281148	732191	791814		Maximum Value
451038	716620	301019	777576	811733		Minimum Value
433763	745943	237455	951406	871650		Average Value
307752	752844	318324	767065	783812		
302477	843392	230855	602245	908222		
456489	722911	725793	841484	831379		
508155	622626	411503	630622	882333		
496107	874945	632851	924456	795595		
555771	689967	238362	733267	917613		
513023	616804	443956	812384	945681		
443304	707422	497125	740451	761464		
724683	780186	495602	887599	710917		
556448	879068	420474	776485	801465		
412230	623933	609504	608381	701266		
473020.2857	733952.1429	417426.5	770400.8571	822496		

In the Experiment 1, no setting is changed and the minimum profit margin for the order selection is set to be 40%. Looking at the Minimum(Red) and Maximum(Green) values, it is clear that there is a big fluctuation in the results of this experiment. This is due to the penalty associated with the late orders, in the experiment there is no limit in how many orders the manufacturer could accept, the only filter is the minimum profit margin, which already excludes a big portion of the incoming orders. The average profit for this experiment is reported in yellow under the same column.

As stated in the previous section of the report, one of the biggest limitation of this simulation is the number of machines that can be built from the manufacturer each day. A low output capacity and a great number of incoming orders will result in an increase in penalties for late delivery. In Experiment 2 this number was increased to 100, doubling the previous simulation. After loosening this constraint, there is a substantial increase in all the profit for each run. The Minimum Value is already very close to the previous maximum value and the Average profit is a 43.23% increase from Experiment 1.

For Experiment 3 a new customer is added. At this point a cap for the orders is not implemented yet, so having 100 potential orders added to the previous ones has a big impact on the overall profit of the simulation. From the raw data it is clear that the penalties have increased and the manufacturer can't keep the pace with the incoming orders. The lowest Minimum Value was registered during this experiment, while the Maximum Value even being relatively high, it could be considered an outlier value keeping in consideration an Average Profit of 417426.5 . With this settings, Experiment 3 performed 54.982% worse than Experiment 2 and 12.4867% worse than Experiment 1

For Experiment 4 a new algorithm is implemented, including an order limit and filtering by plain profit value:

```
if(orders.size()<=50 ||(profit >20000))
if (profit >10000)
```

The algorithm limits the order list to 50 elements and filters the net profit to 10.000£. The OR statement is needed to avoid refusing a really profitable offer, even if it is exceeding the orders size.
As expected, after limiting the amount of orders and consequentially the penalties for late delivery the Average Value for this Experiment is almost double the one from Experiment 3, with the Maximum Value almost reaching 7 figures.

In the 5th and last Experiment the order limit was increased from 50 to 80. After running Experiment 4 it was noted that during the loop for the selection of which order could be built, it could happen that the build limit was too low and there wasn't an order that could fit the last slots, so building potential was wasted. By having 30 more orders, the manufacturer is not overwhelmed and has a wider choice to use

the 100 build slots to their full extent. The Average Value for this experiment is the highest one, a 6.541% increase from Experiment 4.

Conclusion

The simulation may be improved by further changing the various parameters, but being a controlled environment it is not realistic to expect the same efficiency in a real world scenario. Realistically Customers would tent to spend as low as possible for their orders, so new filters should be implemented, taking also into account that a costumer could request a build that is not possible to put together, breaking the constraint we set for the motherboard and CPU pairing. Also, when an order gets refused, in a real situation the customer may want to make a new order with different parts.
Suppliers won't have an infinite stock of parts, so there may be the necessity to look in to new suppliers, potentially increasing the PCs build costs and having different delivery times for components, which could result in additional late delivery penalties. Another thing to take into account when translating the problem in the real world is the fluctuation of prices, electronic components vary widely in price, changing their value overtime and that could also be caused by things not directly related to the PC market (For example the great increase in price of GPUs after BitCoin and BitCoin farms became popular.).
The design could be expanded including outsourcing, when the demand for a product is too high and the manufacturer can't keep up with it, it may be wise to settle for a lower profit and outsourcing the already built pc from other companies, this may help with the late delivery penalty and should only be an option when the demand is too high, the primary profit should come from the manufacturing of the machines.
One thing to look into to optimize the simulation is how orders are handled, by having a complex system of priority based on delivery date, profit and contact with the supplier the efficiency of the system may increase greatly, the main issue would be to determine which are the correct parameters and how to identify what is to be prioritised.
From the Experiment Results it is clear that even a small change can completely break the balance of the simulation. In a real world scenario it would not be possible to take the risk of going negative balance with trial and error, this is the main reason why having a solid simulation environment is necessary for a big company and multi agent programming provides the tools to simulate multiple levels of complexity while getting as close as possible to a real world scenario when programmed carefully.

References

1. Claudia B. Gilbertson, Mark W. Lehman (2008) *Fundamentals of Accounting: Course 1*, : Cengage Learning.[Pag.456]
2. Hakansson, Anne, and Ronald Hartung. *Agent and Multi-Agent Systems in Distributed Systems - Digital Economy and E-Commerce*. Springer Berlin Heidelberg, 2013.
3. Sharma, Deepak, et al. “Multi-Agent Modeling for Solving Profit Based Unit Commitment Problem.” *Applied Soft Computing*, vol. 13, no. 8, 2013, pp. 3751–3761., doi:10.1016/j.asoc.2013.04.001.

Appendix 1

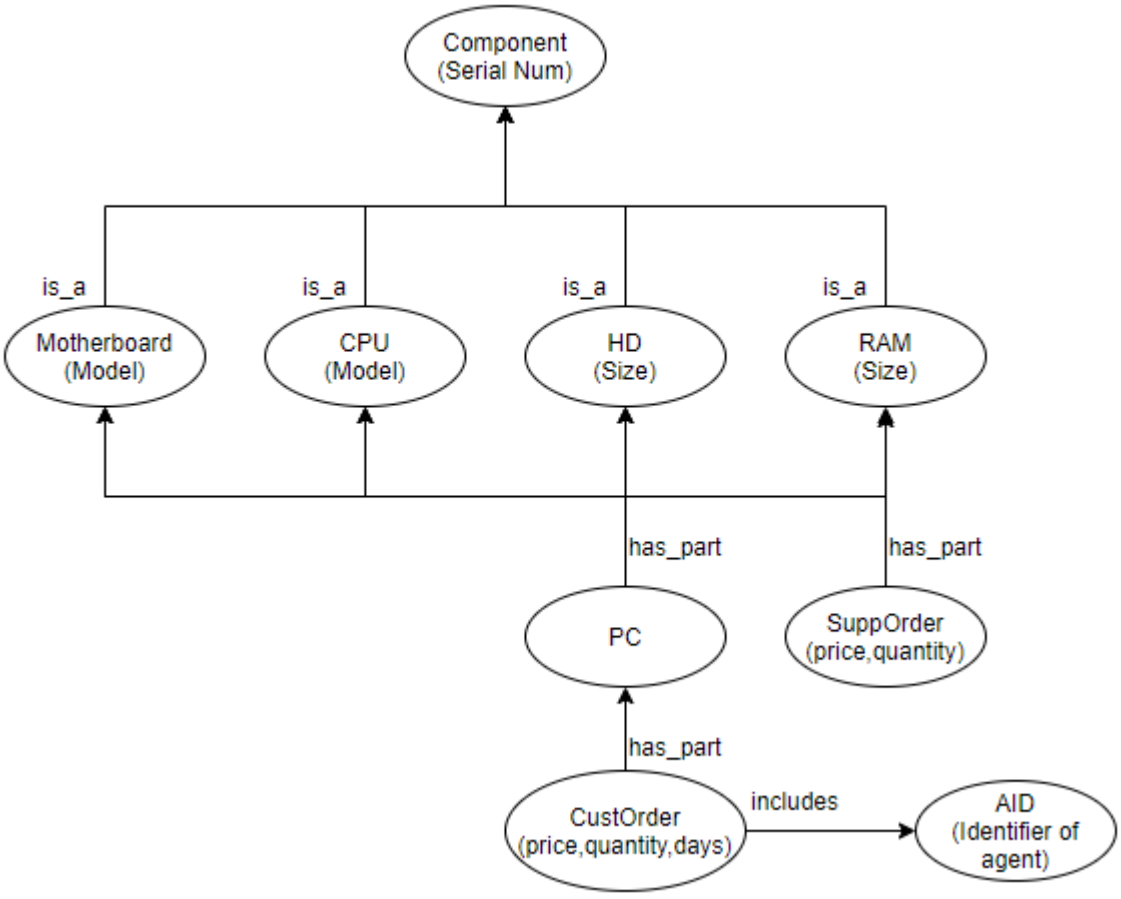

Figure 1: Ontology Diagram

Component Class: A concept class which implements a serial number, the serial number can be used to keep track of specific parts in the simulation. While this was not needed in the scope of the experiment, it is good practice to structure the ontology keeping in mind that the simulation could be expanded and in a work environment multiple programmers may use it, needing an unique value for a method implementation.

