Multi-Agent Systems Coursework

Edinburgh Napier University

Enrique Belenguer Carrasco
40406742@live.napier.ac.uk

1 Introduction

The aim of this Coursework is design, implement and evaluate a Multi-Agent System[in Jade[1] to model a smartphone manufacturing supply chain[3] scenario[6].

Manufacturing supply chain is the process of planning, implementing and controlling the operations of the supply network in order to meet the customer's needs as effectively as possible. The management of the supply chain goes through all the movement and storage of goods as well as corresponding inventory resulting from the process, and the finished goods from the point of origin to the point of consumption. Proper supply chain management should consider all possible events and factors that may cause a disruption. The chain is formed from many for many components, from the supplier, manufacture to the customer. Traditionally, all those processes were implemented by humans, which involved a long process of communication, cooperation and negotiation. In order to maximize production and improve the entire process, new technologies have been introduced within the manufacturing supply chain, such as multi agent systems. The different components of the chain can be replaced by intelligent software agents which will collaborate to each other increasing the efficiency of the chain.

The objective of any type of distribution chain is to obtain the maximum benefits. In order to to this, the first thing they try to do is reduce all kinds of unnecessary cost[7]. Today companies do so be being innovative and trying to improve communication between the components of the chain, thus avoiding the bullwhip[5] effect.

In a global market as competitive as we are today, traditional methods are becoming less effective.This can be notes especially if we focus on the manufacturing supply chain where many different components can intervene and in which communication and coordination must be the key part. The use of multi agent systems can be very beneficial for those manufacturing supplier chains[2], especially when it comes to automating businesses between buyers and sellers. In addition to that, multi agent systems can help reduce human errors that can cause delays in orders; communication between agents is much more effective with them, in case errors will occur the solution will always be implemented faster. Production can be improved due to the management of these rush times, improving efficiency and executing much better order flow between manufacturers and customers as well as between manufacturers and suppliers.

2 Model Design

Throughout coursework, four agents have been developed for the implementation of the Smartphone Supply Chain using Multi-Agent systems: CustomerAgent, ManufacturerAgent, SupplierAgent, TickerAgent. Those agents will have the following roles:

- CustomerAgents are responsible for the following:
  - Generating smartphone orders and sending them to ManufacturerAgent.
  - Receive the orders and send payment to ManufacturerAgent.
- ManufacturerAgent is responsible for the following:
  - Receiving offers from CustomerAgents and decided which should accept.
  - Requesting components from SupplierAgents.
  - Buying components from SupplierAgents.
  - Receiving components from the SupplierAgents
  - Controlling Stock in the Warehouse.
  - Assembling the Smartphones.
  - Sending orders ready to SupplierAgents.
  - Getting payments from SupplierAgents.
  - Calculating the profit of the day.
- SupplierAgent are responsible for the following:
  - Getting requests from manufacturers and answering them with the availability.
  - Selling components to manufacturers.
- TickerAgent is responsible for the following:
  - Coordinating the days between all the agents.
  - Informing all the agents that new day has started.
  - Informing all the agents that simulation has finished.

In accordance with the ontology, smartphone are made of four different components: Screen, Storage, Memory, Battery.Each of them, including mobile phones, inherits from Item (Appendix 3).Each of the four components has an attribute, in this case it is the same one for all them, the size besides the ItemID coming from the Item. The smartphones, in addition to the ItemID, and each of the attributes of the four components mentioned above, also have a name, which can be Thablet or Phone. This name will depend on the size of the screen. This is the only concept in my system.
In relation to the communication protocols chosen for communication between the client and the manufacturer (Appendix 6), he only protocol used is a FIPA-Request with which the customer request to the manufacturer the previously generated order. The Request message contains the Order (Appendix 5). If the manufacturer accepts the order, he sends a confirmation message to the customer. If the order is rejected, no communication with the client is necessary. Also, if the order is accepted, the manufacturer will inform the customer with the predicate AssemblyOrder that contains the Deliver. Once received, the client will respond with an inform message with the Payment.

In relation to the communication protocols chosen for the communication between the Manufacturer and the Supplier (Appendix 7), the only protocols used are, a FIPA-Request with which the manufacturer requests from the suppliers the Items he needs depending on the orders to be assembled. The Request message contains the BuyComponentsToSuppliers (Sell) (Appendix 5) In case the suppliers have the items they will respond with an Accept_Proposal containing SellingItemToManufactor (Sell). If they do not have the requested Items, they will respond with a message from Reject_Proposal.

Finally, the last communication protocol between the agents is between the TickerAgent and the rest of the agents. No type of FIPA protocols are used in this protocol. Simply Informs messages with which the TickerAgent announces that a new day has begun and the rest of the agents must respond once they have finished all their tasks informing them that they have finished. Once the messages have been received from all the agents, the tickerAgent will send the message again informing the NewDay.

3 Model Implementation

All agents have two commons *Behaviors*. The first is is used for communication between them and the *TickerAgent*, it is a kind of *CyclicBehaviour*. The other similar Behavior is *EndDay*, it is also a *CyclicBehaviour* and is used to report that the day is over and subtract all *DailyVariables*. In addition to these two behaviors that they share (Source Code 1 & Source Code 2) the agents have the following particular Behaviors:

- *CustomerAgents* behaviours:
  - *GenerateOrder().* It is *OneShotBehaviour* which generates the order following the Coursework specification and Requests the order to the Manufacturer Agent using the AgentAction Order()(Source Code 3).
  - *ReceiveAnswerFromManufactures().* It is *OneShotBehaviour* which receives the answer from Manufacturer if the order has been accepted(Source Code 4).
  - *GetOrders().* It is a *CyclicBehaviour* which receives the Smartphones delivery from Manufacturer using a *AgentAction(Deliver)* and execute the Payment(Source Code 5).
- ManufacturerAgent behaviours:
  - *FindCustomersAndSuppliers().* It is *OneShotBehaviour* used to find the CustomersAgents and SuplierAgents at the beginning of each day(Source Code 6).
  - *ReceiveCustomerOrders().* It is a Behaviour which receives the Orders from CustomersAgents and studies them. It decides which day the order is going to be Assembled(Source Code 7).
  - *RequestComponentsSupplier().* It is *OneShotBehaviour* which just adds all the Items from different orders that has to be ordered into a HashMap(Source Code 8).
  - *BuyComponentsToSuppliers().*It is a*Behaviour* which buys the the components needed for the following day. To buy, it sends a Propose message with the *AgentAction(Sell)* to the SupplierAgents(Source Code 9).
  - *GetComponentsFromSuppliers().* It is a Behaviour which receives the answer with the Items from the Suppliers. It also adds the items to the warehouseStock(Source Code 10).
  - *AssemblySmartphones().* It is *OneShotBehaviour* which compares the actual day and assemblyDay of each of the ordersToAssembly and decides to assemble them if the stock in the warehouse allows it. It also sends the orderAssembled to the CustomerAgent using an *AgentAction(Deliver)*(Source Code 11).
  - *GetPaymentsFromSuppliers().* It is a Behaviour which receives the payment from the CustomerAgent and adds it to orderPayment variable which later on will be used to calculate the DailyProfit (Source Code 12).

  - *GetProfit().* It is *OneShotBehaviour* which calculates the DailyProfit and accumulates the TotalProfit of the simulation(Source Code 13).

- SupplierAgent behaviours:
  - *GetStock().* It is *OneShotBehaviour* which loads the Stock into the supplierStock Hashmap(Source Code 14).
  - *SellingItemsToManufactures().* It is a *CyclicBehaviour* which gets the Propose from the Manufactures and answers it with an *AgentAction(Sell)* in case of having the Items in stock, or with a Reject_Proposal if does not have them(Source Code 15).
- TickerAgent does not have any other behaviour apart from the first two mentioned.

