



同济经管
TONGJI SEM

基于 AnyLogic 的三阶供应链仿真分析

管理建模与仿真期末课程设计报告

封钰震 1951362 龚海瑜 1952775

武丹宁 1952145 竺瑞航 1952346

2021-6

目录

一、建模背景	1
1.1 建模目标	2
1.2 理论背景	2
a . 定期订货法和定期订货法	2
b . 配送模式描述	2
1.3 贡献、发现和结论	3
1.4 理论探讨及应用	3
二、模型介绍	4
2.1 假设条件	4
2.2 概念模型	4
2.3 模型功能布局	4
2.4 模型界面布局	8
三、建模	11
3.1 模块化方案	11
3.2 智能体设计	16
3.3 智能体交互	23
3.4 输入与输出	24
四、课程反馈	25
4.1 组内分工	25
4.2 反思与总结	25
五、附件清单	26

一、建模背景

1.1 建模目标

库存控制作为企业供需平衡的一种手段,是保证生产经营活动的正常运作和降低供需所带来的经营风险的需要。科学的库存决策是企业降低成本,提高服务水平,对顾客的需求做出快速响应的有力保障,同时也是避免因原材料市场价格的波动、需求市场的波动等不确定因素造成风险的有效手段。本模型基于 Anylogic 实现了对三阶供应链模型建模,分析了工厂、分销商、零售商不同的订货、配送模式对各级库存和订单的完成率及完成时间的影响。通过比较各阶供应链的库存水平、提前期等数据得出结论。

1.2 理论背景

a. 定期订货法和定期订货法

两者的基本区别是:定量订货模型是“事件驱动”,而定期订货模型是“时间驱动”。在定量订货模型中,当库存量到达规定的再订货水平后,才引发订货行为。这一事件有可能随时发生,主要取决于对该物资的需求情况。与之相对的是,定期订货模型只限于在盘点期末进行订货,模型中唯一的驱动原因是时间的变化。运用定量订货模型时,必须连续监控剩余库存量当库存量降低到预先设定的再订购点 R 时,就进行订货;且它要求每次从库存里取出货物或者往库存里增添货物时,必须刷新记录以确认是否已达到再订购点。而定期订货模型中,库存盘点只在盘点期发生。其操作方式是按照规定时间检查库存量,并随即提出订购,补充至所规定的数量。

定量订货模型与定期订货模型的比较

特征	定量订货模型	定期订货模型
订购量	Q 是固定的(每次订购量相同)	Q 是变化的(每次的订购量不同)
订货点	固定	可变
何时订购	当库存量降低到订货点 R 时发出订购请求	订购的间隔是固定的,每隔一个固定的间隔期 T ,就可发出订购请求,即在盘点期到来时进行订购
检查周期	可变	固定
库存记录	每次出库都记录	只在盘点期记录

图 1.1 两种订货模型的对比

b. 配送模式描述

分销商送货的模式是指由生产企业直接把连锁零售企业所需商品在规定时间内送到各连锁门店的物流活动。许多零售型企业都在全国范围内建立自己的分销体系,这种方式比较适用于销售量比较大的情况,并且适用于整车运输的商品配送。

零售商取货的模式是由零售商自行根据需求从上游分销商处取货的方式,能够更好地满

足企业配送业务上的时间、空间要求，特别是配送要求频繁的企业，这种模式更快速、灵活地满足客户需求。

1.3 贡献、发现和结论

本次建模的系统分为两大模块，分别是分销商与上游工厂的供应链模块和分销商与下游零售商的供应链模块。前者重点讨论定期订货模型和定量订货模型的特点及区别，和其对分销商库存的影响。后者重点对两种供货模式对库存及订单的完成率的影响进行研究，即分销商根据零售商产生需求送货和零售商根据需求自主取货两种模式。在下面的建模中，假定工厂的供应是充足的，不存在缺货的情况，因为工厂处于本供应链结构的最上游。

通过基于 AnyLogic 的供应链建模，比较各阶供应链的库存水平可得出以下结论：

- (1) 各阶供应链的库存水平波动程度逐级扩大。
- (2) 定期订货模型平均库存较大，因为要预防在盘点期和订货提前期内发生缺货情况；相反定量订货模型的库存量相对较小，因其只要在订货的提前期内预防缺货即可。
- (3) 零售商自主取货的模式下需求发送时间和货物到达时间差较小，即提前期较短，相比起供应商供货的模式可以更加快速、灵活地满足需求。
- (4) 同样的参数条件下，分销商配送产生的因超出库存容量而被一起的库存量几乎为 0，而零售商取货模式下该项数值较大，分销商处产生了很多废弃库存。

1.4 理论探讨及应用

(1) 因为平均库存量较低，所以定量订货模型有利于贵重物资的库存。对于重要的物资如关键维修零件，定量订货模型将更适合，因为该模型对库存的监控更密切，这样可以对潜在的缺货更快地做出反应。

(2) 维持定量订货模型需要更多的时间，因为每一次补充库存或货物出库都要进行记录。

(3) 在缺货时分销商不能一味根据下游的订货量来决定配给量，以此才能杜绝下游企业夸大订货量来获得较多配给量的现象。因为很多供应商允许下游企业进行退货，如果配给过多，后期批发商处可能出现大量的超出仓库容量而被遗弃的库存。

(4) 供应商之间以及分销商、零售商应该共享供应能力、库存状况信息，让供应商了解到更加准确的需求信息，根据需求信息设计合理的生产计划，进而合理安排生产进度，达到联合库存管理的目的，防止下游企业进行博弈行为。

