Christopher Johnson 40275286

SET10111 Multi-Agent Systems Coursework Report

Table of Contents

1 Introduction

The problem specified for this project describes a supply chain involving a smart phone manufacturer. Every day the manufacturer receives orders from customers and must decide which orders to accept and which orders to reject. It is then down to the manufacturer to source parts from suppliers for any accepted orders and to assemble the phones before a set delivery due date to avoid paying late delivery fees. All of these interactions and decisions need to be made with a goal of maximising profit for the manufacturer.

This is an important problem to address as most manufacturing companies are part of a supply chain. These supply chains are made up of many elements and involve lots of different interactions. The more efficient these processes the more successful the company and the more satisfied the customer.

This supply chain system is to be designed and implemented in JADE, a Java agent development framework, as a multi-agent system (MAS). In a MAS, different elements of the supply chain are represented as agents making it a suitable solution for this problem as intelligent agents can act autonomously without human intervention on behalf of the manufacturer to manage orders, the procuring of goods and order assembly.

2 Model Design

2.1 Agents

There are four different types of agents implemented into the system, Customer, Manufacturer, Supplier and Ticker.

2.1.1 Customer Agents

Customer agents are responsible for generating orders. The orders are randomly generated and include:

- Specification of phone.
- Quantity of phones.
- Price per phone.
- Days until order is due.
- Penalty per day for late delivery.

2.1.2 Manufacturer Agent

The manufacturer agent receives customer orders and decides whether to accept or reject them. It also interacts with suppliers to request and receive component deliveries and is responsible for the assembly and shipment of completed customer orders.

2.1.3 Supplier Agents

There are two supplier agents that are a part of the implemented system. The first supplier stocks all available smart phone components and ships deliveries next day and the second supplier ships a selection of components within four days but at a cheaper cost. The suppliers receive orders from the manufacturer and ships deliveries containing the requested components back to the manufacturer on the day they're due.

2.1.4 Ticker Agent

In order to synchronise all of the agent's activities a ticker agent is used. This agent tracks the days passed, synchronising agent behaviour by communicating with the other agents. The ticker sends a "New Day" message to all agents in the system at the beginning of the day and then waits to receive confirmation that all agents have finished their daily behaviours before starting a new day.

2.2 Ontology

Ontology is used to establish a relationship between concepts, data and entities within a domain. In the case of this project the domain is the supply chain modelled by the system. In a supply chain scenario it should include knowledge about the good that the system is dealing with and the interaction rules (Chen et al., 2012).

The system's ontology is incomplete, only containing predicates (Figures 3 and 4), however they are important as they ensure that agents receive the correct information.

2.3 Communication Protocols

All agent communication uses the FIPA Agent Communication Language (ACL). This ensures interoperability by providing a standard message structure (FIPA ACL Message Structure Specification., 2002). A sequence diagram showing all agent communication is given in Figure 5.

The message performative REQUEST is used when a customer makes an order to the manufacturer or when the manufacturer makes an order to a supplier. This means the receiving agent knows it is expected to perform an action upon receiving the message.

The INFORM performative is used when an agent is sharing information with another agent. An example of the INFORM performative being used is for end of day notifications to the ticker agent.

3 Model Implementation

3.1 Agent Behaviours

Tasks that agents are to carry out are contained within behaviours. The behaviours executed by each agent in the system are described below:

3.1.1 Customer Agent Behaviours

The customer agent has a single, simple task to perform each day. For this reason all of its functions are contained within a single CyclicBehaviour called DailyBehaviour. Within this behaviour the customer generates a random order, sends it to the manufacturer agent and then lets the ticker agent know that it is done for the day.