Motherboard Class: Extends Component, has a model string that contains the type of motherboard. The model is set as Mandatory.

CPU Class: Extends Component, has a model string that contains the type of CPU. The model is set as Mandatory.

HD Class: Extends Component, has a Size integer that contains the amount of memory of the part. The size is set as Mandatory.

RAM class: Extends Component, has a size integer that contains the amount of memory of the part. The size is set as Mandatory.

PC Class: Concept class that represents a finished product, includes one of each component part.

SuppOrder Class: An Agent Action that contains one of each component, price and quantity of an order. Is used by the Suppliers to send parts to the manufacturer.

CustOrder Class: An Agent Action that contains a PC, a price, a quantity and a delivery date. Is used by the customer class to place an order and by the manufacturer to request parts for a specific order.

Appendix 2

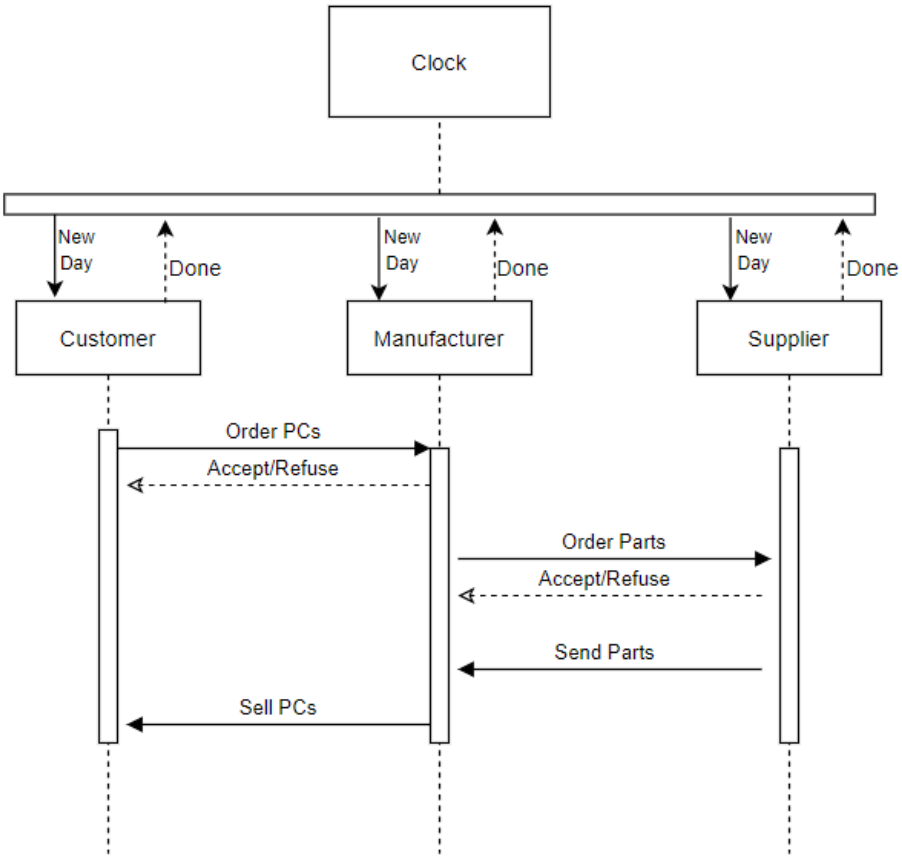

Figure 1: Sequence Diagram

Each day cycle starts with the Clock agent sending an INFORM message to all the Agents registered in the yellow pages, the message content is “New Day”.
Then the Clock waits for all the Agents to finish their daily routine, when they do they’ll send an INFORM message to the clock, the message content is “Done”

The customer sends a REQUEST message containing a CustOrder Agent Action to the Manufacturer, the CustOrder will include a PC, date, price and quantity.

The manufacturer accepts the order and sends a REQUEST message containing the CustOrder to one of the two suppliers.

The Supplier accepts the order and send a REQUEST message to the Manufacturer when the parts are ready do be delivered, the message content is a SuppOrder Agent Action, containing the parts, quantity and price of the order.

The PC is sold to the customer and profit calculated.