In adittion with the other Constrains: the component delivery times(Source Code 16) has been implemented in each supplier. It gets assigned at the same time as the Stock is assigned in GetStock(). The delivery time will depends on the name of the SupplierAgent(Source Code 14).
The per-component-per-day warehouse cost is enforced by the warehouse variable warehouseStorageCost(Source Code 16) in the ManufactureAgent. The value of warehouseStorageCost is multiplied by the numbers of items in the warehouseStock hashmap at the end of the day to calculate the total amount(Source Code 17).
An order can only by shipped if there are sufficient components in the warehouse is enforced in the AssemblySmartphones (Source Code 18). Only if each of the Items are over the orderQuantity the order will be assembled.
A maximum of 50 smartphones can be assembled and shipped on one day is enforced with the array smartphoneDayToAssemby (Source Code 19) and in ReceiveCustomerOrders (Source Code 20) when it assign the dayToAssemblly only if the quantity of smartphone to assembly that day is smaller than the maximum.
Penalties for late deliveries are enforced when assigning the assemblyDay(Source Code 21) in ReceiveCustomerOrders. The assembly day must be a day between the day the order has been received and the dueDate attribute of the Order.
The last one the Constrains is the correct calculation of profit at the end of each day. It is enforced in GetProfit(Source Code 13) at Manufacturer which is going check the daily expenses and it going to extract them from the paymentsRecieved that day from the CustomerAgents.

4 Design of Manufacturer Agent Control Strategy

The strategy that my simulator will follow is calculating if the price offered is greater or less than the price of the components of the Smartphone in particular. In case the price of the Order is greater than the value of the parts to be purchased, the Manufacturer will accept the order. Normally in the real world, the system will be more complex, but for this work I have decided to use this method.

The decision to study the offers at the beginning and deciding that the assembly day is the closest possible and is not the due-date day is based on bullwhip effect[5]. Thus, the manufacturer never stops assembling smartphone even if the due-date is quite far. This way, if more than one offer with a large number of smartphones is received the next day, it will be easier to accept it all.

In the use of the supplier and after studying the difference in prices between them and knowing the per-component-per-day warehouse cost. I have decided that the best way was using just one supplier following the Just-in-time manufacturing methodology[4] which says that the supplies arrive at the factory, or the products to the customer, "just in time", that being shortly before they are used and only in the necessary quantities. This reduces or even eliminates the need for the storage and transfer of the raw materials from the warehouse to the production line .

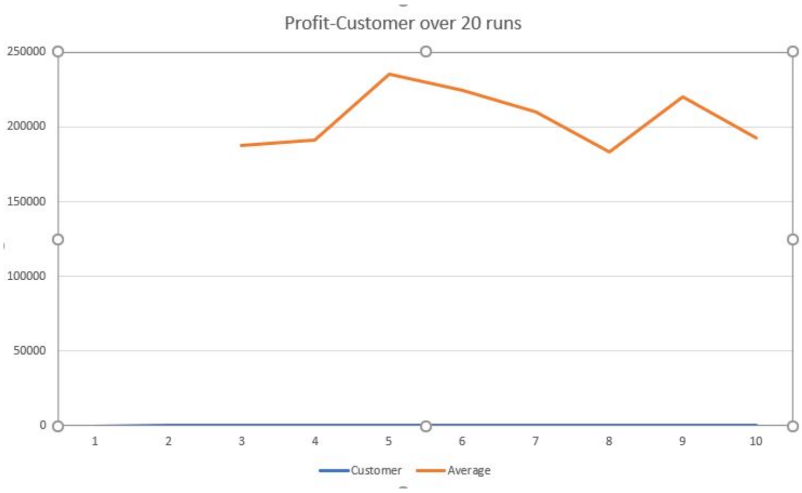
When it comes to which components to place in the warehouse, as mentioned before, the strategy is for the warehouse to be as empty as possible to make sure the warehouse cost is minimum.

Finally, regarding to the order assemble. The manufacturer keeps the tracks of all the orders to assemble, once the day arrives it checks it and because the components have been ordered the day before, it should find always stock in the storage.

5 Experimental Results

For the experimental results, the results will be calculated twenty times with each of the variables and the averages will be calculated. The parameters to be varied are the ones defined in section 2.6, customer(c) and cost of warehouse storage per-day per-component (w). The increase c should lead to an improvement in profit but could stagnate due to the maximum limit of smartphones to assembly. This will not allow the to accept more orders if the limit is reached. The increase of w will have a negative effect to the profit.

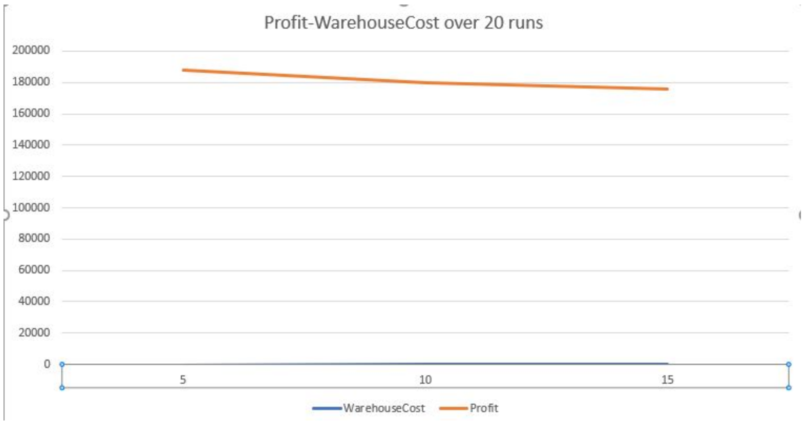
The first parameter tested was c:



1. Profit-Customer over 20 runs

From the results obtained it can be seen that the results vary between £187864 with three customers and the £ 235758.8 obtained with the five customers. The results are more or less as expected, the variance occurs due to the fact that the orders are obtained randomly which influences the performance of the simulator. I suppose that the greater the number of simulations, the smaller the variance.

The second parameter tested was w:



2. Profit-WarehouseCost over 20 runs

From the results obtained it can be seen that the results vary from £187864 obtained with the £5 of the warehouse cost and £ 175658.4 obtained with the £15 of the warehouse cost. The results are more or less as expected, although they should show more variety.The results prove that the Just-in-time methodology works well in manufacture supply chain.

6 Conclusions

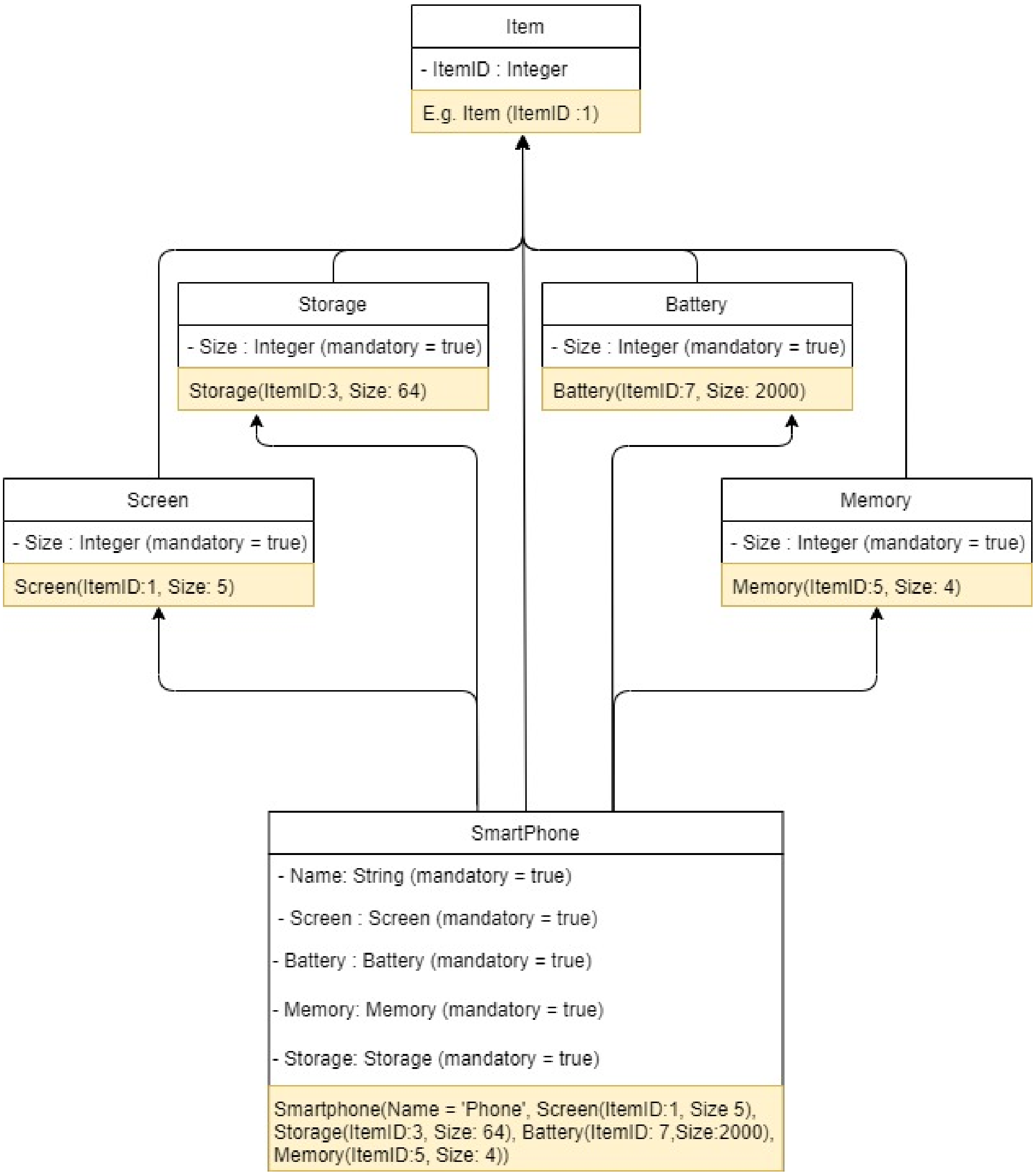
The smartphone manufacturing supply chain implemented in this coursework is a very simplistic version of what it would look in real life. There are many elements that can be added to the simulator to be closer to what a real life version should looks.