3.1.2 Manufacturer Agent Behaviours

The manufacturer agent has a number of tasks to carry out each day; these are executed as OneShotBehaviours one at a time within a SequentialBehaviour. These behaviours are executed in the order as listed below:

- ReceiveCustomerMessages – At the beginning of every day the manufacturer receives an order from each customer agent. By default there are three customers but this can be modified to any number (see experimental results in section 5).
- ReadyToReceive – This behaviour is to let the supplier agents know that the manufacturer is ready to receive any deliveries from the supplier for that day.
- ReceiveDelivery – The manufacturer receives any deliveries from suppliers for that day and adjusts the warehouse stock accordingly.
- AssembleAndShip – Here every accepted order is checked against warehouse stock. If there’s enough stock in the warehouse to assemble an order then assembly and immediate shipping takes place.
- OrderSorting – This is where the manufacturer’s strategy for choosing orders to accept is implemented. Orders that are accepted are added to a list of accepted orders and rejected orders are ignored.
- MakeSupplierOrders – The manufacturer checks what stock it needs for the list of accepted deliveries and makes an order request to one of the suppliers.

- End Day – The manufacturer sends end of day messages to the supplier agents and the ticker agents. The message is also sent to supplier agents so that they know not to expect any more orders from the manufacturer for the day.

3.1.3 Supplier Agent Behaviours

The supplier agent's both have DailyBehaviours that are executed each day; within this behaviour the following behaviours are implemented:

- OrdersInServer – This cyclic behaviour waits to receive a message containing a ManufacturerSupplierOrder. Upon receiving an order the supplier calculates the cost and creates a SupplierManufacturerDelivery which is stored in an ArrayList until the day it's due to be shipped.
- ShipOrder – This is a cyclic behaviour which waits to receive a "Ready for delivery" message from the manufacturer. It then sorts through the list of deliveries and sends out any deliveries that are due to arrive to the manufacturer on the current day.
- EndOfDayListener – This is another cyclic behaviour that waits to receive end of day messages from the manufacturer before sending the ticker agent its own end of day message, now that it knows it won't be needed for the rest of the day.

3.1.4 Ticker Agent Behaviour

The ticker agent has one simple behaviour called SyncAgentsBehaviour. At the beginning of every day the ticker sends all agents in the system a "New Day" message letting them know to start their daily behaviours. The ticker agent then waits to receive "Done" messages from each of the agents in the system before ending the day and sending new "New Day" messages to the system again.

3.2 System Constraints

Small and phablet smartphone specifications – When a customer creates an order it specifies a phone type. This is then used by the Manufacturer to determine the screen and battery sizes it will need to use for assembling the phone. See listings 1 and 2 for the implementation of this constraint. This is enforced in the ontology as when a customer creates a CustomerOrder predicate they must specify a type of phone; shown in Figure 3.

Component delivery times from the two suppliers – When a manufacturer creates a ManufacturerSupplierOrder the attribute "Day due" is assigned based on the current day plus the number of days for delivery. Everyday suppliers check if any of the order's they have received is due on the current day, if it is due then the supplier sends the SupplierManufacturerDelivery to the manufacturer. See listings 3 and 4 as well as figure 3 for the implementation of this constraint.

Per-component per-day warehouse storage costs – At the end of the day, the sum of products being stored in the warehouse is calculated and multiplied by five to get the storage costs for the day. This is then deducted from the daily profit before the cumulative profit is calculated. See listing 5 for the implementation of this constraint.

An order can only be shipped if there are sufficient components in the warehouse to build all of the smartphones in the order – before assembling a given order the Manufacturer checks the parts for the order against warehouse stock. It will only proceed with assembly if there is enough stock in the warehouse at the time. See listing 6 for the implementation of this constraint.

A maximum of 50 smartphones can be assembled and shipped on one day – When orders are assembled the manufacturer keeps track of how many devices have been assembled so far, the manufacturer will not assemble another order if it would result in more than 50 devices being assembled in total for that day. See listing 7 for the implementation of this constraint.

Penalties for late delivery - When an order is assembled the manufacturer compares the day the order was due with the current day. If the current day is after the due date the penalty is calculated and deducted from the orders profit.

Correct calculation of profit at the end of each day – As orders for the day are assembled the total daily profit for the orders is calculated, deducting part costs and any penalties. At the end of the day warehouse storage costs are also deducted to get the profit for the end of the day. This is then added to the manufacturer’s cumulative profit.