Appendix 3

Class ClockAgent:

```
1   package coursework;
2
3   import jade.content.Concept;
4   import jade.content.ContentElement;
5   import jade.content.lang.Codec;
6   import jade.content.lang.Codec.CodecException;
7   import jade.content.lang.sl.SLCodec;
8   import jade.content.onto.Ontology;
9   import jade.content.onto.OntologyException;
10  import jade.content.onto.basic.Action;
11  import jade.core.AID;
12  import jade.core.Agent;
13  import jade.core.behaviours.Behaviour;
14  import jade.core.behaviours.CyclicBehaviour;
15  import jade.lang.acl.ACLMessage;
16  import jade.lang.acl.MessageTemplate;
17  import jade.core.behaviours.TickerBehaviour;
18  import jade.domain.DFService;
19  import jade.domain.FIPAException;
20  import jade.domain.FIPAAgentManagement.DFAgentDescription;
21  import jade.domain.FIPAAgentManagement.ServiceDescription;
22
23  import java.util.ArrayList;
24  import java.util.concurrent.TimeUnit;
25
26  import coursework.pcshop_ontology.pcshopOntology;
27  import coursework.pcshop_ontology.elements.*;
28
29  public class ClockAgent extends Agent {
30
31        public static final int NUM_DAYS = 100;
32
33        // Change verbose value from 0 to 2 to change the amount of information output in
34  the console
35        private int verbose = 0;
36
37        @Override
38        protected void setup() {
39              if(verbose >0) {System.out.println("clock-agent " +
40  this.getAID().getName() + " is ready.");}
41              if(verbose==2) {System.out.println(this.getAID().getName() + " Registering
42  to the yellow pages");}
43              DFAgentDescription dfd = new DFAgentDescription();
44              dfd.setName(getAID());
45              ServiceDescription sd = new ServiceDescription();
46              sd.setType("clock");
47              sd.setName(getLocalName() + "-clock");
48              dfd.addServices(sd);
49              try {
50                    DFService.register(this, dfd);
51              } catch (FIPAException e) {
52                    e.printStackTrace();
```

```
53                }
54                // wait for the other agents to start
55                doWait(1000);
56                addBehaviour(new SynchAgentsBehaviour(this));
57          }
58
59          @Override
60          protected void takeDown() {
61                // Deregister from the yellow pages
62                try {
63                      DFService.deregister(this);
64                } catch (FIPAException e) {
65                      e.printStackTrace();
66                }
67          }
68
69          public class SynchAgentsBehaviour extends Behaviour {
70
71                private int step = 0;
72                private int numFinReceived = 0; // done messages from other agents
73                private int day = 0;
74                private ArrayList<AID> simulationAgents = new ArrayList<>();
75
76                public SynchAgentsBehaviour(Agent a) {
77                      super(a);
78                }
79
80                @Override
81                public void action() {
82
83                      switch (step) {
84
85                      case 0:
86                            try {
87                                  if(verbose>0)TimeUnit.MILLISECONDS.sleep(100);
88                            } catch (InterruptedException e1) {
89                                  e1.printStackTrace();
90                            }
91                            // find all agents from yellow pages
92                            DFAgentDescription template1 = new DFAgentDescription();
93                            ServiceDescription sd = new ServiceDescription();
94                            sd.setType("shop");
95                            template1.addServices(sd);
96                            DFAgentDescription template2 = new DFAgentDescription();
97                            ServiceDescription sd2 = new ServiceDescription();
98                            sd2.setType("supplier1");
99                            template2.addServices(sd2);
100                           DFAgentDescription template3 = new DFAgentDescription();
101                           ServiceDescription sd3 = new ServiceDescription();
102                           sd3.setType("customer");
103                           template3.addServices(sd3);
104                           DFAgentDescription template4 = new DFAgentDescription();
105                           ServiceDescription sd4 = new ServiceDescription();
106                           sd4.setType("supplier2");
107                           template4.addServices(sd4);
108                           //add all agents AID to simulationAgents list
109                           try {
110                                 DFAgentDescription[] agentsType1 =
111   DFService.search(myAgent, template1);
112                                 for (int i = 0; i < agentsType1.length; i++) {
```

```
113                                         simulationAgents.add(agentsType1[i].getName());
114   // this is the AID
115                                   }
116                                   DFAgentDescription[] agentsType2 =
117   DFService.search(myAgent, template2);
118                                   for (int i = 0; i < agentsType2.length; i++) {
119                                   simulationAgents.add(agentsType2[i].getName()); // this
120   is the AID
121                                   }
122                                   DFAgentDescription[] agentsType3 =
123   DFService.search(myAgent, template3);
124                                   for (int i = 0; i < agentsType3.length; i++) {
125                                   simulationAgents.add(agentsType3[i].getName()); // this
126   is the AID
127                                   }
128                                   DFAgentDescription[] agentsType4 =
129   DFService.search(myAgent, template4);
130                                   for (int i = 0; i < agentsType4.length; i++) {
131                                   simulationAgents.add(agentsType4[i].getName()); // this
132   is the AID
133                                   }
134                             } catch (FIPAException e) {
135                                   e.printStackTrace();
136                             }
137                             // send new day message to each agent
138                             ACLMessage tick = new ACLMessage(ACLMessage.INFORM);
139                             tick.setContent("new day");
140
141                             for (AID id : simulationAgents) {
142                                   tick.addReceiver(id);
143                             }
144                             myAgent.send(tick);
145
146                             step++;
147                             day++;
148                             break;
149
150                       case 1:
151                             // wait to receive a " done " message from all agents
152                             MessageTemplate mt = MessageTemplate.MatchContent("done");
153                             ACLMessage msg = myAgent.receive(mt);
154                             if (msg != null) {
155                                   numFinReceived++;
156                                   if (numFinReceived >= simulationAgents.size()) {
157                                         step++;
158                                   }
159                             } else {
160                                   block();
161                             }
162
163                       }
164                 }
165
166                 @Override
167                 public boolean done() {
168                       return step == 2;
169                 }
170
171                 @Override
172                 public void reset() {
```

```
173                     super.reset();
174                     step = 0;
175                     simulationAgents.clear();
176                     numFinReceived = 0;
177             }
178
179             @Override
180             public int onEnd() {
181
182                     if(verbose >0) {System.out.println("End of day " + day);};
183
184                     if (day == NUM_DAYS) {
185                             // send termination message to each agent
186                             ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
187                             msg.setContent("terminate");
188                             for (AID agent : simulationAgents) {
189                                     msg.addReceiver(agent);
190                             }
191                             myAgent.send(msg);
192
193                             myAgent.doDelete();
194                     } else {
195                             reset();
196                             myAgent.addBehaviour(this);
197                     }
198
199                     return 0;
200             }
201
202     }
203 }
204
```

Class CustomerAgent:

```
1  package coursework;
2
3  import jade.content.lang.Codec;
4  import jade.content.lang.Codec.CodecException;
5  import jade.content.lang.sl.SLCodec;
6  import jade.content.onto.Ontology;
7  import jade.content.onto.OntologyException;
8  import jade.content.onto.basic.Action;
9  import jade.core.AID;
10 import jade.core.Agent;
11 import jade.core.behaviours.CyclicBehaviour;
12 import jade.core.behaviours.OneShotBehaviour;
13 import jade.core.behaviours.SequentialBehaviour;
14 import jade.lang.acl.ACLMessage;
15 import jade.lang.acl.MessageTemplate;
16
17 import jade.domain.DFService;
18 import jade.domain.FIPAException;
19 import jade.domain.FIPAAgentManagement.DFAgentDescription;
20 import jade.domain.FIPAAgentManagement.ServiceDescription;
21 import coursework.pcshop_ontology.pcshopOntology;
22 import coursework.pcshop_ontology.elements.*;
23
24 public class customerAgent extends Agent {
25       // Change verbose value from 0 to 2 to change the amount of information output
26       // in the console
27       private int verbose = 0;
28       private Codec codec = new SLCodec();
29       private Ontology ontology = pcshopOntology.getInstance();
30       private AID shopAgent = new AID("shop", AID.ISLOCALNAME);
31       private AID tickerAgent;
32       private CustOrder custOrder;
33
34       protected void setup() {
35             getContentManager().registerLanguage(codec);
36             getContentManager().registerOntology(ontology);
37             if (verbose > 0)
38                   System.out.println("customer-agent " + getAID().getName() + " is
39 ready.");
40             // Register the customer in the yellow pages
41             if (verbose == 2) {
42                   System.out.println(this.getAID().getName() + " Registering to the
43 yellow pages");
44             }
45             DFAgentDescription dfd = new DFAgentDescription();
46             dfd.setName(getAID());
47             ServiceDescription sd = new ServiceDescription();
48             sd.setType("customer");
49             sd.setName("JADE-shop");
50             dfd.addServices(sd);
51             try {
52                   DFService.register(this, dfd);
53             } catch (FIPAException fe) {
54                   fe.printStackTrace();
55             }
56             addBehaviour(new TickerWaiter(this));
57       }
58
```

```
59          public class TickerWaiter extends CyclicBehaviour {
60
61                  // Wait for a new day
62                  public TickerWaiter(Agent a) {
63                        super(a);
64                  }
65
66                  @Override
67                  public void action() {
68                        MessageTemplate mt =
69    MessageTemplate.or(MessageTemplate.MatchContent("new day"),
70                                    MessageTemplate.MatchContent("terminate"));
71                        ACLMessage msg = myAgent.receive(mt);
72                        if (msg != null) {
73                              if (tickerAgent == null) {
74                                    tickerAgent = msg.getSender();
75                              }
76                              if (msg.getContent().equals("new day")) {
77                                    // spawn new sequential behaviour for day 's activities
78                                    SequentialBehaviour dailyActivity = new
79    SequentialBehaviour();
80                                    // sub - behaviours will execute in the order they are
81    added
82                                    dailyActivity.addSubBehaviour(new MakeOrder(myAgent));
83                                    dailyActivity.addSubBehaviour(new EndDay(myAgent));
84
85                                    myAgent.addBehaviour(dailyActivity);
86                              } else {
87                                    // termination message to end simulation
88                                    if (verbose > 0)
89                                          System.out.println(myAgent.getAID().getName() + "
90    terminating.");
91                                    myAgent.doDelete();
92                              }
93                        } else {
94                              block();
95                        }
96                  }
97
98                  // Create a new order
99                  public class MakeOrder extends OneShotBehaviour {
100
101                       public MakeOrder(Agent a) {
102                             super(a);
103                             myAgent = a;
104                       }
105
106                       @Override
107                       public void action() {
108
109                             motherboard mb = new motherboard();
110                             CPU cpu = new CPU();
111                             RAM ram = new RAM();
112                             HD hd = new HD();
113                             // Randomize PC parts
114                             if (Math.random() < 0.5) {
115                                   cpu.setModel("Mintel");
116                                   mb.setModel("Mintel");
117                             } else {
118                                   cpu.setModel("IMD");
```

```
119                                     mb.setModel("IMD");
120                               }
121
122                               if (Math.random() < 0.5) {
123                                     ram.setSize(4);
124                               } else {
125                                     ram.setSize(16);
126                               }
127
128                               if (Math.random() < 0.5) {
129                                     hd.setSize(1);
130                               } else {
131                                     hd.setSize(2);
132                               }
133                               // Create the PC for the order
134                               PC pc = new PC();
135                               pc.setCPU(cpu);
136                               pc.setHD(hd);
137                               pc.setMB(mb);
138                               pc.setRAM(ram);
139
140                               // Randomize attributes for the order
141                               int quantity = (int) Math.floor(1 + (50 * Math.random()));
142                               int price = quantity * (int) Math.floor(500 + (200 *
143   Math.random()));
144                               int days = (int) Math.floor(1 + (10 * Math.random()));
145
146                               // Create the Customer order
147                               CustOrder co = new CustOrder();
148                               co.setCustomer(myAgent.getAID());
149                               co.setDays(days);
150                               co.setPc(pc);
151                               co.setPrice(price);
152                               co.setQuantity(quantity);
153                               if (verbose == 2) {
154                                     System.out.println(myAgent.getAID().getName() + "
155   created the following order: " + co.toString());
156                               }
157                               ;
158
159                               // Send the order to the shop
160                               ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
161                               msg.addReceiver(shopAgent);
162                               msg.setLanguage(codec.getName());
163                               msg.setOntology(ontology.getName());
164                               Action request = new Action();
165                               request.setAction(co);
166                               request.setActor(shopAgent);
167                               try {
168                                     getContentManager().fillContent(msg, request);
169                                     send(msg);
170                               } catch (CodecException ce) {
171                                     ce.printStackTrace();
172                               } catch (OntologyException oe) {
173                                     oe.printStackTrace();
174                               }
175                         }
176
177                   }
178
```

```
179             public class EndDay extends OneShotBehaviour {
180
181                     public EndDay(Agent a) {
182                             super(a);
183                     }
184
185                     @Override
186                     public void action() {
187                             ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
188                             msg.addReceiver(tickerAgent);
189                             msg.setContent("done");
190                             myAgent.send(msg);
191                     }
192             }
193     }
194 }
```