- New Suppliers and Costumers should be allowed to join the chain at run time.
- Suppliers should not have undefined stock.
- Supplier's prices should be able to change depending of the demands.
- Warehouse should be able to increase the production in a limited time even if it is against making benefits(it could be a penalty) in order to make regular customers happy by not rejecting their orders.
- Supplier should not't have undefined stock.
- Costumer orders should contains more than one smartphone type.

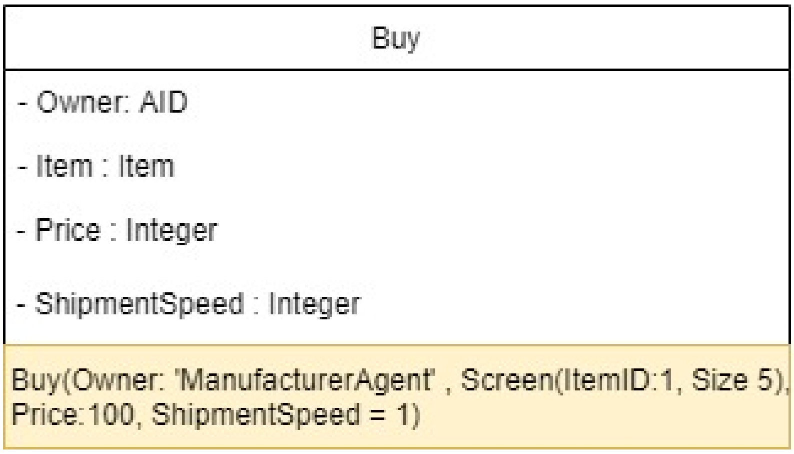
My manufacturer agent control strategy could be improved in many ways. Unfortunately I have not been able to implement what I would have liked. The order management should be improved to be certain that no money is going to be lost with any of the orders. It's easy to implement, however I could not do it because I needed to have orders every day in order to make my simulator work. I have been looking for that error for a long time but I could not find it. In addition to that, I could also implement the assembling of an order in two days, which would allow me to assemble some smartphones and send them one day and finish assembling the remaining ones the next day. Those two things would greatly improve the profit of the manufacturer and I will try to implement it during the Christmas Holidays.

A Appendix

A.1 Ontology


Item
- ItemID : Integer
E.g. Item (ItemID :1)
Storage
- Size : Integer (mandatory = true)
Storage(ItemID:3, Size: 64)
Battery
- Size : Integer (mandatory = true)
Battery(ItemID:7, Size: 2000)
Screen
- Size : Integer (mandatory = true)
Screen(ItemID:1, Size: 5)
Memory
- Size : Integer (mandatory = true)
Memory(ItemID:5, Size: 4)
SmartPhone
- Name: String (mandatory = true)
- Screen : Screen (mandatory = true)
- Battery : Battery (mandatory = true)
- Memory: Memory (mandatory = true)
- Storage: Storage (mandatory = true)
Smartphone(Name = 'Phone', Screen(ItemID:1, Size 5), Storage(ItemID:3, Size: 64), Battery(ItemID: 7,Size:2000), Memory(ItemID:5, Size: 4))

3. Ontology Concepts

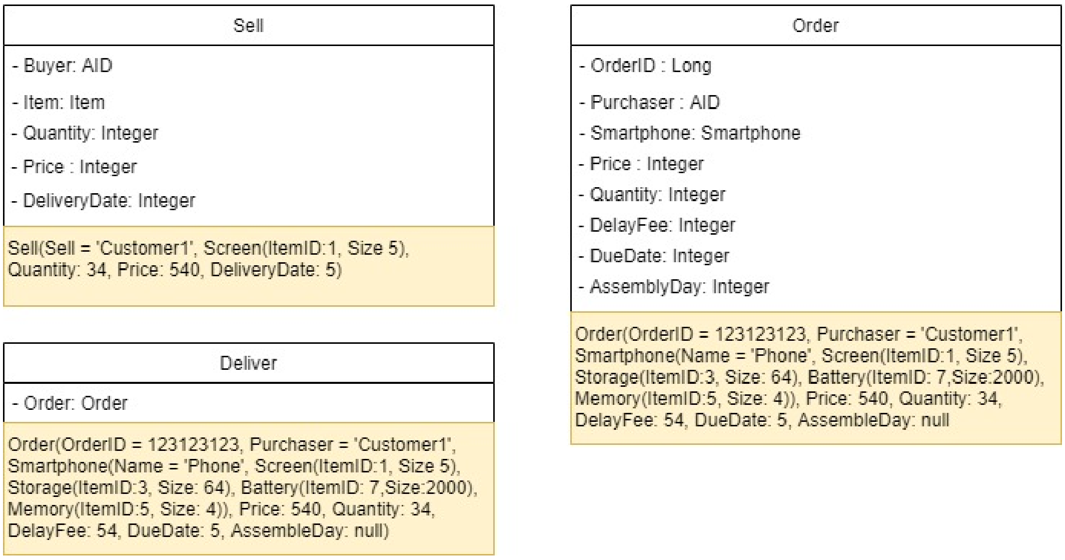


Buy

- Owner: AID
- Item : Item
- Price : Integer
- ShipmentSpeed : Integer

Buy(Owner: 'ManufacturerAgent' , Screen(ItemID:1, Size 5), Price:100, ShipmentSpeed = 1)

4. Predicate Concepts



Sell

- Buyer: AID
- Item: Item
- Quantity: Integer
- Price : Integer
- DeliveryDate: Integer

Sell(Sell = 'Customer1', Screen(ItemID:1, Size 5), Quantity: 34, Price: 540, DeliveryDate: 5)

Deliver

- Order: Order

Order(OrderID = 123123123, Purchaser = 'Customer1', Smartphone(Name = 'Phone', Screen(ItemID:1, Size 5), Storage(ItemID:3, Size: 64), Battery(ItemID: 7,Size:2000), Memory(ItemID:5, Size: 4)), Price: 540, Quantity: 34, DelayFee: 54, DueDate: 5, AssembleDay: null)