4 Design of Manufacturer Agent Control Strategy

4.1 Initial Strategy

Initially the strategy for the manufacturer agent was as follows:

1. Receive an order from each customer.
2. Receive any deliveries for the day.
3. Check if any accepted orders can be assembled with currently warehouse stock, assemble and ship if possible.
4. Calculate the potential profit for each order using part prices from Supplier 1.
5. Accept the order with the highest positive profit, reject the rest. If none of the orders have a positive profit they will all be rejected for the day.
6. Make a ManufacturerSupplierOrder based off of parts needed to completed any outstanding accepted orders and send the order to Supplier1.
7. End the day

Using this strategy the manufacturer is expected to always produce profit. The manufacturer will only accept orders which will generate a net gain of profit and order the parts for them the same day from Supplier1 so that they can be assembled and shipped the next day.

With this strategy the manufacturer never utilises warehouse storage, components that arrive at the beginning of the day are used to assemble orders on the same day so they are never stored in the warehouse. This removes one of the largest sources of profit loss and means profit will never be less than the profit calculated when an order is received.

Another source of revenue loss which is completely removed using this strategy is late delivery penalties. All orders that are accepted are assembled and shipped the next day and given an order cannot be due on the same day it's made the manufacturer never has to pay late penalties.

4.2 Revised Strategy

The above strategy was eventually revised to include small improvements with the goal of maximising profit. Instead of accepting the single most profitable order each day the manufacturer will see if there are any combinations of orders with the total quantity of devices under 50 that would result in more profit than just the single highest profit order.

This strategy is purely an improvement compared to the initial strategy, it still means there are no late penalties or storage costs but there are more possibilities for ways to make profit. Therefore, it is expected that the revised strategy will perform better and produce more profit for the manufacturer.

5 Experimental Results

Profit made by the manufacturer agent will be used to evaluate the manufacturer's control strategy performance as a better strategy should produce more profit.

Each of the control strategies were implemented and ran over a period of 100 days, at the end of each simulation the cumulative profit is logged and recorded. Each strategy was run 30 times due to the random nature of orders so that an average could be calculated. It was hypothesised that the initial strategy would perform worse than the revised strategy as the revised strategy was a direct improvement. The results for experimentation are in Figure 1:

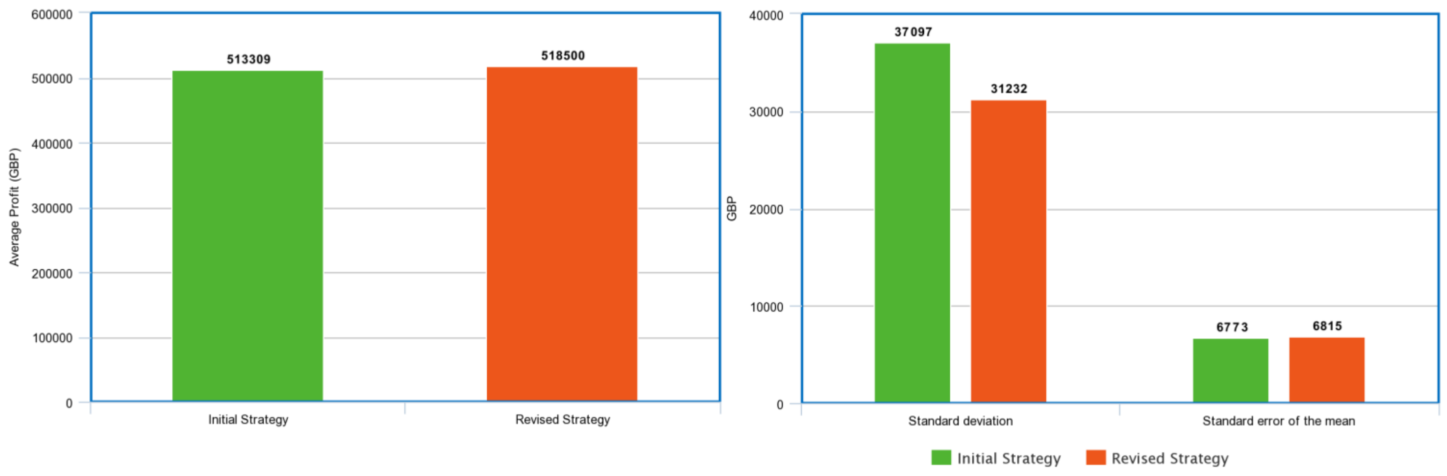


Figure 1 Results for first experiment. Revised strategy only showing an average £5191 improvement.