Class ShopAgent:

```
1  package coursework;
2
3  import java.util.ArrayList;
4  import jade.content.Concept;
5  import jade.content.ContentElement;
6  import jade.content.lang.Codec;
7  import jade.content.lang.Codec.CodecException;
8  import jade.content.lang.sl.SLCodec;
9  import jade.content.onto.Ontology;
10 import jade.content.onto.OntologyException;
11 import jade.content.onto.UngroundedException;
12 import jade.content.onto.basic.Action;
13 import jade.core.AID;
14 import jade.core.Agent;
15 import jade.core.behaviours.CyclicBehaviour;
16 import jade.core.behaviours.OneShotBehaviour;
17 import jade.core.behaviours.SequentialBehaviour;
18 import jade.lang.acl.ACLMessage;
19 import jade.lang.acl.MessageTemplate;
20 import jade.domain.DFService;
21 import jade.domain.FIPAException;
22 import jade.domain.FIPAAgentManagement.DFAgentDescription;
23 import jade.domain.FIPAAgentManagement.ServiceDescription;
24 import coursework.pcshop_ontology.pcshopOntology;
25 import coursework.pcshop_ontology.elements.*;
26
27 public class shopAgent extends Agent {
28       // Change verbose value from 0 to 2 to change the amount of information output
29       // in the console
30       private int verbose = 1;
31       private Codec codec = new SLCodec();
32       private Ontology ontology = pcshopOntology.getInstance();
33       private AID tickerAgent;
34       private ArrayList<CustOrder> orders = new ArrayList<CustOrder>();
35       private Warehouse wh = new Warehouse();
36       private AID supplier1;
37       private AID supplier2;
38       private int totalProfit = 0;
39       private int prevProfit= 0;
40       protected void setup() {
41             getContentManager().registerLanguage(codec);
42             getContentManager().registerOntology(ontology);
43             if (verbose > 0)
44                   System.out.println("shop-agent " + getAID().getName() + " is
45 ready.");
46             // Register the shop service in the yellow pages
47             if (verbose == 2) {
48                   System.out.println(this.getAID().getName() + " Registering to the
49 yellow pages");
50             }
51             DFAgentDescription dfd = new DFAgentDescription();
52             dfd.setName(getAID());
53             ServiceDescription sd = new ServiceDescription();
54             sd.setType("shop");
55             sd.setName("JADE-shop");
56             dfd.addServices(sd);
57             try {
58                   DFService.register(this, dfd);
```

```
59              } catch (FIPAException fe) {
60                    fe.printStackTrace();
61              }
62
63              addBehaviour(new OneShotBehaviour() {
64                    public void action() {
65                          DFAgentDescription template1 = new DFAgentDescription();
66                          ServiceDescription sd1 = new ServiceDescription();
67                          sd1.setType("supplier1");
68                          template1.addServices(sd1);
69                          DFAgentDescription template2 = new DFAgentDescription();
70                          ServiceDescription sd2 = new ServiceDescription();
71                          sd2.setType("supplier2");
72                          template2.addServices(sd2);
73                          try {
74                                DFAgentDescription[] agentsType1 =
75    DFService.search(myAgent, template1);
76                                supplier1 = agentsType1[0].getName();
77
78                                DFAgentDescription[] agentsType2 =
79    DFService.search(myAgent, template2);
80                                supplier2 = agentsType2[0].getName();
81                          } catch (FIPAException fe) {
82                                fe.printStackTrace();
83                          }
84
85                    }
86
87              });
88              addBehaviour(new TickerWaiter(this));
89
90        }
91
92        @Override
93        protected void takeDown() {
94              // Deregister from the yellow pages
95              try {
96                    DFService.deregister(this);
97              } catch (FIPAException e) {
98                    e.printStackTrace();
99              }
100       }
101
102       public class TickerWaiter extends CyclicBehaviour {
103
104             // behaviour to wait for a new day
105             public TickerWaiter(Agent a) {
106                   super(a);
107             }
108
109             @Override
110             public void action() {
111                   MessageTemplate mt =
112   MessageTemplate.or(MessageTemplate.MatchPerformative(ACLMessage.REQUEST),
113                               MessageTemplate.MatchPerformative(ACLMessage.INFORM));
114                   ACLMessage msg = myAgent.receive(mt);
115                   ContentElement ce = null;
116                   if (msg != null) {
117                         switch (msg.getPerformative()) {
118                         case (ACLMessage.REQUEST):
```

```
119
120                                     try {
121                                           ce = getContentManager().extractContent(msg);
122                                     } catch (UngroundedException e) {
123
124                                           e.printStackTrace();
125                                     } catch (CodecException e) {
126
127                                           e.printStackTrace();
128                                     } catch (OntologyException e) {
129
130                                           e.printStackTrace();
131                                     }
132                                     if (ce instanceof Action) {
133                                           Concept action = ((Action) ce).getAction();
134                                           if (action instanceof CustOrder) {
135                                                 CustOrder co = (CustOrder) action;
136
137
138                                                 EvaluateOrder(co);
139
140                                           }
141                                           if (action instanceof SuppOrder) {
142                                                 SuppOrder so = (SuppOrder) action;
143
144
145                                                 AddToWarehouse(so);
146                                                 totalProfit -= so.getPrice();
147                                           }
148                                     }
149
150                                     break;
151
152                               case (ACLMessage.INFORM):
153
154                                     if (msg.getContent().equals("new day")) {
155                                           if (tickerAgent == null) {
156                                                 tickerAgent = msg.getSender();
157                                           }
158                                           prevProfit=totalProfit;
159                                           if (verbose == 2) {
160                                                 System.out.println("Current profit: " +
161   totalProfit);
162                                           }
163
164
165                                           SequentialBehaviour dailyActivity = new
166   SequentialBehaviour();
167
168
169                                           dailyActivity.addSubBehaviour(new
170   Daily(myAgent));
171                                           dailyActivity.addSubBehaviour(new
172   EndDay(myAgent));
173
174                                           myAgent.addBehaviour(dailyActivity);
175                                     } else {
176                                           // termination message to end simulation
177
178                                           if (verbose > 0)
```

```
179                                         System.out.println(myAgent.getAID().getName() + "
180   terminating.");
181                                         System.out.println("Simulation ended. Total
182   Profit: " + totalProfit);
183                                         myAgent.doDelete();
184                                   }
185
186                                   break;
187
188                             }
189
190                       } else
191
192                       {
193                             block();
194                       }
195                 }
196
197                 // Daily function routine
198                 public class Daily extends OneShotBehaviour {
199
200                       public Daily(Agent a) {
201                             super(a);
202                       }
203
204                       @Override
205                       public void action() {
206                             OrderDayCounter();
207                             totalProfit -= wh.Penality();
208                             Manufacture();
209                       }
210                 }
211
212                 public class EndDay extends OneShotBehaviour {
213
214                       public EndDay(Agent a) {
215                             super(a);
216                       }
217
218                       @Override
219                       public void action() {
220                             if(verbose >0)
221                             System.out.println("Daily profit: " + (totalProfit-
222   prevProfit));
223                             ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
224                             msg.addReceiver(tickerAgent);
225                             msg.setContent("done");
226                             myAgent.send(msg);
227                       }
228                 }
229           }
230
231           // Evaluates if an order is profitable or not (Earning at least 40% of the gross
232           // profit)
233           private boolean EvaluateOrder(CustOrder co) {
234
235                 int profit = 0;
236                 int supp = 0;
237                 if (co.getDays() < 4) {
```

```
238                     profit = co.getPrice() - CalculateOrderPrice(co.getPc(),
239   co.getQuantity(), 2);
240                     supp = 2;
241               } else {
242                     profit = co.getPrice() - CalculateOrderPrice(co.getPc(),
243   co.getQuantity(), 1);
244                     supp = 1;
245               }
246
247               if (((profit * 100) / co.getPrice()) > 40)
248
249               {
250                     orders.add(co);
251                     OrderComponents(co, supp);
252                     return true;
253               }
254
255               return false;
256         }
257
258         // Orders components from the supplier
259         private void OrderComponents(CustOrder co, int supplier) {
260               ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
261               Action request = new Action();
262               co.setCustomer(this.getAID());
263               if (verbose == 2) {
264                     System.out.println("Sending the follow order to supplier" + supplier
265   + ": " + co.toString());
266               }
267               switch (supplier) {
268               case 1:
269                     msg.addReceiver(supplier1);
270                     msg.setLanguage(codec.getName());
271                     msg.setOntology(ontology.getName());
272                     request.setAction(co);
273                     request.setActor(supplier1);
274                     try {
275                           getContentManager().fillContent(msg, request);
276                           send(msg);
277                     } catch (CodecException ce) {
278                           ce.printStackTrace();
279                     } catch (OntologyException oe) {
280                           oe.printStackTrace();
281                     }
282                     break;
283               case 2:
284                     msg.addReceiver(supplier2);
285                     msg.setLanguage(codec.getName());
286                     msg.setOntology(ontology.getName());
287                     request.setAction(co);
288                     request.setActor(supplier2);
289                     try {
290                           getContentManager().fillContent(msg, request);
291                           send(msg);
292                     } catch (CodecException ce) {
293                           ce.printStackTrace();
294                     } catch (OntologyException oe) {
295                           oe.printStackTrace();
296                     }
297
```

```
298                     break;
299               }
300         }
301
302         // Local price table from both of the suppliers
303         private int CalculateOrderPrice(PC pc, int quantity, int supplier) {
304               if (supplier != 1 && supplier != 2) {
305                     return 0;
306               }
307               int price = 0;
308
309               switch (supplier) {
310               case 1:
311                     if (pc.getCPU().getModel() == "Mintel") {
312                           price += 200;
313                     } else {
314                           price += 150;
315                     }
316                     if (pc.getMB().getModel() == "Mintel") {
317                           price += 125;
318                     } else {
319                           price += 75;
320                     }
321                     if (pc.getRAM().getSize() == 16) {
322                           price += 90;
323                     } else {
324                           price += 50;
325                     }
326                     if (pc.getHD().getSize() == 2) {
327                           price += 75;
328                     } else {
329                           price += 50;
330                     }
331                     break;
332               case 2:
333                     if (pc.getCPU().getModel() == "Mintel") {
334                           price += 175;
335                     } else {
336                           price += 130;
337                     }
338                     if (pc.getMB().getModel() == "Mintel") {
339                           price += 115;
340                     } else {
341                           price += 65;
342                     }
343                     if (pc.getRAM().getSize() == 16) {
344                           price += 80;
345                     } else {
346                           price += 40;
347                     }
348                     if (pc.getHD().getSize() == 2) {
349                           price += 65;
350                     } else {
351                           price += 45;
352                     }
353                     break;
354
355               }
356
357               return price * quantity;
```

```
358
359         }
360
361         // Add items to the warehouse storage
362         private void AddToWarehouse(SuppOrder so) {
363               int quantity = so.getQuantity();
364               wh.addCpu(so.getCpu(), quantity);
365               wh.addHd(so.getHd(), quantity);
366               wh.addMb(so.getMb(), quantity);
367               wh.addRam(so.getRam(), quantity);
368         }
369
370         // Remove items from the warehouse storage
371         private void RemoveFromWarehouse(CustOrder co) {
372               int quantity = 0 - co.getQuantity();
373               wh.addCpu(co.getPc().getCPU(), quantity);
374               wh.addHd(co.getPc().getHD(), quantity);
375               wh.addMb(co.getPc().getMB(), quantity);
376               wh.addRam(co.getPc().getRAM(), quantity);
377         }
378
379         // Updates the day count on orders
380         private void OrderDayCounter() {
381               for (CustOrder co : orders) {
382                     co.setDays(co.getDays() - 1);
383               }
384               for (CustOrder co : new ArrayList<CustOrder>(orders)) {
385                     if (co.getDays() < 0) {
386                           totalProfit -= 50;
387                     }
388               }
389         }
390
391         // Find if an order is possible, then fulfils it
392         private void Manufacture() {
393               int maxPC = 50;
394               //Loop all orders
395               for (CustOrder c : new ArrayList<CustOrder>(orders)) {
396                     //If all the parts are available
397                     if (wh.canManufacture(c)) {
398                           //If the number of pcs don't exceed the maximum number
399   allowed
400                           if (c.getQuantity() <= maxPC)
401                           {
402                                 // Remove parts to be used, add profit, remove order
403   from list
404                                 RemoveFromWarehouse(c);
405                                 totalProfit += c.getPrice();
406                                 maxPC -= c.getQuantity();
407                                 orders.remove(c);
408                           }
409                     }
410               }
411
412         }
413
414   }
415
```