Order

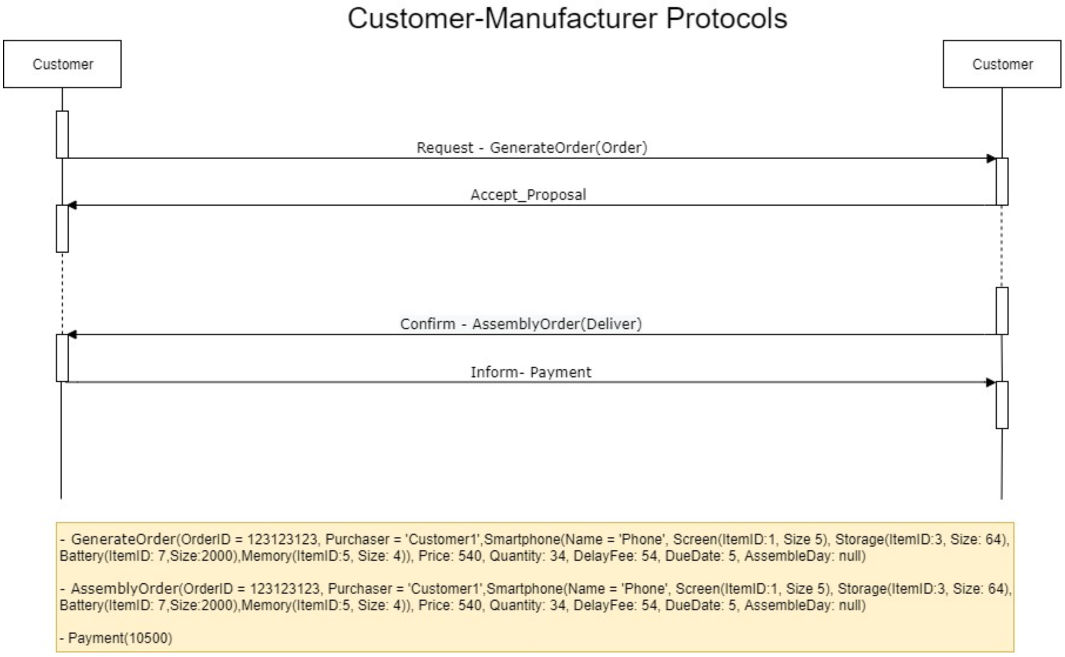
- OrderID : Long
- Purchaser : AID
- Smartphone: Smartphone
- Price : Integer
- Quantity: Integer
- DelayFee: Integer
- DueDate: Integer
- AssemblyDay: Integer

Order(OrderID = 123123123, Purchaser = 'Customer1', Smartphone(Name = 'Phone', Screen(ItemID:1, Size 5), Storage(ItemID:3, Size: 64), Battery(ItemID: 7,Size:2000), Memory(ItemID:5, Size: 4)), Price: 540, Quantity: 34, DelayFee: 54, DueDate: 5, AssembleDay: null)

5. Agent Action Concepts

B Communication Protocol

B.1 Customer-Manufacturer

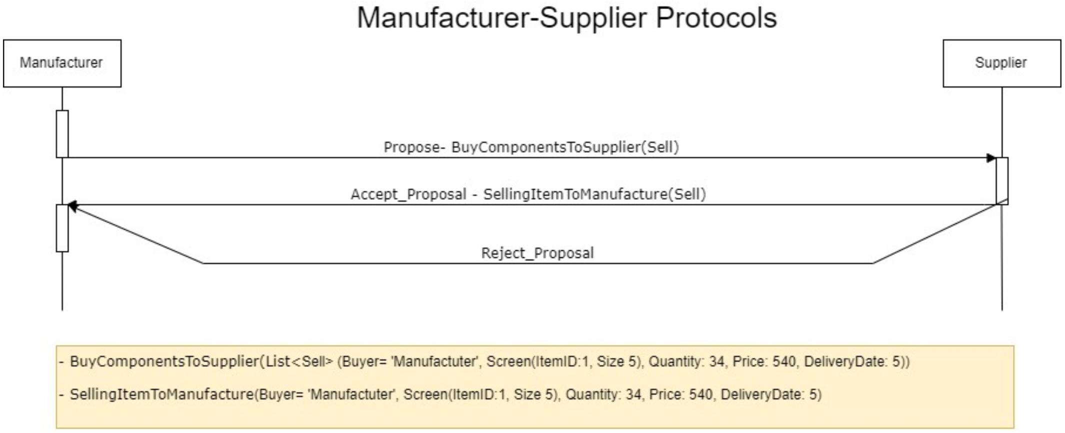

Customer-Manufacturer Protocols
Customer
Customer
Request - GenerateOrder(Order)
Accept_Proposal
Confirm - AssemblyOrder(Deliver)
Inform- Payment

- GenerateOrder(OrderID = 123123123, Purchaser = 'Customer1',Smartphone(Name = 'Phone', Screen(ItemID:1, Size 5), Storage(ItemID:3, Size: 64), Battery(ItemID: 7,Size:2000),Memory(ItemID:5, Size: 4)), Price: 540, Quantity: 34, DelayFee: 54, DueDate: 5, AssembleDay: null)

- AssemblyOrder(OrderID = 123123123, Purchaser = 'Customer1',Smartphone(Name = 'Phone', Screen(ItemID:1, Size 5), Storage(ItemID:3, Size: 64), Battery(ItemID: 7,Size:2000),Memory(ItemID:5, Size: 4)), Price: 540, Quantity: 34, DelayFee: 54, DueDate: 5, AssembleDay: null)

- Payment(10500)

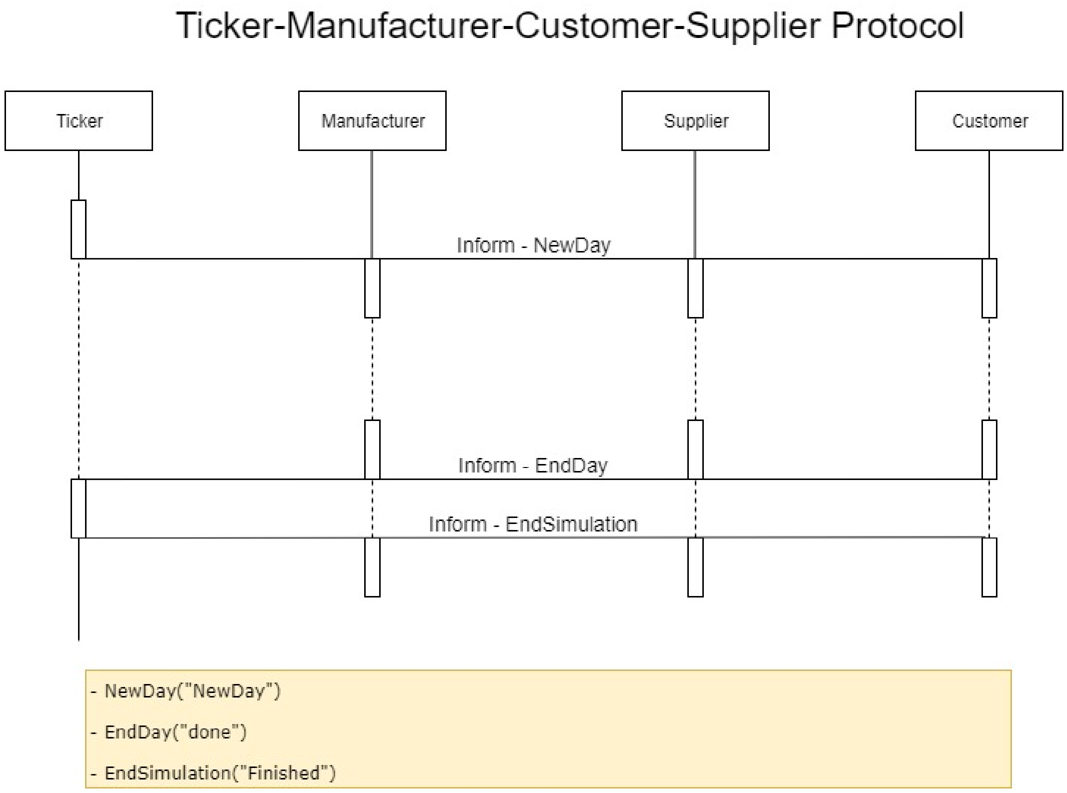
6. Customer-Manufacturer Communication Protocol