The difference between the average revenues for the two strategies was not significant enough to distinguish whether or not the revised method was an improvement. This was potentially due to the random nature of orders being produced by customer agents so further testing would be needed to compare the two accurately. Therefore the system was modified temporarily to run both strategies simultaneously so that both strategies were being run using the same data for customer orders. This was repeated 30 times and an average was calculated for each strategy. The results for this second experiment are shown in Figure 2:

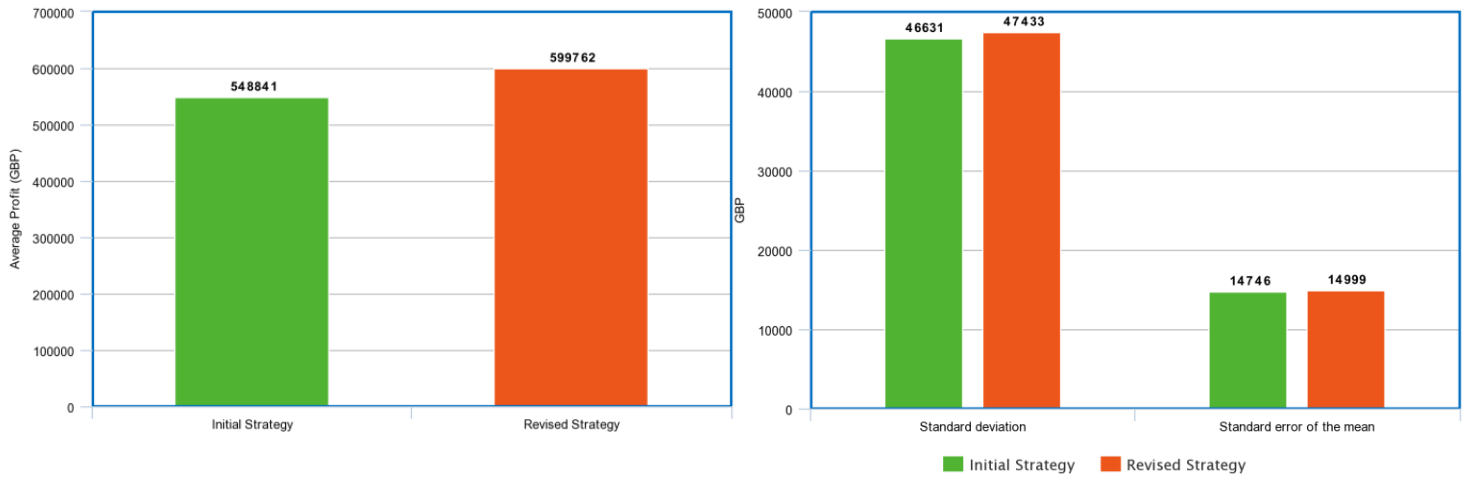


Figure 2 Results for the second experiment. Revised strategy showing an average £50921 improvement.

With this data the advantages of using the revised strategy is clear as both strategies used identical orders for each of the 30 runs with the revised strategy producing an average of £50,921 more for each 100 day run. There was not a single run in which the initial strategy performed better.

Changing the price of warehouse storage does not affect the results for the two strategies as both of them prevent the manufacturer having to pay for storage. Increasing the number of customers, however, was found to result in more profit as the manufacturer has more orders to pick from when choosing which to accept. When running the system with ten customers producing daily orders instead of three it was found the manufacturer would make over twice as much profit on average. Increasing the total number of phones the manufacturer could assemble each day also resulted, as expected, more cumulative profit as the manufacturer can accept more orders each day.

The implemented strategy is not ideal for an increased amount of customers however as most daily orders will be rejected as the manufacturer only accepts up to a maximum of 50 devices worth of orders per day.