Class Supplier:

```
package coursework;

import java.util.ArrayList;

import jade.content.Concept;
import jade.content.ContentElement;
import jade.content.lang.Codec;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.Ontology;
import jade.content.onto.OntologyException;
import jade.content.onto.UngroundedException;
import jade.content.onto.basic.Action;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.SequentialBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import coursework.pcshop_ontology.pcshopOntology;
import coursework.pcshop_ontology.elements.*;

public class Supplier extends Agent {
	// Change verbose value from 0 to 2 to change the amount of information output
	// in the console
	private int verbose = 0;
	private Codec codec = new SLCodec();
	private Ontology ontology = pcshopOntology.getInstance();
	private AID tickerAgent;
	private ArrayList<CustOrder> coList = new ArrayList<>();

	protected void setup() {
		getContentManager().registerLanguage(codec);
		getContentManager().registerOntology(ontology);
		if (verbose > 0)
			System.out.println("supplier-agent " + getAID().getName() + " is
ready.");
		// Register the supplier service in the yellow pages
		if (verbose == 2) {
			System.out.println(this.getAID().getName() + " Registering to the
yellow pages");
		}
		DFAgentDescription dfd = new DFAgentDescription();
		dfd.setName(getAID());
		ServiceDescription sd = new ServiceDescription();
		sd.setType("supplier1");
		sd.setName("JADE-shop");
		dfd.addServices(sd);
		try {
			DFService.register(this, dfd);
		} catch (FIPAException fe) {
```

```
57                      fe.printStackTrace();
58                }
59                addBehaviour(new TickerWaiter(this));
60
61          }
62
63          public class TickerWaiter extends CyclicBehaviour {
64
65                // Wait for a new day
66                public TickerWaiter(Agent a) {
67                      super(a);
68                }
69
70                @Override
71                public void action() {
72                      MessageTemplate mt =
73    MessageTemplate.or(MessageTemplate.MatchPerformative(ACLMessage.REQUEST),
74                                  MessageTemplate.MatchPerformative(ACLMessage.INFORM));
75                      ACLMessage msg = myAgent.receive(mt);
76
77                      if (msg != null) {
78                            switch (msg.getPerformative()) {
79                            case (ACLMessage.REQUEST):
80                                  ContentElement ce = null;
81                                  try {
82                                        ce = getContentManager().extractContent(msg);
83                                  } catch (UngroundedException e) {
84                                        e.printStackTrace();
85                                  } catch (CodecException e) {
86                                        e.printStackTrace();
87                                  } catch (OntologyException e) {
88                                        e.printStackTrace();
89                                  }
90                                  if (ce instanceof Action) {
91                                        Concept action = ((Action) ce).getAction();
92                                        // Check if the message is an instance of
93    Customer Order
94                                        if (action instanceof CustOrder) {
95                                              CustOrder co = (CustOrder) action;
96                                              // Set delivery time to 1 day and add
97    order to the list
98                                              co.setDays(1);
99                                              coList.add(co);
100                                       }
101                                 }
102
103                                 break;
104
105                           case (ACLMessage.INFORM):
106
107                                 if (msg.getContent().equals("new day")) {
108                                       if (tickerAgent == null) {
109                                             tickerAgent = msg.getSender();
110                                       }
111                                       OrderDayCounter();
112                                       SequentialBehaviour dailyActivity = new
113   SequentialBehaviour();
114                                       dailyActivity.addSubBehaviour(new
115   SendParts(myAgent));
```

```
116                                         dailyActivity.addSubBehaviour(new
117   EndDay(myAgent));
118                                         myAgent.addBehaviour(dailyActivity);
119                                   } else {
120                                         // termination message to end simulation
121                                         if (verbose > 0)
122
123         System.out.println(myAgent.getAID().getName() + " terminating.");
124                                         myAgent.doDelete();
125                                   }
126
127                                   break;
128
129                             }
130
131                       } else
132
133                       {
134                             block();
135                       }
136                 }
137
138                 // Send parts to the manufacturer
139                 public class SendParts extends OneShotBehaviour {
140
141                       public SendParts(Agent a) {
142                             super(a);
143                       }
144
145                       ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
146                       Action request = new Action();
147
148                       @Override
149                       public void action() {
150                             for (CustOrder c : coList) {
151                                   if (c.getDays() == 0) {
152                                         // Create a new supplier order and fills it with
153   the pieces, setting up the
154                                         // quantity and price
155                                         SuppOrder so = new SuppOrder();
156                                         so.setCpu(c.getPc().getCPU());
157                                         so.setHd(c.getPc().getHD());
158                                         so.setMb(c.getPc().getMB());
159                                         so.setRam(c.getPc().getRAM());
160                                         so.setQuantity(c.getQuantity());
161                                         so.setPrice(CalcPrice(c.getPc(),
162   c.getQuantity()));
163                                         // Send a message to the manufacturer with the
164   SuppOrder as content
165                                         msg.addReceiver(c.getCustomer());
166                                         msg.setLanguage(codec.getName());
167                                         msg.setOntology(ontology.getName());
168                                         request.setAction(so);
169                                         request.setActor(c.getCustomer());
170                                         try {
171                                               getContentManager().fillContent(msg,
172   request);
173                                               send(msg);
174                                         } catch (CodecException ce) {
175                                               ce.printStackTrace();
```

```
176                                     } catch (OntologyException oe) {
177                                           oe.printStackTrace();
178                                     }
179                               }
180                         }
181
182                   }
183
184             }
185
186             public class EndDay extends OneShotBehaviour {
187
188                   public EndDay(Agent a) {
189                         super(a);
190                   }
191
192                   @Override
193                   public void action() {
194                         ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
195                         msg.addReceiver(tickerAgent);
196                         msg.setContent("done");
197                         myAgent.send(msg);
198                   }
199             }
200       }
201
202       // Updates the days in the order list
203       private void OrderDayCounter() {
204             for (CustOrder co : coList) {
205                   co.setDays(co.getDays() - 1);
206             }
207             for (CustOrder co : new ArrayList<CustOrder>(coList)) {
208                   if (co.getDays() < 0) {
209                         coList.remove(co);
210                   }
211             }
212
213       }
214
215       // Calculate the price of an order
216       private int CalcPrice(PC pc, int quantity) {
217             int price = 0;
218             if (pc.getCPU().getModel() == "Mintel") {
219                   price += 200;
220             } else {
221                   price += 150;
222             }
223             if (pc.getMB().getModel() == "Mintel") {
224                   price += 125;
225             } else {
226                   price += 75;
227             }
228             if (pc.getRAM().getSize() == 16) {
229                   price += 90;
230             } else {
231                   price += 50;
232             }
233             if (pc.getHD().getSize() == 2) {
234                   price += 75;
235             } else {
```

```
236                     price += 50;
237               }
238               return price * quantity;
239         }
240
241
```