Manufacturer-Supplier Protocols
Manufacturer
Supplier
Propose- BuyComponentsToSupplier(Sell)
Accept_Proposal - SellingItemToManufacture(Sell)
Reject_Proposal

- BuyComponentsToSupplier(List<Sell> (Buyer= 'Manufactuter', Screen(ItemID:1, Size 5), Quantity: 34, Price: 540, DeliveryDate: 5))

- SellingItemToManufacture(Buyer= 'Manufactuter', Screen(ItemID:1, Size 5), Quantity: 34, Price: 540, DeliveryDate: 5)

7. Manufacturer Supplier Communication Protocol



8. Ticker-Manufacturer-Customer-Supplier Communication Protocol

C Source Code

Source Code 1: Customer Waiter Behaviour

```java
public class TickerWaiter extends CyclicBehaviour {
   public TickerWaiter(Agent a) {
      super(a);
   }

   @Override
   public void action() {
      MessageTemplate mt =
          MessageTemplate.or(MessageTemplate.MatchContent("NewDay"),
            MessageTemplate.MatchContent("terminate"));
      ACLMessage msgTicker = myAgent.receive(mt);
      if (msgTicker != null) {
         if (tickerAgent == null) {
            tickerAgent = msgTicker.getSender();
         }
         if (msgTicker.getContent().equals("NewDay")) {
            cyclicBehaviours.clear();
            myAgent.addBehaviour(new GenerateOrder());
            myAgent.addBehaviour(new
                ReceiveAnswerFromManufactures());
            CyclicBehaviour gOrders = new GetOrders();
            myAgent.addBehaviour(gOrders);
            cyclicBehaviours.add(gOrders);

            myAgent.addBehaviour(new EndDay(cyclicBehaviours,
                myAgent));
         } else {
            myAgent.doDelete();
         }
      } else {
         block();
      }
   }
}
```

Source Code 2: Customer EndDay Behaviour

```java

   @Override
   public void action() {
         MessageTemplate mt = MessageTemplate.MatchContent("done");
         ACLMessage msgEndDay = myAgent.receive(mt);
         if (msgEndDay != null) {
            if (msgEndDay.getSender().equals(manufacturerAgent)) {
               ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
               msg.setContent("done");
               msg.addReceiver(tickerAgent);
               myAgent.send(msg);
               day++;
               for (Behaviour behaviour : cyclicB) {
                  myAgent.removeBehaviour(behaviour);
               }
               myAgent.removeBehaviour(this);
            }
         } else {
            block();
         }
      }
   }
}
```

Source Code 3: Customer GenerateOrder Behaviour

```java
public class GenerateOrder extends OneShotBehaviour {
   private Order order = new Order();
   private Screen screen = new Screen();
   private Storage storage = new Storage();
   private Memory memory = new Memory();
   private Battery battery = new Battery();
   private Smartphone smartphone = new Smartphone();
   private int quantity;
   private int price;
   private int deliveryDue;
   private int penaltyDelay;
   private Random rand = new Random();

   public void action() {
      /*
       * The item ID will be assigned as follow 1. Screen 5’ 2. Screen 7’ 3. Storage
       * 64Gb 4. Storage 256Gb 5. Memory 4Gb 6. Memory 8Gb 7. Battery 2000mAh 8.
       * Battery 3000mAh
       */
      if (Math.random() < 0.5) {
         screen.setSize(5);
         screen.setItemID(1);
         battery.setSize(2000);
         battery.setItemID(7);
         smartphone.setName("Phone");
      } else {
         screen.setSize(7);
         screen.setItemID(2);
         battery.setSize(3000);
         battery.setItemID(8);
         smartphone.setName("Thablet");
      }
      if (Math.random() < 0.5) {
         storage.setSize(64);
         storage.setItemID(3);

      } else {
         storage.setSize(256);
         storage.setItemID(4);
      }
      if (Math.random() < 0.5) {
         memory.setSize(4);
         memory.setItemID(5);
      } else {
         memory.setSize(8);
         memory.setItemID(6);
      }

      smartphone.setScreen(screen);
      smartphone.setBattery(battery);
      smartphone.setStorage(storage);
      smartphone.setMemory(memory);

      ACLMessage msgOrderSupplier = new
          ACLMessage(ACLMessage.REQUEST);
      msgOrderSupplier.setLanguage(codec.getName());
      msgOrderSupplier.setOntology(ontology.getName());
      msgOrderSupplier.addReceiver(manufacturerAgent);

      quantity = (int) Math.floor(1 + 50 * Math.random());
      price = (int) (Math.floor(100 + 500 * Math.random()));
      deliveryDue = (int) Math.floor(1 + 10 * Math.random());
      penaltyDelay = (int) (quantity * Math.floor(1 + 50 *
          Math.random()));

      order.setPurchaser(myAgent.getAID());
      order.setSmartphone(smartphone);
      order.setQuantity(quantity);
      order.setPrice(price);
      order.setDueDate(deliveryDue + day);
      order.setDelayFee(penaltyDelay);
      order.setOrderID(Math.abs(rand.nextLong()));

      Action orderToSupplier = new Action();
      orderToSupplier.setAction(order);
      orderToSupplier.setActor(manufacturerAgent);

      try {
         getContentManager().fillContent(msgOrderSupplier,
             orderToSupplier);
         send(msgOrderSupplier);
      } catch (CodecException ce) {
```

```java
      ce.printStackTrace();
    } catch (OntologyException oe) {
      oe.printStackTrace();
    }

  }
}

/*
 * ReceiveAnswerFromManufactures receive the answer from the Manufacture in case
 * that the order has been accepted. If haven't been accepted the Manufacture
 * does not have to reply. In case the order is accepted, it added to the
 * workingOrders List
 *
 */
public class ReceiveAnswerFromManufactures extends OneShotBehaviour {
```

Source Code 4: Customer ReceiveAnswerFromManufactures Behaviour

```java
          ContentElement ce = null;
          ce = getContentManager().extractContent(answerFromSuppliers);
          if (ce instanceof Action) {
            Concept action = ((Action) ce).getAction();
            if (action instanceof Order) {
              Order order = (Order) action;
              workingOrders.add(order.getOrderID());
            }
          }
        } catch (CodecException ce) {
          ce.printStackTrace();
        } catch (OntologyException oe) {
          oe.printStackTrace();
        }
      } else {
        return;
      }
    } else {
      block();
```

```
        }
      }
    }

    /*
     * GetOrders wait for the message from the manufacture which contains the
     * Assembled Smartphone If the Order received is on the workingOrders List, it
     * execute the Payment to the Manufacture paymentAmount = orderPrice *
     * orderQuantity; The amount is passed to String to encapsulate it in a
     * messageContent. Once order has been paid, it get deleted from workingOrders.
     * In case the Order Received is not in workingOrders List, it send a
```

Source Code 5: Customer GetOrders Behaviour

```

        MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CONFIRM);
        ACLMessage msgGetOrders = myAgent.receive(mt);
        if (msgGetOrders != null) {
          try {
            ContentElement ce = null;
            ce = getContentManager().extractContent(msgGetOrders);
            if (ce instanceof Action) {
              Concept action = ((Action) ce).getAction();
              if (action instanceof Deliver) {
                Deliver delivery = (Deliver) action;

                if (workingOrders.contains(delivery.getOrder().getOrderID())) {
                  ACLMessage msgPayment = new ACLMessage(ACLMessage.INFORM);
                  msgPayment.setConversationId("PaymentFromCustomerToManu");
                  msgPayment.addReceiver(msgGetOrders.getSender());

                  orderQuantity = delivery.getOrder().getQuantity();
```

```
                            orderPrice = delivery.getOrder().getPrice();
                            paymentAmount = orderPrice * orderQuantity;
                            // It pass the Integer to String
                            msgPayment.setContent(Integer.toString(paymentAmount));

                            myAgent.send(msgPayment);

                            workingOrders.remove(delivery.getOrder().getOrderID());
                        } else {
                            ACLMessage msgWrong = new ACLMessage(ACLMessage.FAILURE);
                            msgWrong.setConversationId("PaymentFromCustomerToManu");
                            msgWrong.setContent("PaymentWrong");
                            msgWrong.addReceiver(msgGetOrders.getSender());

                            myAgent.send(msgWrong);
                        }
                    }
                }
            } catch (CodecException ce) {
                ce.printStackTrace();
            } catch (OntologyException oe) {
                oe.printStackTrace();
            }
        } else {
            block();
        }
    }
}

/*
 * EndDay is a CyclicBehaviour that restart the CustomerAgent once get the done
 * message from the manufacture. It also delete all the behaviour from the day
 * finishing.
 *
```