二、模型介绍

2.1 假设条件

- (1) 初始设置需求随机产生，满足正态分布。
- (2) 初始设置三处分销商，每处分销商分别为三家连锁零售商供货。
- (3) 工厂的供应是无限的，不会分销商出现订单无法完成的情况。

2.2 概念模型

三阶供应链概念模型分为：工厂-分销商-零售商。

消费者总有需求新产品的趋势，这样便产生了需求。该需求会反馈给零售商，零售商为了满足顾客的需求会做出相对应的动作：向上游下订单。而分销商接收到零售商的订单会通过物流方式向零售商配送货物，或者也可以选择零售商自主取货。一般情况下，分销商只有产成品库存，可以选择在库存不足的时候或指定的盘点期向上游工厂进行订货。

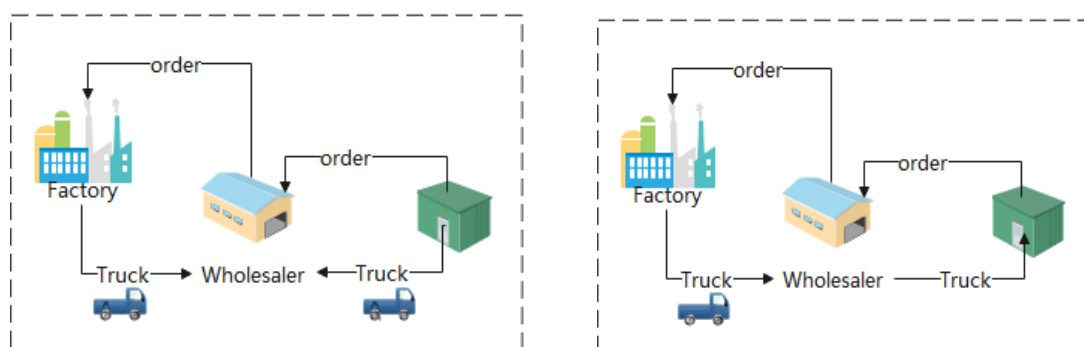


图 2.1 三阶供应链的仿真概念模型

2.3 模型功能布局

a. 数据库

本模型为对部分智能体进行定位，建立了 2 个数据表，分别记录分销商、零售商相关信息。具体参数如下：

◆ Wholesaler

	name	longitude	latitude
	▼	▼	▼
1	Nanjing	118.78	32.04
2	Hangzhou	120.21	30.21
3	Shanghai	121.48	31.22

◆ Retailer

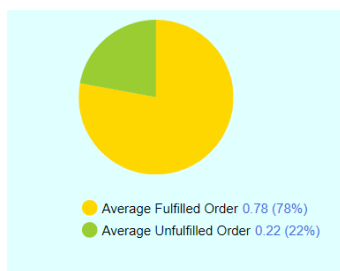
	wholesaler	name	longitude	latitude
1	Nanjing	Maanshan	118.84	31.56
2	Nanjing	Zhenjiang	119.43	32.13
3	Nanjing	Taizhou	119.88	32.32
4	Hangzhou	Shaoxing	120.49	30.08
5	Hangzhou	Ningbo	121.84	29.9
6	Hangzhou	Zhoushan	122.1	30.02
7	Shanghai	Suzhou	120.63	31.3
8	Shanghai	Jiaxing	120.78	30.75
9	Shanghai	Changzhou	119.6	31.72