Class Supplier2:

```
1   package coursework;
2
3   import java.util.ArrayList;
4   import jade.content.Concept;
5   import jade.content.ContentElement;
6   import jade.content.lang.Codec;
7   import jade.content.lang.Codec.CodecException;
8   import jade.content.lang.sl.SLCodec;
9   import jade.content.onto.Ontology;
10  import jade.content.onto.OntologyException;
11  import jade.content.onto.UngroundedException;
12  import jade.content.onto.basic.Action;
13  import jade.core.AID;
14  import jade.core.Agent;
15  import jade.core.behaviours.CyclicBehaviour;
16  import jade.core.behaviours.OneShotBehaviour;
17  import jade.core.behaviours.SequentialBehaviour;
18  import jade.lang.acl.ACLMessage;
19  import jade.lang.acl.MessageTemplate;
20  import jade.domain.DFService;
21  import jade.domain.FIPAException;
22  import jade.domain.FIPAAgentManagement.DFAgentDescription;
23  import jade.domain.FIPAAgentManagement.ServiceDescription;
24  import coursework.pcshop_ontology.pcshopOntology;
25  import coursework.pcshop_ontology.elements.*;
26
27  public class Supplier2 extends Agent {
28        // Change verbose value from 0 to 2 to change the amount of information output
29        // in the console
30        private int verbose = 0;
31        private Codec codec = new SLCodec();
32        private Ontology ontology = pcshopOntology.getInstance();
33        private AID tickerAgent;
34        private ArrayList<CustOrder> coList = new ArrayList<>();
35
36        protected void setup() {
37              getContentManager().registerLanguage(codec);
38              getContentManager().registerOntology(ontology);
39              if (verbose > 0)
40                    System.out.println("supplier-agent " + getAID().getName() + " is
41  ready.");
42              // Register the supplier service in the yellow pages
43              DFAgentDescription dfd = new DFAgentDescription();
44              if (verbose == 2) {
45                    System.out.println(this.getAID().getName() + " Registering to the
46  yellow pages");
47              }
48              dfd.setName(getAID());
49              ServiceDescription sd = new ServiceDescription();
50              sd.setType("supplier2");
51              sd.setName("JADE-shop");
52              dfd.addServices(sd);
```

```
53              try {
54                    DFService.register(this, dfd);
55              } catch (FIPAException fe) {
56                    fe.printStackTrace();
57              }
58              addBehaviour(new TickerWaiter(this));
59
60        }
61
62        public class TickerWaiter extends CyclicBehaviour {
63
64              // Wait for a new day
65              public TickerWaiter(Agent a) {
66                    super(a);
67              }
68
69              @Override
70              public void action() {
71                    MessageTemplate mt =
72  MessageTemplate.or(MessageTemplate.MatchPerformative(ACLMessage.REQUEST),
73                                MessageTemplate.MatchPerformative(ACLMessage.INFORM));
74                    ACLMessage msg = myAgent.receive(mt);
75
76                    if (msg != null) {
77                          switch (msg.getPerformative()) {
78                          case (ACLMessage.REQUEST):
79                                ContentElement ce = null;
80                                try {
81                                      ce = getContentManager().extractContent(msg);
82                                } catch (UngroundedException e) {
83                                      e.printStackTrace();
84                                } catch (CodecException e) {
85                                      e.printStackTrace();
86                                } catch (OntologyException e) {
87                                      e.printStackTrace();
88                                }
89                                if (ce instanceof Action) {
90                                      Concept action = ((Action) ce).getAction();
91                                      // Check if the message is an instance of
92  Customer Order
93                                      if (action instanceof CustOrder) {
94                                            CustOrder co = (CustOrder) action;
95                                            // Set delivery time to 4 days and add
96  order to the list
97
98                                            co.setDays(4);
99                                            coList.add(co);
100                                     }
101                               }
102
103                               break;
104
105                         case (ACLMessage.INFORM):
106
107                               if (msg.getContent().equals("new day")) {
108                                     if (tickerAgent == null) {
109                                           tickerAgent = msg.getSender();
110                                     }
111                                     OrderDayCounter();
```

```
112                                         SequentialBehaviour dailyActivity = new
113   SequentialBehaviour();
114                                         dailyActivity.addSubBehaviour(new
115   SendParts(myAgent));
116                                         dailyActivity.addSubBehaviour(new
117   EndDay(myAgent));
118                                         myAgent.addBehaviour(dailyActivity);
119                                   } else {
120                                         if (verbose > 0)
121
122         System.out.println(myAgent.getAID().getName() + " terminating.");
123                                         // termination message to end simulation
124                                         myAgent.doDelete();
125                                   }
126
127                                   break;
128
129                             }
130
131                       } else
132
133                       {
134                             block();
135                       }
136                 }
137
138                 // Send parts to the manufacturer
139                 public class SendParts extends OneShotBehaviour {
140
141                       public SendParts(Agent a) {
142                             super(a);
143                       }
144
145                       ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
146                       Action request = new Action();
147
148                       @Override
149                       public void action() {
150                             for (CustOrder c : coList) {
151                                   if (c.getDays() == 0) {
152                                         // Create a new supplier order and fills it with
153   the pieces, setting up the
154                                         // quantity and price
155                                         SuppOrder so = new SuppOrder();
156                                         so.setCpu(c.getPc().getCPU());
157                                         so.setHd(c.getPc().getHD());
158                                         so.setMb(c.getPc().getMB());
159                                         so.setRam(c.getPc().getRAM());
160                                         so.setQuantity(c.getQuantity());
161                                         so.setPrice(CalcPrice(c.getPc(),
162   c.getQuantity()));
163                                         // Send a message to the manufacturer with the
164   SuppOrder as content
165                                         msg.addReceiver(c.getCustomer());
166                                         msg.setLanguage(codec.getName());
167                                         msg.setOntology(ontology.getName());
168                                         request.setAction(so);
169                                         request.setActor(c.getCustomer());
170                                         try {
```

```
171                                                 getContentManager().fillContent(msg,
172   request);
173                                                 send(msg);
174                                           } catch (CodecException ce) {
175                                                 ce.printStackTrace();
176                                           } catch (OntologyException oe) {
177                                                 oe.printStackTrace();
178                                           }
179                                     }
180                               }
181
182                         }
183
184                   }
185
186                   public class EndDay extends OneShotBehaviour {
187
188                         public EndDay(Agent a) {
189                               super(a);
190                         }
191
192                         @Override
193                         public void action() {
194                               ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
195                               msg.addReceiver(tickerAgent);
196                               msg.setContent("done");
197                               myAgent.send(msg);
198                         }
199                   }
200             }
201
202             // Updates the days in the order list
203             private void OrderDayCounter() {
204                   for (CustOrder co : coList) {
205                         co.setDays(co.getDays() - 1);
206                   }
207                   for (CustOrder co : new ArrayList<CustOrder>(coList)) {
208                         if (co.getDays() < 0) {
209                               coList.remove(co);
210                         }
211                   }
212
213             }
214
215             // Calculate the price of an order
216             private int CalcPrice(PC pc, int quantity) {
217                   int price = 0;
218                   if (pc.getCPU().getModel() == "Mintel") {
219                         price += 175;
220                   } else {
221                         price += 130;
222                   }
223                   if (pc.getMB().getModel() == "Mintel") {
224                         price += 115;
225                   } else {
226                         price += 65;
227                   }
228                   if (pc.getRAM().getSize() == 16) {
229                         price += 80;
230                   } else {
```

```
231                     price += 40;
232                 }
233                 if (pc.getHD().getSize() == 2) {
234                     price += 65;
235                 } else {
236                     price += 45;
237                 }
238                 return price * quantity;
239         }
240     }
241
```