6 Conclusion

The implemented strategy for the manufacturer could definitely be improved, utilizing the cheaper parts available from another supplier and managing order assembly to decrease production costs. This would also make more customer orders profitable so there will be less order rejections, which is important for the manufacturer's reputation in a real life supply chain.

Another way the manufacturer could maximise profit further would be to keep spare stock in the warehouse so that orders can possibly be fulfilled without having to wait for a supplier delivery. This would need to be balanced in a way such that the extra warehouse costs would not exceed the additional profit.

To handle a more realistic supply-chain scenario there would need to be much more agent communication implemented. The current system makes an assumption that both suppliers will have stock levels that can fulfil any given order and assumes payment will be received after completing an order. If the system were to be implemented to a more realistic scenario the manufacturer would have to check if a supplier can fulfil an order and also have more interactions with customers to confirm order acceptance, completion and payment.

Although the current ontology works for ensuring information shared between agents is in the correct format and contains all of the needed information, it lacks concepts. For a more robust system a more detailed ontology implementing concepts would need to be developed.

References

Chen, Y., Peng, Y., Finin, T., Labrou, Y., Cost, S., Chu, B., … Wilhelm, B. (2012). A Negotiation-Based Multi-Agent System for Supply Chain Management.

FIPA ACL Message Structure Specification. (2002). Retrieved November 27, 2019, from http://www.fipa.org/specs/fipa00061/SC00061G.html