Source Code 6: Manufacturer FindCustomersAndSuppliers Behaviour

```
public class FindCustomersAndSuppliers extends OneShotBehaviour{
    public FindCustomersAndSuppliers(Agent a){
        super(a);
```

```
   }

   @Override
   public void action(){
      DFAgentDescription customerTemplate = new
          DFAgentDescription();
      ServiceDescription csd = new ServiceDescription();
      csd.setType("Customer");
      customerTemplate.addServices(csd);

      DFAgentDescription supplierTemplate = new
          DFAgentDescription();
      ServiceDescription ssd = new ServiceDescription();
      ssd.setType("Supplier");
      supplierTemplate.addServices(ssd);

      try  {
         customersAgent.clear();
         DFAgentDescription[] custAgent =
             DFService.search(myAgent, customerTemplate);
         for(int i = 0; i<custAgent.length; i++) {
            customersAgent.add(custAgent[i].getName());
         }
         suppliersAgent.clear();
         DFAgentDescription[] supplierAgent =
             DFService.search(myAgent, supplierTemplate);
         for(int i = 0; i<supplierAgent.length; i++){
            suppliersAgent.add(supplierAgent[i].getName());
         }
      }
      catch (FIPAException fe){
         fe.printStackTrace();
      }
   }
}
```

Source Code 7: Manufacturer ReceiveCustomerOrders Behaviour

```
public class ReceiveCustomerOrders extends Behaviour{

   private int numOrders;
   private int totalQ;
   public ReceiveCustomerOrders(Agent a){
       super(a);
    }

    @Override
    public void action(){
       int minPric = 15;
       MessageTemplate mt =
           MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
       ACLMessage order = myAgent.receive(mt);

       if(order != null){
          numOrders++;
          try  {
             ContentElement ce = null;
             ce = getContentManager().extractContent(order);
             if(ce instanceof Action){
                Concept action = ((Action)ce).getAction();
                if(action instanceof Order){
                   Order custOrder = (Order)action;
                   int dueToDeliver = custOrder.getDueDate();
                   int numberToDeliver =
                       smartphoneDayToAssembly[dueToDeliver];
                   int quantityInOrder = custOrder.getQuantity();

    /*
     * Here is where I accept the order checking if can be
         assembly on the day. minPric is a variable
     * that has been added at the end to try to improve the profit
         of the Chain. It was going to
     * depend of the Order, so it wouln't accept any order that
         will make me lose money. But due
     * some problem I couldn't fix, the simulation brakes when
         there is not order to assembly that day.
     * I "made" that solution that is not ideal, min price changes
         depending of the day, so the first
     * 5 days i make sure i add some orders and from day 5 the
         minimum price increase to 200. This way
     * even if some days the simulation lose money which is not
         ideal it keeps making profit at the end
     * of the whole simulation. The "good version" would be
         without if(smartphoneDayToAssembly[day + 1] == 0)
     *
     * The forLoop what does is check if from the Day we are until
```

```
          the day DueToDelivery the order can be
       * assembled. In case there is space it accept the order and
          set a setAssemblyDay for that day. In case
       * either the price is less than minPric or it can be
          assembled before the dueDay, the order get refuse

       */
                    if(day < 5)
                       minPric = 100;
                    else
                       minPric = 200;


                    if(custOrder.getPrice() >= minPric) {
                       if(smartphoneDayToAssembly[day + 1] == 0) {
                          numberToDeliver =
                              smartphoneDayToAssembly[day+1];
                          totalQ = smartphoneDayToAssembly[day+1] +
                              quantityInOrder;
                          smartphoneDayToAssembly[day+1] = totalQ;
                          custOrder.setAssemblyDay(day+1);
                          acceptedOrders.add(custOrder);
                       }
                       else
                       for(int x = day ; x <= dueToDeliver ; x++) {
                          numberToDeliver =
                              smartphoneDayToAssembly[x];
                          if(x > TickerAgent.NUM_DAYS + 1 ){
                             rejectedOrders.add(custOrder);
                          }else if(x > dueToDeliver) {
                             rejectedOrders.add(custOrder);
                          }else if(numberToDeliver + quantityInOrder
                              <= assemblyMax) {
                             totalQ = smartphoneDayToAssembly[x] +
                                 quantityInOrder;
                             smartphoneDayToAssembly[x] = totalQ;
                             custOrder.setAssemblyDay(x);
                             acceptedOrders.add(custOrder);
                             break;
                          }
                       }
                     }
                     else {
```

```
                  rejectedOrders.add(custOrder);
                }
              }
            }
          }catch (CodecException ce){
            ce.printStackTrace();
          }catch (OntologyException oe){
            oe.printStackTrace();
          }
        }else{
          block();
        }

        //Go here once has received all the orders from the Customers
        if(numOrders == customersAgent.size()){
          for(int x = 0; x < acceptedOrders.size(); x++) {
            //Add the order to workingOrders
            workingOrders.add(acceptedOrders.get(x));

            ACLMessage accepted = new
                ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
            accepted.setLanguage(codec.getName());
            accepted.setOntology(ontology.getName());
            accepted.addReceiver(acceptedOrders.get(x).getPurchaser());
            accepted.setConversationId("ManufactureAnswerToCustomer");

            Order ord = acceptedOrders.get(x);
            Action sendReply = new Action();
            sendReply.setAction(ord);
            sendReply.setActor(acceptedOrders.get(x).getPurchaser());

            try{
              getContentManager().fillContent(accepted, sendReply);
              send(accepted);
            }catch (CodecException ce){
              ce.printStackTrace();
            }catch (OntologyException oe){
              oe.printStackTrace();
            }

          }

          for(int x = 0; x < rejectedOrders.size(); x++) {
            ACLMessage rejected = new
               ACLMessage(ACLMessage.REFUSE);
            rejected.setLanguage(codec.getName());
            rejected.setOntology(ontology.getName());
            rejected.addReceiver(rejectedOrders.get(x).getPurchaser());
            rejected.setConversationId("ManufactureAnswerToCustomer");
            //Reject the orders in rejectedOrders
            Order ord = rejectedOrders.get(x);
            Action sendReply = new Action();
            sendReply.setAction(ord);
            sendReply.setActor(rejectedOrders.get(x).getPurchaser());

            try{
               getContentManager().fillContent(rejected, sendReply);
               send(rejected);
            }catch (CodecException ce){
               ce.printStackTrace();
            }catch (OntologyException oe){
               oe.printStackTrace();
            }
         }
      }
   }

   @Override
   public boolean done(){
      return numOrders == customersAgent.size();
   }
}
```

Source Code 8: Manufacturer RequestComponentsSupplier Behaviour

```
public class RequestComponentsSupplier extends OneShotBehaviour {
   int ordersToSend = 0;
   int ordersSent = 0;
   @Override
   public void action(){
      for(int x = 0; x < workingOrders.size(); x++) {
         Storage storage =
            workingOrders.get(x).getSmartphone().getStorage();
         Battery battery =
            workingOrders.get(x).getSmartphone().getBattery();
         Screen screen =
            workingOrders.get(x).getSmartphone().getScreen();
         Memory memory =
            workingOrders.get(x).getSmartphone().getMemory();
         int quantity = workingOrders.get(x).getQuantity();
         //It add the items to Buy that has to be order today to
            receive it tomorrow
         if(workingOrders.get(x).getAssemblyDay() - day == 1){
           ordersToSend ++;
           if(toBuy1.containsKey(screen)){
             toBuy1.put(screen, (toBuy1.get(screen) + quantity));
           }else{
             toBuy1.put(screen, quantity);
           }
           if(toBuy1.containsKey(memory)){
             toBuy1.put(memory, (toBuy1.get(memory) + quantity));
           }else{
             toBuy1.put(memory, quantity);
           }
           if(toBuy1.containsKey(storage)){
             toBuy1.put(storage, (toBuy1.get(storage) +
                 quantity));
           }else{
             toBuy1.put(storage, quantity);
           }
           if(toBuy1.containsKey(battery)){
             toBuy1.put(battery, (toBuy1.get(battery) +
                 quantity));
           }else{
             toBuy1.put(battery, quantity);
           }
           ordersToAssembly.add(workingOrders.get(x));
           workingOrders.remove(x);
         }
      }
   }
}
```