Class Warehouse:

```
1   package coursework;
2
3   import coursework.pcshop_ontology.elements.*;
4
5   public class Warehouse {
6
7         int CPUMintel = 0;
8         int CPUIMD = 0;
9         int MBMintel = 0;
10        int MBIMD = 0;
11        int RAM4 = 0;
12        int RAM16 = 0;
13        int HD1 = 0;
14        int HD2 = 0;
15
16        // Calculate the daily cost of storage
17        public int Penality() {
18              return (CPUMintel + CPUIMD + MBMintel + MBIMD + RAM4 + RAM16 + HD1 + HD2);
19
20        }
21
22        // Checks if an order is possible with the stored parts comparing the order
23  quantity with the stored quantity of each piece, switching between models and sizes.
24        public boolean canManufacture(CustOrder co) {
25
26              boolean cpu = false;
27              boolean mb = false;
28              boolean hd = false;
29              boolean ram = false;
30
31              switch (co.getPc().getCPU().getModel()) {
32              case "Mintel":
33                    if (co.getQuantity() <= CPUMintel) {
34                          cpu = true;
35                    }
36                    break;
37              case "IMD":
38                    if (co.getQuantity() <= CPUIMD) {
39                          cpu = true;
40                    }
41                    break;
42              }
43
44              switch (co.getPc().getMB().getModel()) {
45              case "Mintel":
46                    if (co.getQuantity() <= MBMintel) {
47                          mb = true;
48                    }
49                    break;
50              case "IMD":
51                    if (co.getQuantity() <= MBIMD) {
52                          mb = true;
53                    }
54                    break;
55              }
56
57              switch (co.getPc().getHD().getSize()) {
58              case 1:
```

```
59                      if (co.getQuantity() <= HD1) {
60                              hd = true;
61                      }
62                      break;
63              case 2:
64                      if (co.getQuantity() <= HD2) {
65                              hd = true;
66                      }
67                      break;
68              }
69
70              switch (co.getPc().getRAM().getSize()) {
71              case 4:
72                      if (co.getQuantity() <= RAM4) {
73                              ram = true;
74                      }
75                      break;
76              case 16:
77                      if (co.getQuantity() <= RAM16) {
78                              ram = true;
79                      }
80                      break;
81              }
82
83              if (cpu && mb && hd && ram) {
84                      return true;
85              }
86              return false;
87      }
88
89      // Getters and setters
90
91      public int getCPUMintel() {
92              return CPUMintel;
93      }
94
95      public void setCPUMintel(int cPUMintel) {
96              CPUMintel = cPUMintel;
97      }
98
99      public int getCPUIMD() {
100             return CPUIMD;
101     }
102
103     public void setCPUIMD(int cPUIMD) {
104             CPUIMD = cPUIMD;
105     }
106
107     public int getMBMintel() {
108             return MBMintel;
109     }
110
111     public void setMBMintel(int mBMintel) {
112             MBMintel = mBMintel;
113     }
114
115     public int getMBIMD() {
116             return MBIMD;
117     }
118
```

```
119         public void setMBIMD(int mBIMD) {
120               MBIMD = mBIMD;
121         }
122
123         public int getRAM4() {
124               return RAM4;
125         }
126
127         public void setRAM4(int rAM4) {
128               RAM4 = rAM4;
129         }
130
131         public int getRAM16() {
132               return RAM16;
133         }
134
135         public void setRAM16(int rAM16) {
136               RAM16 = rAM16;
137         }
138
139         public int getHD1() {
140               return HD1;
141         }
142
143         public void setHD1(int hD1) {
144               HD1 = hD1;
145         }
146
147         public int getHD2() {
148               return HD2;
149         }
150
151         public void setHD2(int hD2) {
152               HD2 = hD2;
153         }
154
155         public void addRam(RAM ram, int quantity) {
156               switch (ram.getSize()) {
157               case 4:
158                     RAM4 += quantity;
159                     break;
160               case 16:
161                     RAM16 += quantity;
162                     break;
163               }
164         }
165
166         public void addCpu(CPU cpu, int quantity) {
167               switch (cpu.getModel()) {
168               case "Mintel":
169                     CPUMintel += quantity;
170                     break;
171               case "IMD":
172                     CPUIMD += quantity;
173                     break;
174               }
175         }
176
177         public void addHd(HD hd, int quantity) {
178               switch (hd.getSize()) {
```

```
179                 case 1:
180                       HD1 += quantity;
181                       break;
182                 case 2:
183                       HD2 += quantity;
184                       break;
185                 }
186
187           }
188
189           public void addMb(motherboard mb, int quantity) {
190                 switch (mb.getModel()) {
191                 case "Mintel":
192                       MBMintel += quantity;
193                       break;
194                 case "IMD":
195                       MBIMD += quantity;
196                       break;
197                 }
198           }
199
200           @Override
201           public String toString() {
202                 return "Warehouse [CPUMintel=" + CPUMintel + ", CPUIMD=" + CPUIMD + ",
203     MBMintel=" + MBMintel + ", MBIMD="
204                             + MBIMD + ", RAM4=" + RAM4 + ", RAM16=" + RAM16 + ", HD1=" +
205     HD1 + ", HD2=" + HD2 + "]";
206           }
207
208     }
```