Appendices

A Ontology

A.1 Predicates

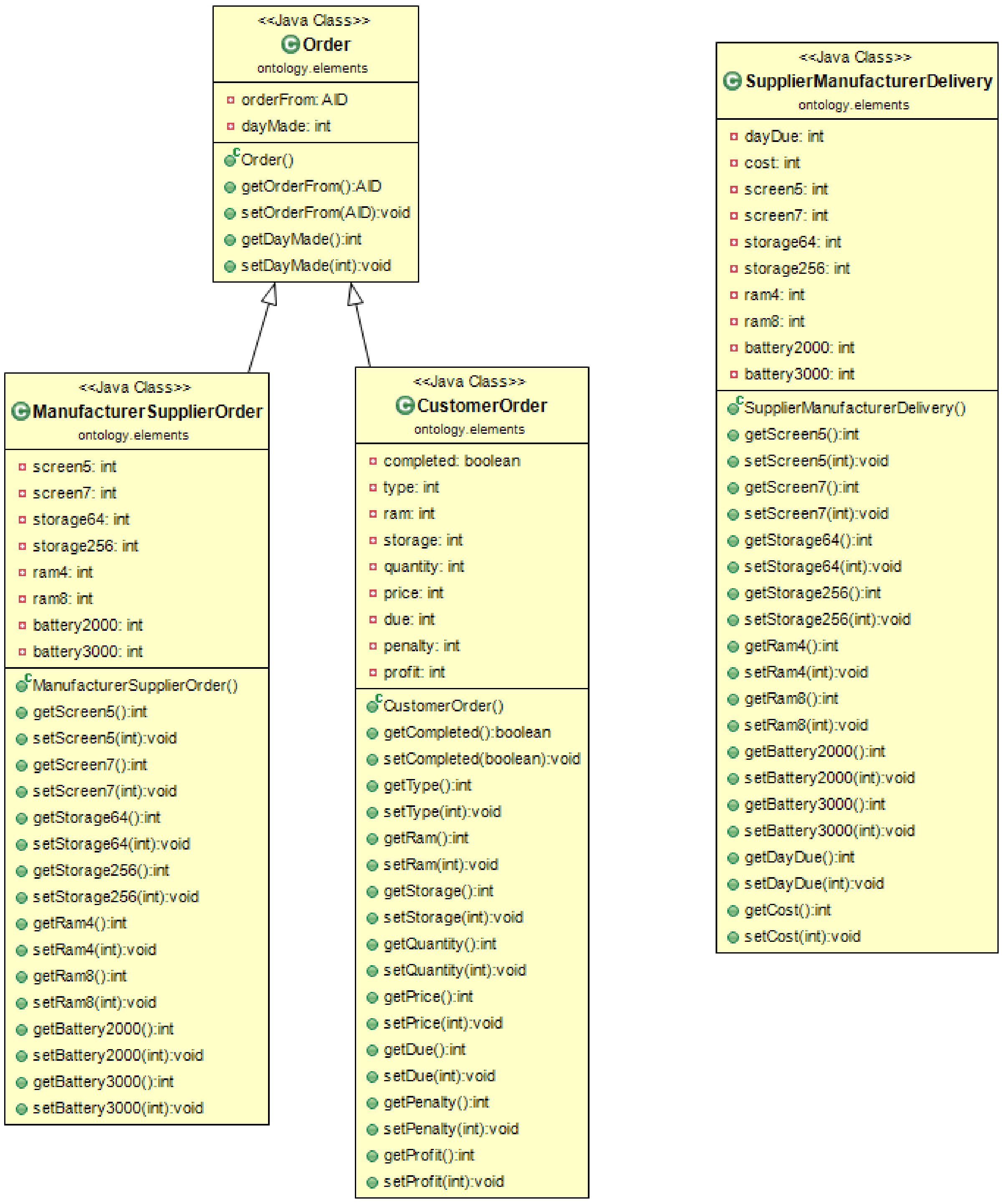
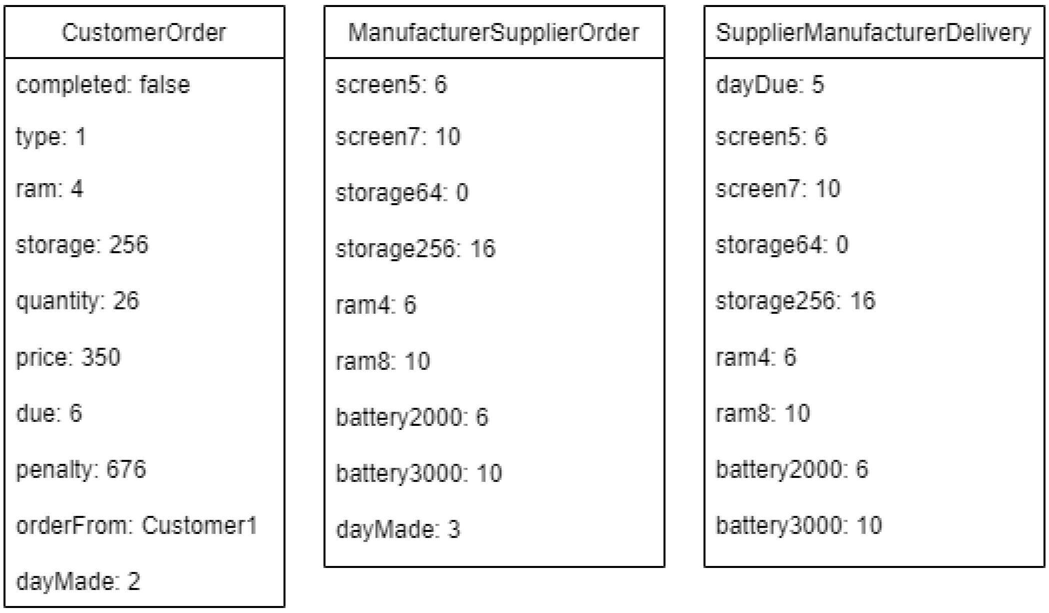


Figure 3 Class diagrams for each of the predicates defined in the ontology

A.2 Predicate Examples



CustomerOrder
completed: false
type: 1
ram: 4
storage: 256
quantity: 26
price: 350
due: 6
penalty: 676
orderFrom: Customer1
dayMade: 2

ManufacturerSupplierOrder
screen5: 6
screen7: 10
storage64: 0
storage256: 16
ram4: 6
ram8: 10
battery2000: 6
battery3000: 10
dayMade: 3

SupplierManufacturerDelivery
dayDue: 5
screen5: 6
screen7: 10
storage64: 0
storage256: 16
ram4: 6
ram8: 10
battery2000: 6
battery3000: 10