b. 图表

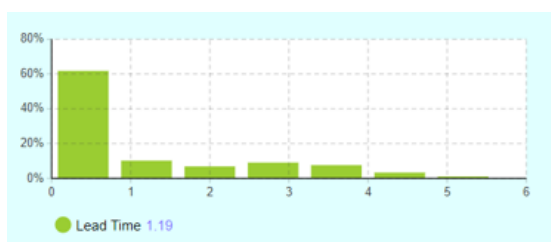
模型中通过一些统计图表反映库存、提前期及订单的完成情况等。

◆ Main

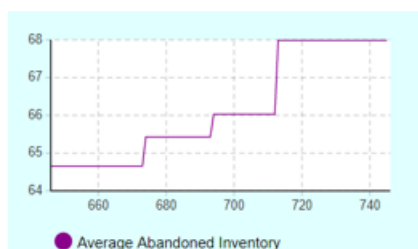
Main 窗体中存放的图标反映的是几个分销商/零售商的平均数据。



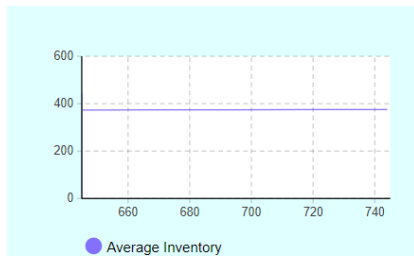
反映平均满足和未满足的零售商订单的比例



反映零售商提前期的均值

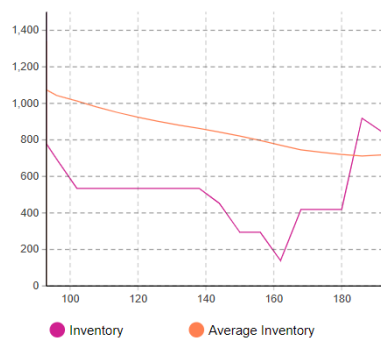


反映分销商因超出仓库容量而产生的被遗弃的库存的均值

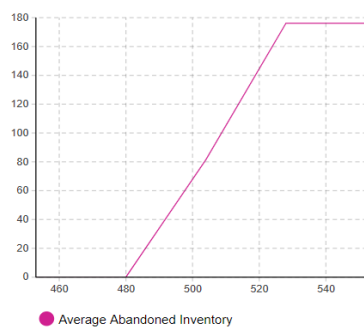


反映分销商的平均库存

◆ Wholesaler



反映某分销商的库存变化和平均库存

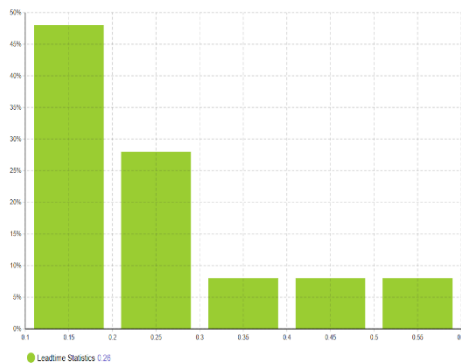


反映某分销商因超出仓库容量而产生的被遗弃的库存量变化



反映某分销商满足与未满足订单的比例

◆ Retailer



反映某零售商需求发送时间
和货物到达时间差

c. 智能体

本模型一共包含六个智能体，各智能体功能如下：

◆ Main

存放接口的连接；启动时对分销商与零售商进行初始化，将其在地图上定位；通过 GIS 地图和 GIS 点反映整个供应链的可视化过程。

◆ Factory

源头工厂仿真。向卡车发送信息提供运输目的地和数量的信息。

◆ Wholesaler

分销商仿真。处理来自下游零售商的订单、向上游工厂发送订单、存放反映库存和订单状态的图表。本模型共设置三处分销商。

◆ Retailer

零售商仿真。根据需求产生订单、自主取货模式下可向卡车发送信息前往分销商处取货。存放反映提前期变化的图表。本模型共设置九处零售商，三处零售商与一处分销商对应。

◆ Truck

运输工具仿真。规定卡车的运输逻辑。

◆ AnalysisCharts

用于存放反映各分销商、零售商平均数据的图表，通过接口接收数据信息。

d. 接口

在本模型中，Port 在消息传递机制中起着核心作用。消息通过 Port 发送和接收。Port 是双向的，可以用于输入和输出。接口的具体连接及代码将在建模部分详细介绍。

e. Simulation

工厂		零售商	
经度	<input type="text"/>	库存容量	<input type="text"/>
纬度	<input type="text"/>	取货模式	<input checked="" type="radio"/> 零售商取货 <input type="radio"/> 分销商送货
卡车容量	<input type="text"/>	需求均值	<input type="text"/>
		需求方差	<input type="text"/>
		卡车容量	<input type="text"/>

分销商	
库存容量	<input type="text"/>
订货模式	<input checked="" type="radio"/> 定期订货 <input type="radio"/> 定量订货
	定期订货 订货期 <input type="text"/> 定量订货 安全库存 <input type="text"/>
卡车容量	<input type="text"/>

对参数进行模拟设置。可以设置工厂位置、卡车容量、库存、需求等参数，选择取货模式和订货模式。设置选项界面如右图

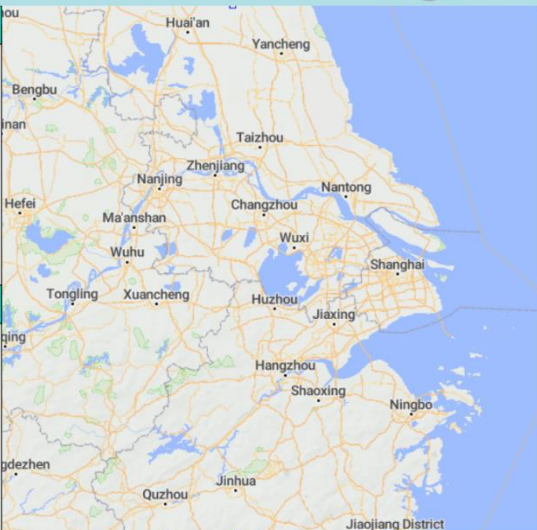
2.4 模型界面布局

a. 初始参数设置界面

SupplyChainModel

工厂		零售商	
经度	<input type="text" value="120.29"/>	库存容量	<input type="text" value="1000.0"/>
纬度	<input type="text" value="31.59"/>	取货模式	<input type="radio"/> 零售商取货 <input checked="" type="radio"/> 分销商送货
卡车容量	<input type="text" value="500.0"/>	需求均值	<input type="text" value="20.0"/>
		需求方差	<input type="text" value="3.0"/>
		卡车容量	<input type="text" value="100.0"/>

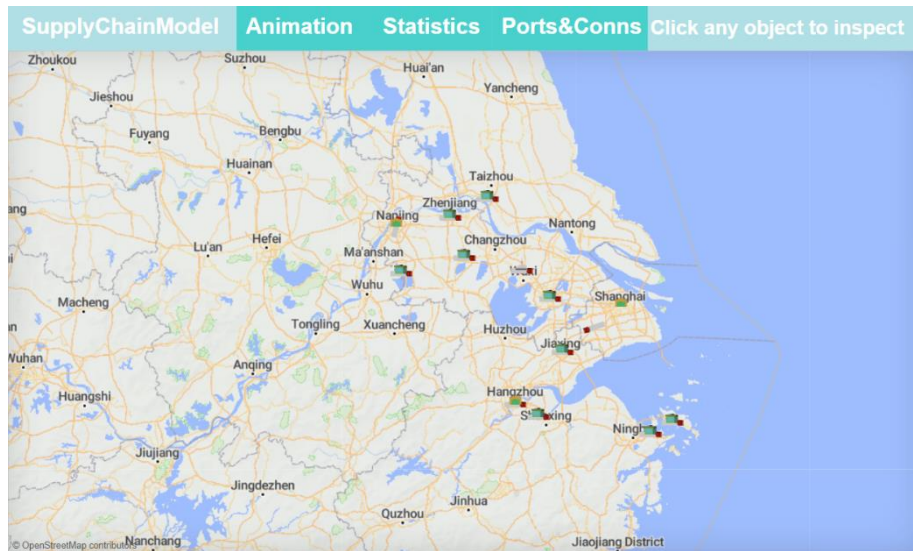
分销商	
库存容量	<input type="text" value="1000.0"/>
订货模式	<input type="radio"/> 定期订货 <input checked="" type="radio"/> 定量订货
	定期订货 订货期 <input type="text" value="14.0"/> 定量订货 安全库存 <input type="text" value="20.0"/>
卡车容量	<input type="text" value="200.0"/>