Source Code 9: Manufacturer BuyComponentsToSuppliers Behaviour

```
public class BuyComponentsToSuppliers extends Behaviour{
  int sent = 0;
  AID supplier1;
  AID supplier2;
```

```java
   public void action(){

      for(int x = 0; x < suppliersAgent.size(); x++) {
         if(suppliersAgent.get(x).getName().contains("Supplier_1")){
            supplier1 = suppliersAgent.get(x);
         }else {
            supplier2 = suppliersAgent.get(x);
         }
      }
      //Order the items expected for tomorrow
      for(Item key : toBuy1.keySet()) {
         ACLMessage msgBuyCompSupp1 = new ACLMessage(ACLMessage.PROPOSE);
         msgBuyCompSupp1.setLanguage(codec.getName());
         msgBuyCompSupp1.setOntology(ontology.getName());
         msgBuyCompSupp1.addReceiver(supplier1);

         Sell sell = new Sell();
         sell.setBuyer(getAID());
         sell.setItem(key);
         sell.setQuantity(toBuy1.get(key));

         Action myOrder = new Action();
         myOrder.setAction(sell);
         myOrder.setActor(myAgent.getAID());


         try {
            getContentManager().fillContent(msgBuyCompSupp1, myOrder);
         } catch (CodecException | OntologyException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
         }
         send(msgBuyCompSupp1);
         sent++;
      }

   }
   public boolean done(){
      return sent == toBuy1.size();
   }
}
```

Source Code 10: Manufacturer GetComponentsFromSuppliers Behaviour

```java
public class GetComponentsFromSuppliers extends Behaviour{
   int noReplies = 0;
   public void action(){
      MessageTemplate mt =
          MessageTemplate.or(MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROP
            MessageTemplate.MatchPerformative(ACLMessage.REJECT_PROPOSAL));
      ACLMessage msgGetCompSup = myAgent.receive(mt);
      if(msgGetCompSup != null){
         noReplies++;
         if(msgGetCompSup.getPerformative() ==
             ACLMessage.ACCEPT_PROPOSAL){
            try  {
               ContentElement ce = null;
               //If the answer is positive add the order to
                   openDeliveris
               ce =
                   getContentManager().extractContent(msgGetCompSup);
               if(ce instanceof Action){
                  Concept action = ((Action)ce).getAction();
                  if(action instanceof Sell){
                     Sell order = (Sell)action;
                     openDeliveries.add(order);
                     componentCost = componentCost +
                         order.getPrice();
                  }
               }
            }catch (CodecException ce){
               ce.printStackTrace();
            }catch (OntologyException oe){
               oe.printStackTrace();
            }
         }
      }
      else{
         block();
      }
      // Add the order components from the order accepted to the
          warehouse
      if(!openDeliveries.isEmpty()){
         for(Sell order : openDeliveries){
            //If component exist, it increase the quantity and if
```

```
                not it add a new item
            if(warehouseStock.containsKey(order.getItem().getItemID())){
               int quantity =
                 warehouseStock.get(order.getItem().getItemID());
               warehouseStock.put(order.getItem().getItemID(),
                 quantity + order.getQuantity());
            }else{
               warehouseStock.put(order.getItem().getItemID(),
                 order.getQuantity());
            }
         }
      }else{
         block();
      }
      openDeliveries.clear();

      if(toBuy1.size() == 0) {
         return;
      }
   }

   public boolean done(){
      return noReplies == toBuy1.size();
   }
}
```

Source Code 11: Manufacturer AssemblySmartphones Behaviour

```
public class GetPaymentsFromSuppliers extends Behaviour{
   int msgReceived = 0;

   public void action(){
      MessageTemplate mt =
        MessageTemplate.MatchConversationId("PaymentFromCustomerToManu");
      ACLMessage order = myAgent.receive(mt);
      if(order!=null) {
         msgReceived++;
         //Get the payment from customer
         if(order.getPerformative() == ACLMessage.INFORM){
            try  {
               int payment = Integer.parseInt(order.getContent());
               orderPayment = orderPayment + payment;
            }catch (NumberFormatException nfe){
```

```java
               nfe.printStackTrace();
            }
         }
      }
   }

   public boolean done(){
      return ordersSent == msgReceived;
   }
}
```

Source Code 12: Manufacturer GetPaymentsFromSuppliers Behaviour

```java
public class RequestComponentsSupplier extends OneShotBehaviour {
   int ordersToSend = 0;
   int ordersSent = 0;
   @Override
   public void action(){
      for(int x = 0; x < workingOrders.size(); x++) {
         Storage storage =
             workingOrders.get(x).getSmartphone().getStorage();
         Battery battery =
             workingOrders.get(x).getSmartphone().getBattery();
         Screen screen =
             workingOrders.get(x).getSmartphone().getScreen();
         Memory memory =
             workingOrders.get(x).getSmartphone().getMemory();
         int quantity = workingOrders.get(x).getQuantity();
         //It add the items to Buy that has to be order today to
             receive it tomorrow
         if(workingOrders.get(x).getAssemblyDay() - day == 1){
            ordersToSend ++;
            if(toBuy1.containsKey(screen)){
               toBuy1.put(screen, (toBuy1.get(screen) + quantity));
            }else{
               toBuy1.put(screen, quantity);
            }
            if(toBuy1.containsKey(memory)){
               toBuy1.put(memory, (toBuy1.get(memory) + quantity));
            }else{
               toBuy1.put(memory, quantity);
            }
            if(toBuy1.containsKey(storage)){
               toBuy1.put(storage, (toBuy1.get(storage) + quantity));
            }else{
               toBuy1.put(storage, quantity);
            }
            if(toBuy1.containsKey(battery)){
               toBuy1.put(battery, (toBuy1.get(battery) + quantity));
            }else{
               toBuy1.put(battery, quantity);
            }
            ordersToAssembly.add(workingOrders.get(x));
            workingOrders.remove(x);
         }
      }
   }
}
```

Source Code 13: Manufacturer GetProfit Behaviour

```
public class GetProfit extends OneShotBehaviour{
   public void action(){
      int warehouseCost = 0;
      int lateDelivery = 0;
      int dailyProfit = 0;
      //Calculate profit as explained in the coursework
      if(!warehouseStock.isEmpty()){
         for(Integer i : warehouseStock.values()){
            warehouseCost = warehouseCost + (i * warehouseStorageCost);
         }
      }

      dailyProfit = orderPayment - warehouseCost - lateDelivery - componentCost;
      totalProfit = totalProfit + dailyProfit;
      System.out.println("Total profit: " + totalProfit);
   }
```