Figure 4 Examples of predicate instances

B Communication Protocols

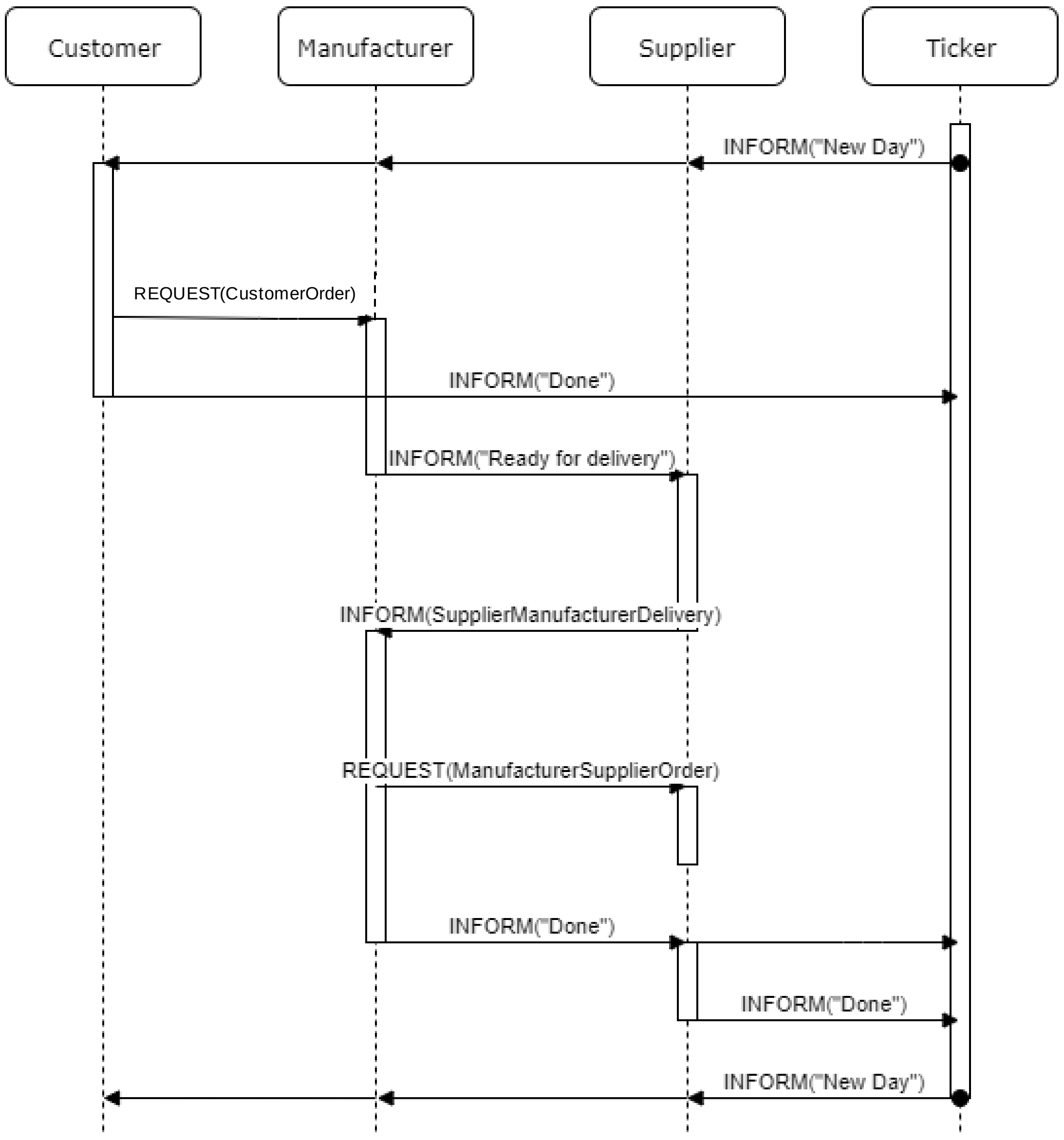


Figure 5 Sequence diagram for agent communication

C Source Code

C.1 Small and phablet smartphone specifications

Customer agent creating an order:

```
// Determine the type of phone
if (Math.random() < 0.5) {
   customerOrder.setType(0);
} else {
   customerOrder.setType(1);
}
```