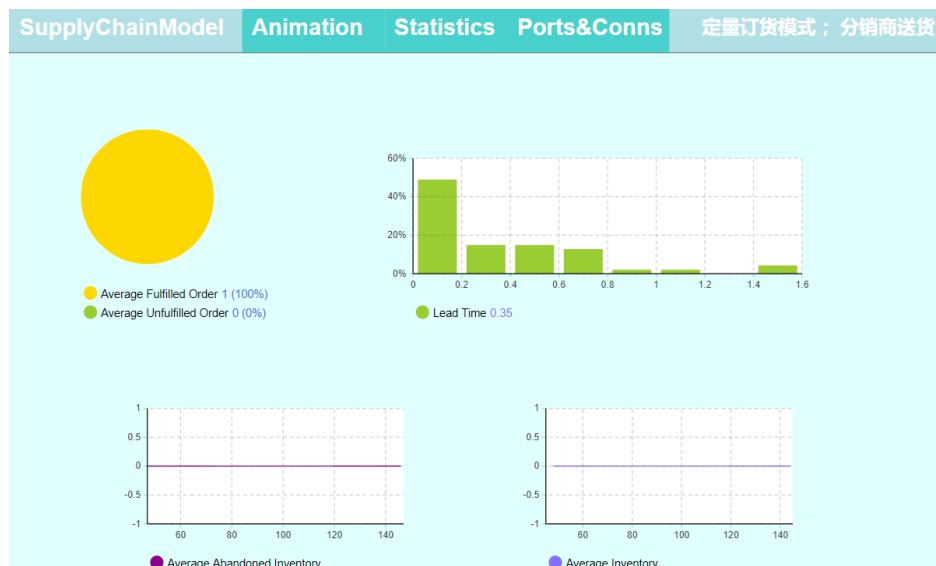
b. 运行界面

◆ 可视化分区及功能:

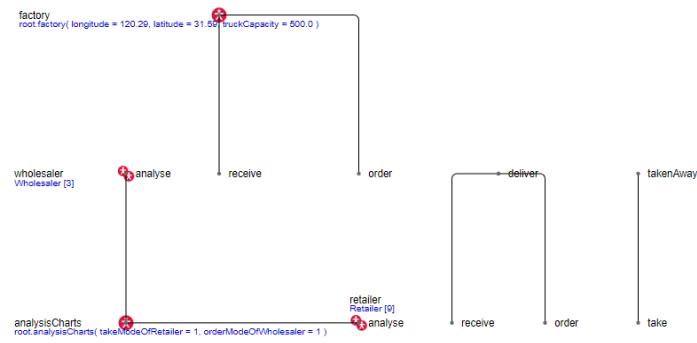
Animation: 通过 map 反映供应链模型各层次智能体的位置和配送过程, 可以通过点击图标进入每个智能体的可视化界面, 查看其相关图标和参数状态。



Statistics: 图表界面, 查看反映平均数据的图表格。



Ports&Conns: 端口连接可视化界面。



◆ 模型的可视化方案

采取分区的可视化设计，不同功能的界面之间相互独立又可以方便跳转，使各部分界面的界限更加清晰明确，模式的可视化更强，便于数据的提取和观察。

三、建模

3.1 模块化方案

Port 在本模型的消息传递机制中起着核心作用。智能体模块之间仅能通过端口进行互动，以主动或者响应对方请求的方式提供信息与服务，以提升模块的可复用能力。只要正确实现端口的对接或拆除，就可以实现对智能体模块的功能的调用或解除。下面将详细介绍本模型中的端口设计及连接方式。

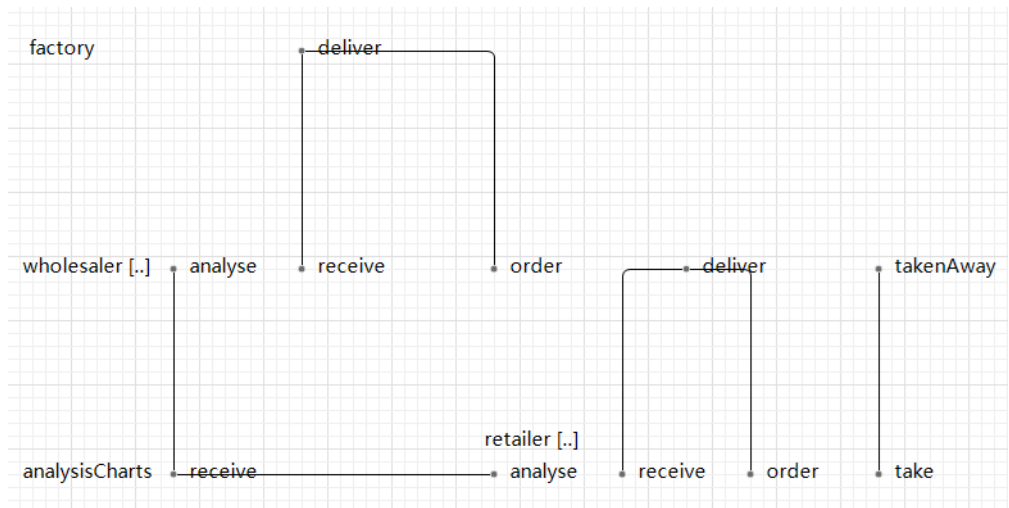


图 3.1 端口连接方式

a. Factory

• deliver - Port

Name:

☒ Show name ☐ Ignore ☒ Visible on upper agent

Visible: ☒ yes

▼ Message handling actions

On receive:

```
// 收到来自分销商的订单之后就让自己的卡车运送  
truck.deliver(msg.getFirst(), msg.getSecond());
```

On send:

“deliver”端口与 Wholesaler 中的“receive”和“order”连接，作用是接收运送的目的地和数量信息，并传递给传给卡车。

11

b. Wholesaler

order - Port

Name: ☒ Show name ☐ Ignore ☒ Visible on upper agent

Visible: ☒ yes

Message handling actions

On receive:

On send:

```
// 记录想工厂发送订单的时间
/* 只有在当前的订单完成后才能进行下一个订单的发生，
   因此这个变量中的数据只有在用完之后才会被更新，不会有被挤掉的可能性*/
sendTime = time();
```

“order” 端口与 Factory 中的 “deliver” 连接，作用是向工厂下单，并记录时间，便于统计提前期。

receive - Port

Name: ☒ Show name ☐ Ignore ☒ Visible on upper agent

Visible: ☒ yes

Message handling actions

On receive:

```
if (this == msg.getFirst()) // 如果工厂发送的货物是给自己的话
{
    // 上一个订单已经收到了，如果想要继续向工厂发送订单的话是可以的
    isLastOrderReceived = true;
    currentInventory += msg.getSecond(); // 库存增加

    // 如果货物到达后仓库容量不够，记录被丢弃的货物数量
    // 更新统计量avgAbandonedInventory，将库存更正为仓库容量
    if (currentInventory > capacity)
    {
        numOfAbandonedInventory += (currentInventory - capacity);
        // 将数据发送给“分析”智能体
        analyse.send(new Pair<String, Double>("numOfAbandonedInventory", 1.0*(currentInventory - capacity)));
        avgAbandonedInventory.update();
        currentInventory = capacity;
    }
    // 记录货物收到时间，计算花费时间，更新统计量avgLeadTime
    receiveTime = time();
    leadTime = (receiveTime - sendTime) / 24;
    avgLeadTime.update();

    // 到货后就需要处理订单列表中的订单
    // 如果订单列表非空，并且仓库容量大于订单列表中第一个订单的需求量
    while (!orderCollection.isEmpty() && currentInventory >= orderCollection.getFirst().getSecond())
    {
        // 库存量减少相应数量，并将货物发送给retailer，移出这一订单
        double sendAmount = orderCollection.getFirst().getSecond();
        currentInventory -= sendAmount;
        truck.deliver(orderCollection.getFirst().getFirst(), sendAmount);
        orderCollection.removeFirst();
    }
    // 如果订单列表非空，并且所剩到量不为0（但不足以完全满足订单列表中第一个订单的需求量）
    // 则所剩的到量先满足订单的一部分需求，并更新这一订单的需求，而且此订单还是排在最前面
    if (currentInventory != 0)
    {
        if (!orderCollection.isEmpty())
        {
            orderCollection.set(0, new Pair(
                orderCollection.getFirst().getFirst(),
                orderCollection.getFirst().getSecond() - currentInventory));
            currentInventory = 0;
        }
    }
    if (orderMode == 1) //如果是定量订货模型，则检查是否需要订货，如果需要则向工厂发送订单
        checkInventory();
}
```

“receive” 端口与 Factory 中的 “deliver” 连接。收到来自工厂的货物后，首先检查是否爆仓，更新相关统计量；其次依次满足订单集合中的订单；最后如果是定量订货模型则判断是否要再次向工厂下单。

• deliver - Port

Name: ☒ Show name ☐ Ignore ☒ Visible on upper agent

Visible: ☒ yes

▼ Message handling actions

```
On receive: // 针对批发商送货模式
// 库存不够将需求增加到订单列表，到货之后补上；但未满足订单增加，更新统计量
if (msg.getFirst().wholesaler == this.name) // 如果订单是由自己的下游零售商发出的话
{
    numOfOrderOfRetailer ++;
    // 将数据发送给“分析”智能体
    analyse.send(new Pair("numOfOrderOfRetailer", 1.0));
    allDemand += msg.getSecond();

    if (currentInventory - msg.getSecond() < 0) // 库存不够时
    {
        double amount = msg.getSecond();
        // 如果还有库存量，利用自己的库存量尽可能满足订单需求
        if (currentInventory > 0)
        {
            truck.deliver(msg.getFirst(), currentInventory);
            currentInventory = 0;
            amount -= currentInventory;
        }
        // 未满足订单数增加，并更新统计数据；将还未满足的订单添加到订单列表中
        numOfUnfulfilledOrder ++;
        // 将数据发送给“分析”智能体
        analyse.send(new Pair("numOfUnfulfilledOrder", 1.0));
        orderCollection.addLast(new Pair(
            msg.getFirst(), amount));
    }
    else // 库存足够时，直接发信息给自己的卡车送货，库存减少相应数量
    {
        truck.deliver(msg.getFirst(), msg.getSecond());
        currentInventory -= msg.getSecond();
    }

    if (orderMode == 1) // 如果是定量订货模型，则检查是否需要订货，如果需要则向工厂发送订单
        checkInventory();
}
```