Source Code 14: Manufacturer GetStock Behaviour

```
public class GetStock extends OneShotBehaviour{
   @Override
   public void action()
```

```
    {
      /*
       *
       * The item ID will be assigned as follow
       *  1. Screen 5'
       *    2. Screen 7'
       *  3. Storage 64Gb
       *  4. Storage 256Gb
       *  5. Memory 4Gb
       *  6. Memory 8Gb
       *  7. Battery 2000mAh
       *  8. Battery 3000mAh
       *
       * */

      supplierStock.clear();

      if (getAID().getName().contains("Supplier_1")) {
        Screen screen = new Screen();
        screen.setSize(5);
        screen.setItemID(1);
        supplierStock.put(screen.getItemID(), 100);
        Screen screen2 = new Screen();
        screen2.setSize(7);
        screen2.setItemID(2);
        supplierStock.put(screen2.getItemID(), 150);

        Storage storage = new Storage();
        storage.setSize(64);
        storage.setItemID(3);
        supplierStock.put(storage.getItemID(), 25);
        Storage storage2 = new Storage();
        storage2.setSize(256);
        storage2.setItemID(4);
        supplierStock.put(storage2.getItemID(), 50);

        Memory memory = new Memory();
        memory.setSize(4);
        memory.setItemID(5);
        supplierStock.put(memory.getItemID(),30);
        Memory memory2 = new Memory();
        memory2.setSize(8);
        memory2.setItemID(6);
```

```
         supplierStock.put(memory2.getItemID(),60);

         Battery battery = new Battery();
         battery.setSize(2000);
         battery.setItemID(7);
         supplierStock.put(battery.getItemID(),70);
         Battery battery2 = new Battery();
         battery2.setSize(3000);
         battery2.setItemID(8);
         supplierStock.put(battery2.getItemID(),100);

         shipmentSpeed = 1;
      } else {
         Storage storage = new Storage();
         storage.setSize(64);
         storage.setItemID(3);
         supplierStock.put(storage.getItemID(), 15);
         Storage storage2 = new Storage();
         storage2.setSize(256);
         storage2.setItemID(4);
         supplierStock.put(storage2.getItemID(), 40);

         Memory memory = new Memory();
         memory.setSize(4);
         memory.setItemID(5);
         supplierStock.put(memory.getItemID(),20);
         Memory memory2 = new Memory();
         memory2.setSize(8);
         memory2.setItemID(6);
         supplierStock.put(memory2.getItemID(),35);

         shipmentSpeed = 4;
      }
   }
}
```

Source Code 15: Manufacturer SellingItemsToManufactures Behaviour

```
public class SellingItemsToManufactures extends CyclicBehaviour{

   @Override
   public void action(){
      MessageTemplate mt =
         MessageTemplate.MatchPerformative(ACLMessage.PROPOSE);
     ACLMessage msg = myAgent.receive(mt);
     if(msg != null){
        try  {
           ContentElement ce = null;
           ce = getContentManager().extractContent(msg);
           if(ce instanceof Action){
              Concept action = ((Action)ce).getAction();
              if(action instanceof Sell){
                 Sell sell = (Sell)action;
                 if(supplierStock.containsKey(sell.getItem().getItemID())){
                    ACLMessage answerToManu = new
                        ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
                    answerToManu.setLanguage(codec.getName());
                    answerToManu.setOntology(ontology.getName());
                    answerToManu.addReceiver(sell.getBuyer());

                    sell.setDeliveryDate(day + shipmentSpeed);
                    sell.setPrice(supplierStock.get(sell.getItem().getItemID())
                        * sell.getQuantity());

                    Action myReply = new Action();
                   myReply.setAction(sell);
                   myReply.setActor(getAID());
                   getContentManager().fillContent(answerToManu,
                        myReply);
                   send(answerToManu);
                 }else{
                   ACLMessage fail = new
                        ACLMessage(ACLMessage.REJECT_PROPOSAL);
                   fail.addReceiver(sell.getBuyer());
                   myAgent.send(fail);
                 }
              }
           }
        }
        catch (CodecException ce){
           ce.printStackTrace();
        }catch (OntologyException oe) {
           oe.printStackTrace();
        }
     }else{
        block();
```

```
     }
   }
}
```

Source Code 16: Manufacturer warehouseStorageCost Constrain

```
private int day = 1;
private int ordersSent = 0;
private int warehouseStorageCost = 5;
private int componentCost = 0; //
private int orderPayment = 0;
private int totalProfit;
```

Source Code 17: Manufacturer warehouseStorageCost Constrain

```
     if(!warehouseStock.isEmpty()){
        for(Integer i : warehouseStock.values()){
          warehouseCost = warehouseCost + (i *
              warehouseStorageCost);
        }
     }
```

Source Code 18: Manufacturer warehouseStorageCost Constrain

```
             if(warehouseStock.get(memory) >=
                ordersToAssembly.get(i).getQuantity() &&
                  warehouseStock.get(storage) >=
                     ordersToAssembly.get(i).getQuantity() &&
                  warehouseStock.get(screen) >=
                     ordersToAssembly.get(i).getQuantity() &&
                  warehouseStock.get(battery) >=
                     ordersToAssembly.get(i).getQuantity())
             {
                //Decrease the quantity from the warehouseStock
                warehouseStock.put(screen,
                   (warehouseStock.get(screen) -
                   order.getQuantity()));
                warehouseStock.put(battery,
                   (warehouseStock.get(battery) -
                   order.getQuantity()));
                warehouseStock.put(memory,
                   (warehouseStock.get(memory) -
                   order.getQuantity()));
                warehouseStock.put(storage,
                   (warehouseStock.get(storage) -
                   order.getQuantity()));

                ACLMessage msg = new
                   ACLMessage(ACLMessage.CONFIRM);
                msg.addReceiver(order.getPurchaser());
                msg.setLanguage(codec.getName());
                msg.setOntology(ontology.getName());

                Deliver deliver = new Deliver();
                deliver.setOrder(order);

                Action myDelivery = new Action();
                myDelivery.setAction(deliver);
                myDelivery.setActor(getAID());

                getContentManager().fillContent(msg, myDelivery);
                send(msg);

                ordersSent++;
                ordersToAssembly.remove(order);
             }
```

Source Code 19: Manufacturer assemblyMax Constrain

```
private int assemblyMax = 50;
private int [] smartphoneDayToAssembly = new int[140]; //Tt's an
  array where I keep the phone to be assemble
```

Source Code 20: Manufacturer assemblyMax Constrain

```
           }else if(numberToDeliver + quantityInOrder
             <= assemblyMax) {
             totalQ = smartphoneDayToAssembly[x] +
               quantityInOrder;
             smartphoneDayToAssembly[x] = totalQ;
             custOrder.setAssemblyDay(x);
             acceptedOrders.add(custOrder);
             break;
           }
```

Source Code 21: Manufacturer lateDelivery Constrain

```
         for(int x = day ; x <= dueToDeliver ; x++) {
```

```
                         numberToDeliver =
                            smartphoneDayToAssembly[x];
                         if(x > TickerAgent.NUM_DAYS + 1 ){
                            rejectedOrders.add(custOrder);
                         }else if(x > dueToDeliver) {
                            rejectedOrders.add(custOrder);
                         }else if(numberToDeliver + quantityInOrder
                            <= assemblyMax) {
                            totalQ = smartphoneDayToAssembly[x] +
                               quantityInOrder;
                            smartphoneDayToAssembly[x] = totalQ;
                            custOrder.setAssemblyDay(x);
                            acceptedOrders.add(custOrder);
                            break;
                         }
                      }
```

References

[1] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. “JADE: a FIPA2000 compliant agent development environment”. In: *Proceedings of the fifth international conference on Autonomous agents*. ACM. 2001, pp. 216–217.

[2] Brahim Chaib-draa and Jörg Müller. *Multiagent based supply chain management*. Vol. 28. Springer Science & Business Media, 2006.

[3] Eon-Kyung Lee, Sungdo Ha, and Sheung-Kown Kim. “Supplier selection and management system considering relationships in supply chain management”. In: *IEEE Transactions on Engineering Management* 48.3 (Sept. 2001), pp. 307–318. ISSN: 1558-0040. DOI: 10.1109/17.946529.

[4] Mark Huson and Dhananjay Nanda. “The impact of just-in-time manufacturing on firm performance in the US”. In: *Journal of Operations Management* 12.3-4 (1995), pp. 297–310.

[5] Hau L Lee, Venkata Padmanabhan, and Seungjin Whang. “The bullwhip effect in supply chains”. In: *Sloan management review* 38 (1997), pp. 93–102.

[6] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[7] Zhenxin Yu, Hong Yan, and TC Edwin Cheng. “Benefits of information sharing with supply chain partnerships”. In: *Industrial management & Data systems* 101.3 (2001), pp. 114–121.