Listing 1

Manufacturer agent using this information to determine which parts to use for assembly:

```
// Check phone type and use appropriate components to assemble
if (order.getType() == 0) {
   WarehouseScreen5 = WarehouseScreen5 - quantity;
   WarehouseBattery2000 = WarehouseBattery2000 - quantity;
} else {
   WarehouseScreen7 = WarehouseScreen7 - quantity;
   WarehouseBattery3000 = WarehouseBattery3000 - quantity;
}
```

Listing 2

C.2 Component delivery times from the two suppliers

When a supplier receives an order it creates a delivery and gives it a due date (current day plus 1 for supplier 1 and current day plus 4 for suppler 2):

```
if (ce instanceof ManufacturerSupplierOrder) {
   ManufacturerSupplierOrder order = (ManufacturerSupplierOrder) ce;
   SupplierManufacturerDelivery delivery = new SupplierManufacturerDelivery();
   // Omitted code to work out price of the order as it’s irrelevant for this context
   delivery.setCost(price);
   delivery.setDayDue(day + 1);
   deliveries.add(delivery);
   System.out.println("Manufacturer Order received by Supplier1");
}
```

Listing 3

Every day the supplier checks all of its order, if they’re due on that day they are sent to the manufacturer:

```
// Check if delivery is due to arrive at the manufacturer on the current day
if (delivery.getDayDue() == day) {
   ACLMessage orderSend = new ACLMessage(ACLMessage.INFORM);
   orderSend.addReceiver(manufacturerAgent);
   orderSend.setLanguage(codec.getName());
   orderSend.setOntology(ontology.getName());
   try {
      getContentManager().fillContent(orderSend, delivery);
      send(orderSend);
      System.out.println("Delivery shipped to manufacturer");
   } catch (CodecException e) {
      e.printStackTrace();
```

```
    } catch (OntologyException e) {
        e.printStackTrace();
    }
}
```

Listing 4

C.3 Per-component per-day warehouse storage costs

At the end of the day the storage cost is calculated by the manufacturer and deducted from the daily profit:

```
int warehouseStorageCosts = 5 * (WarehouseScreen5 + WarehouseScreen7 + WarehouseBattery2000
    + WarehouseBattery3000 + WarehouseStorage64 + WarehouseStorage256 + WarehouseRam4 +
WarehouseRam8);
dailyProfit = dailyProfit - warehouseStorageCosts - dailyPartsCost;
cumulativeProfit = cumulativeProfit + dailyProfit;
```

Listing 5

C.4 An order can only be shipped if there are sufficient components in the warehouse to build all of the smartphones in the order

Before assembly the manufacturer checks to see if all of the components for an order are available. The count integer starts at 0 and must be increased to 3 in this section of code for the next stage of assembly to take place:

```
if (order.getType() == 0) {
    if ((quantity <= WarehouseScreen5) && (quantity <= WarehouseBattery2000)) {
        count++;
    }
} else if ((quantity <= WarehouseScreen7) && (quantity <= WarehouseBattery3000)) {
    count++;
}
if (order.getStorage() == 64) {
    if (quantity <= WarehouseStorage64) {
        count++;
    }
} else if (quantity <= WarehouseStorage256) {
    count++;
}
if (order.getRam() == 4) {
    if (quantity <= WarehouseRam4) {
        count++;
}
} else if (quantity <= WarehouseRam8) {
    count++;
}
```

Listing 6

C.5 A maximum of 50 smartphones can be assembled and shipped on one day

Before assembly, after checking to see if all of the components are in stock, the manufacturer checks assembling the next order would result in more than 50 devices being assembled for that day:

```
if ((count == 3) && ((50 - devicesAssembled) >= quantity)) {
    // Assembly code omitted as it's irrelevant for this context.
}
```

Listing 7

C.6 Penalties for late delivery

When assembling an order, the manufacturer checks when the order was due, if it was due before the current day then the late penalty is calculated and deducted from the order value:

```
if (order.getDue() < day) {
   totalPenalty = penalty * (day - order.getDue());
} else {
   totalPenalty = 0;
}
value = value - totalPenalty;
```

Listing 8