“deliver”端口与 Retailer 中的“receive”和“order”连接，实现在分销商送货模式下 Retailer 和 Wholesaler 之间的信息传递。作用是在分销商送货模式下收到零售商订单需求后，根据仓库库存进行配送。若库存不能满足全部订单，则先利用现有库存满足部分订单需求，如果是定期订货模式，向工厂发送订单。同时记录库存和未满足订单的数量变化。

• takenAway - Port

Name: ☒ Show name ☐ Ignore ☒ Visible on upper agent
Visible: ☒ yes

▼ Message handling actions

```
On receive: // 针对零售商取货模式，如果库存不够，未满足订单增加
if (msg.getFirst().wholesaler == this.name) // 如果订单是由自己的下游零售商发出的话
{
    // 分销商订单计数增加，总需求量增加
    numOfOrderOfRetailer ++;
    // 将数据发送给“分析”智能体
    analyse.send(new Pair("numOfOrderOfRetailer", 1.0));
    allDemand += msg.getSecond();
    double amount = msg.getSecond();

    if (amount < currentInventory) // 如果库存足够
    {
        // 向零售商发信息让他来取订单量相应需求，自身库存减少相应数量
        self.send(new Pair(this, amount));
        currentInventory -= amount;
    }
    else // 如果库存不够
    {
        // 向零售商发信息让他来取已有的库存量，库存设置为0，未满足订单数增加
        self.send(new Pair(this, currentInventory));
        currentInventory = 0;
        // 将数据发送给“分析”智能体
        analyse.send(new Pair("numOfUnfulfilledOrder", 1.0));
    }
    if (orderMode == 1) // 如果是定量订货模型，则检查是否需要订货，如果需要则向工厂发送订单
        checkInventory();
}
```

“takenAway”端口与 Retailer 中的“take”连接。实现在零售商取货模式下 Retailer 和 Wholesaler 之间的信息传递。分销商通过端口接收到自己下游的零售商的订单请求后，检查自身库存后将取货的信息通过端口返回给零售商通知其自行取货。如果库存量不足以满足订单，则先运走一部分，并更新未完成订单数量。

c. Retailer

• order - Port

Name: ☒ Show name ☐ Ignore ☒ Visible on upper agent
Visible: ☒ yes

“order”端口与 Wholesaler 中的“deliver”连接，实现在分销商送货模式下 Retailer 和 Wholesaler 之间的信息传递。作用是向分销商发送需求地点以及需求数量信息。

• take - Port

Name: ☒ Show name ☐ Ignore ☒ Visible on upper agent

Visible: ☐ yes ☒ ves

▼ Message handling actions

On receive:

```
if (msg.getFirst().name == wholesaler) // 如果接收到的信息是分销商发来的
{
    if (msg.getSecond() > 0) // 如果订单量非0（订单量为0的情况：分销商在没有库存的情况下发送的）
        // 调用卡车智能体的deliver函数，让自己的卡车去取货，告诉卡车目的地和数量
        truck.deliver(msg.getFirst(), msg.getSecond());
}
```

“take”端口与 Retailer 中的“take”连接。实现在零售商取货模式下 Retailer 和 Wholesaler 之间的信息传递。在这种模式下，零售商首先通过该接口询问分销商是否可以前往取货，得到的反馈传回该接口。随后零售商派卡车前往分销商处取货。

• receive - Port

Name: ☒ Show name ☐ Ignore ☒ Visible on upper agent

Visible: ☐ yes ☒ ves

▼ Message handling actions

On receive:

```
if (this == msg.getFirst()) // 如果接收到的信息是给自己的
{
    double amount = msg.getSecond();
    /*到达的货物优先满足订单列表中的需求
    如果订单非空，并且到货量（或所剩的到货量）大于订单列表中第一个订单的需求
    则到货量先用于满足订单列表*/
    while (!orderCollection.isEmpty() && amount >= orderCollection.getFirst().getFirst())
    {
        amount -= orderCollection.getFirst().getFirst();
        leadTime = (time() - orderCollection.getFirst().getSecond()) / 24;
        statisticsLeadTime.update();
        histogramLeadTime.update();
        analyse.send(new Pair<String, Double>("histogramLeadTime", 1.0*leadTime));
        orderCollection.removeFirst();
    }
    /*如果订单列表非空，并且所剩到货量不为0（但不足以完全满足订单列表中第一个订单的需求量）
    则所剩的到货量先满足订单的一部分需求，并更新这一订单的需求，而且此订单还是排在最前面*/
    if (!orderCollection.isEmpty() && amount != 0)
    {
        orderCollection.set(0, new Pair<Double, Double>(
            orderCollection.getFirst().getFirst() - amount,
            orderCollection.getFirst().getSecond()));
    }
}
```


“receive”分别与 Wholesaler 和 Retailer 中的“analyse”端口连接，接受来自零售商和分销商在模型运行过程中记录的相关统计量数据，用于数据分析。