Class Component:

```
1  package coursework.pcshop_ontology.elements;
2
3  import jade.content.Concept;
4  import jade.content.onto.annotations.Slot;
5
6  public class component implements Concept {
7
8         private int ID;
9
10        @Slot(mandatory = false)
11        public int getID() {
12              return ID;
13        }
14
15        public void setID(int ID) {
16              this.ID = ID;
17        }
18 }
```

Class CPU:

```
1  package coursework.pcshop_ontology.elements;
2
3  import jade.content.onto.annotations.Slot;
4
5  public class CPU extends component {
6
7         private String cpu_model;
8
9         @Slot(mandatory = true)
10        public String getModel() {
11              return cpu_model;
12        }
13
14        public void setModel(String model) {
15              this.cpu_model = model;
16        }
17
18 }
```

Class HD:

```
1  package coursework.pcshop_ontology.elements;
2
3  import jade.content.onto.annotations.Slot;
4
5  public class HD extends component {
6         private int HD_size;
7
8         @Slot(mandatory = true)
9         public int getSize() {
10              return HD_size;
11        }
12
13        public void setSize(int size) {
14              this.HD_size = size;}}
```

Class RAM:

```
15  package coursework.pcshop_ontology.elements;
16
17  import jade.content.onto.annotations.Slot;
18
19  public class RAM extends component {
20        private int RAM_size;
21
22        @Slot(mandatory = true)
23        public int getSize() {
24              return RAM_size;
25        }
26
27        public void setSize(int size) {
28              this.RAM_size = size;
29        }
30  }
```

Class Motherboard:

```
 1  package coursework.pcshop_ontology.elements;
 2
 3  import jade.content.onto.annotations.Slot;
 4
 5  public class motherboard extends component {
 6        private String mb_model;
 7
 8        @Slot(mandatory = true)
 9        public String getModel() {
10              return mb_model;
11        }
12
13        public void setModel(String model) {
14              this.mb_model = model;
15        }
16
17  }
```

Class PC:

```
 1  package coursework.pcshop_ontology.elements;
 2
 3  import jade.content.Concept;
 4  import jade.content.onto.annotations.Slot;
 5
 6  public class PC implements Concept {
 7
 8        private int ID;
 9        private motherboard MB;
10        private HD HD;
11        private CPU CPU;
12        private RAM RAM;
13
14        @Slot(mandatory = true)
15        public int getID() {
16              return ID;
17        }
18
19        public void setID(int ID) {
```

```
20              this.ID = ID;
21        }
22
23        @Slot(mandatory = true)
24        public HD getHD() {
25              return HD;
26        }
27
28        public void setHD(HD HD) {
29              this.HD = HD;
30        }
31
32        @Slot(mandatory = true)
33        public CPU getCPU() {
34              return CPU;
35        }
36
37        public void setCPU(CPU CPU) {
38              this.CPU = CPU;
39        }
40
41        @Slot(mandatory = true)
42        public RAM getRAM() {
43              return RAM;
44        }
45
46        public void setRAM(RAM RAM) {
47              this.RAM = RAM;
48        }
49
50        @Slot(mandatory = true)
51        public motherboard getMB() {
52              return MB;
53        }
54
55        public void setMB(motherboard MB) {
56              this.MB = MB;
57        }
58  }
```

Class CustOrder:

```
1  package coursework.pcshop_ontology.elements;
2
3  import jade.content.AgentAction;
4
5  import jade.core.AID;
6
7  public class CustOrder implements AgentAction  {
8        private AID customer;
9        private PC pc;
10       private int price;
11       private int quantity;
12       private int days;
13
14       public AID getCustomer() {
15             return customer;
16       }
17
18       public void setCustomer(AID customer) {
19             this.customer = customer;
20       }
21
22       public int getPrice() {
23             return price;
24       }
25
26       public void setPrice(int price) {
27             this.price = price;
28       }
29
30       public PC getPc() {
31             return pc;
32       }
33
34       public void setPc(PC pc) {
35             this.pc = pc;
36       }
37
38       public int getQuantity() {
39             return quantity;
40       }
41
42       public void setQuantity(int quantity) {
43             this.quantity = quantity;
44       }
45
46       public int getDays() {
47             return days;
48       }
49
50       public void setDays(int days) {
51             this.days = days;
52       }
53
54       @Override
55       public String toString() {
56             return "CustOrder [customer=" + customer + ", pc=" + pc + ", price=" +
57 price + ", quantity=" + quantity
58                          + ", days=" + days + "]";
```

```
59          }}
```

Class SuppOrder:

```
 1
 2   package coursework.pcshop_ontology.elements;
 3
 4   import jade.content.AgentAction;
 5
 6
 7   public class SuppOrder implements AgentAction  {
 8
 9          motherboard mb;
10          CPU cpu;
11          RAM ram;
12          HD hd;
13          int quantity;
14          int price;
15
16
17
18          public motherboard getMb() {
19                  return mb;
20          }
21
22          public void setMb(motherboard mb) {
23                  this.mb = mb;
24          }
25
26          public CPU getCpu() {
27                  return cpu;
28          }
29
30          public void setCpu(CPU cpu) {
31                  this.cpu = cpu;
32          }
33
34          public RAM getRam() {
35                  return ram;
36          }
37
38          public void setRam(RAM ram) {
39                  this.ram = ram;
40          }
41
42          public HD getHd() {
43                  return hd;
44          }
45
46          public void setHd(HD hd) {
47                  this.hd = hd;
48          }
49
50          public int getQuantity() {
51                  return quantity;
52          }
53
54          public void setQuantity(int quantity) {
55                  this.quantity = quantity;
56          }
57
```

```
58          public int getPrice() {
59                return price;
60          }
61
62          public void setPrice(int price) {
63                this.price = price;
64          }
65    }
```