d. AnalysisCharts

• receive - Port

Name: ☒ Show name ☐ Ignore ☒ Visible on upper agent
Visible: ☒ yes

▼ Message handling actions

On receive:  // 统计量的更新

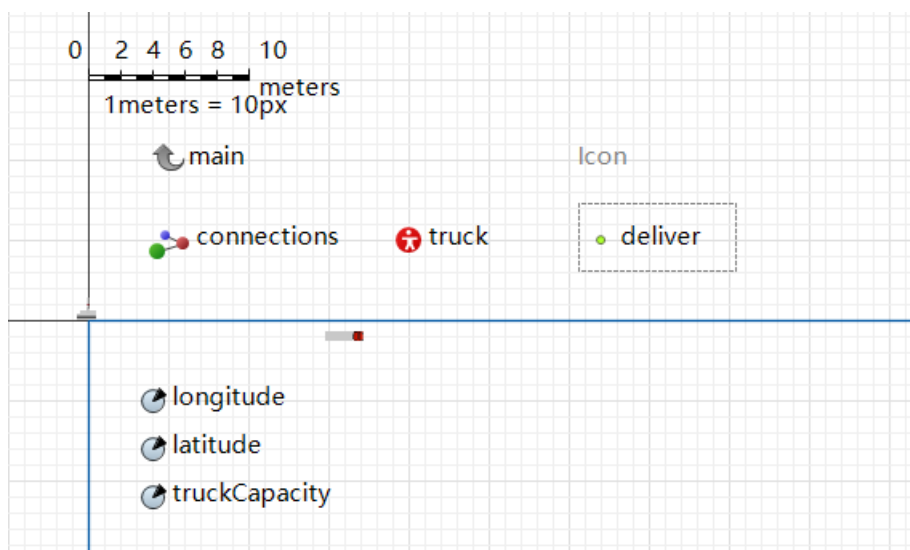
```
if (msg.getFirst() == "numOfOrderOfRetailer")
    numOfOrderOfRetailer += msg.getSecond();
else if (msg.getFirst() == "numOfUnfulfilledOrder")
    numOfUnfulfilledOrder += msg.getSecond();
else if (msg.getFirst() == "histogramLeadTime")
    histogramLeadTime.add(msg.getSecond());
else if (msg.getFirst() == "numOfOrderToFactory")
    numOfOrderToFactory += msg.getSecond();
else if (msg.getFirst() == "numOfAbandonedInventory")
{
    numOfAbandonedInventory += msg.getSecond();
    avgAbandonedInventory.update();
}
else if (msg.getFirst() == "avgInventory")
{
    avgInventory.add(msg.getSecond());
    avgInventory.update();
}
```

“receive”分别与 Wholesaler 和 Retailer 中的“analyse”端口连接，接受来自零售商和分销商的在模型运行过程中记录的相关统计量数据，用于数据分析。

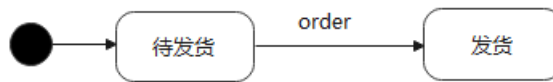
3.2 智能体设计

◆ Factory

智能体界面：

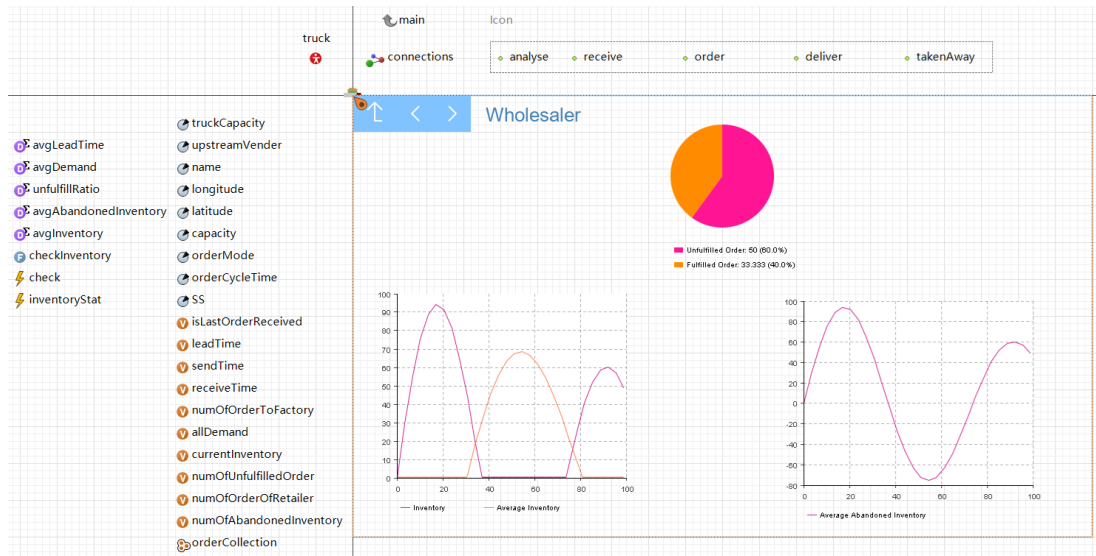


状态图:



◆ Wholesaler

智能体界面:



状态图:

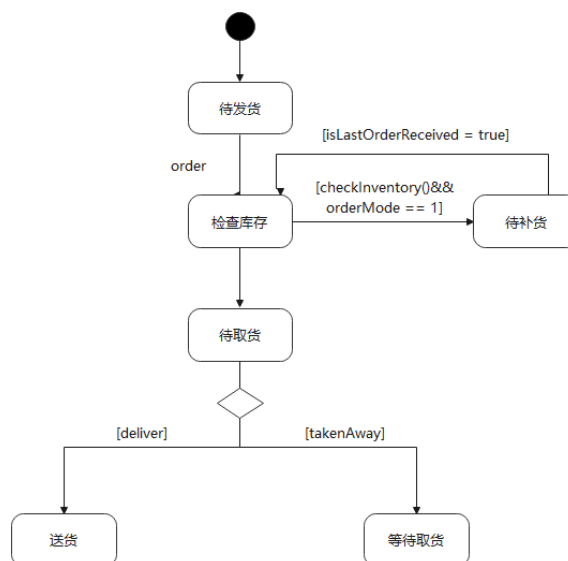


图 3.2 定量订货的订货模式下 Wholesaler 的状态图

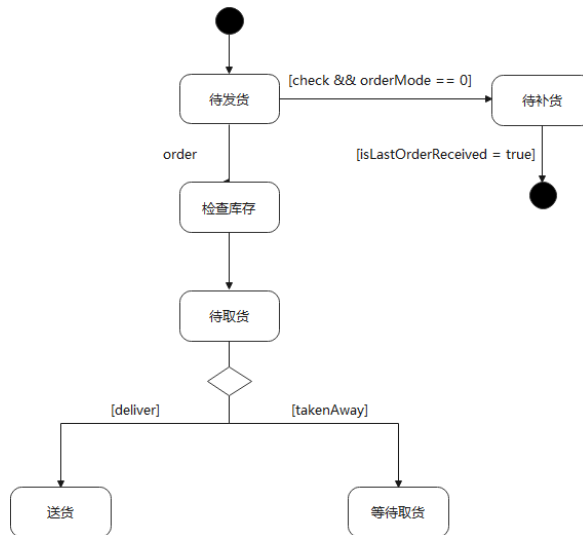


图 3.2 定期订货的订货模式下 Wholesaler 的状态图

部分函数及事件：

checkInventory - Function

Name: ☒ Show name ☐ Ignore

Visible: ☒ yes

☒ Just action (returns nothing)

☐ Returns value

Arguments

Function body

```

// 针对定量订货模型
// 如果库存小于安全库存+提前期*平均需求，且上一个给工厂
// 向工厂发送订单，订单量为分销商容量
if (currentInventory <= avgLeadTime.mean()*avgDemand.
{
    if (isLastOrderReceived == true)
    {
        order.send(new Pair(this, capacity));
        numOfOrderToFactory ++;
        analyse.send(new Pair("numOfOrderToFactory",
        isLastOrderReceived = false;
    }
}

```

checkInventory 函数用于在定量订货模型下检查分销商库存。

⚡

check - Event

Name:

check

☒ Show name ☐ Ignore

Visible:

☒ yes

Trigger type:

Timeout

Mode:

Cyclic

☒ Use model time ☐ Use calendar dates

First occurrence time (absolute):

orderCycleTime

days

Occurrence date:

2021/ 6/10

08:00:00

Recurrence time:

orderCycleTime

days

☒ Log to database

[Turn on model execution logging](#)

▼ Action

// 定期订货模式中，每循环周期向工厂发信息订货

if (orderMode == 0)

{

order.send(new Pair(this, avgLeadTime.mean() * avgDemand.mean() + capacity - currentInventory));

numOfOrderToFactory ++;

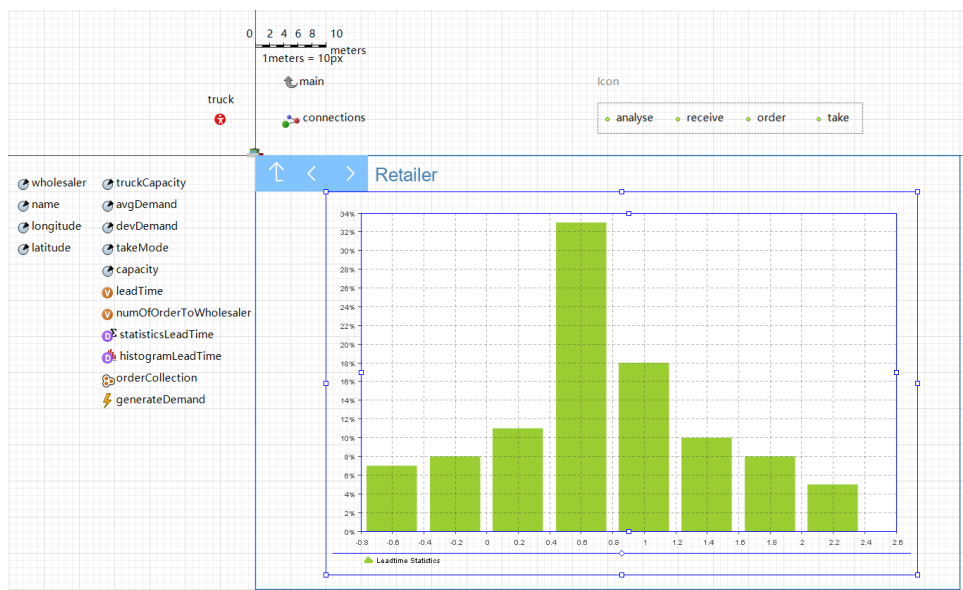
analyse.send(new Pair("numOfOrderToFactory", 1.0));

}

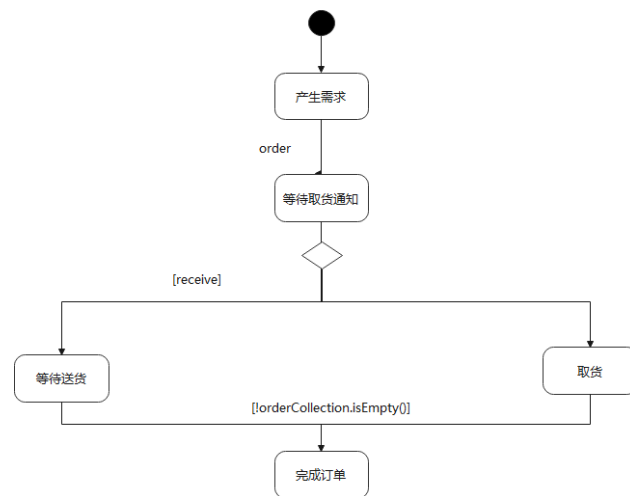
check 事件用于在定期订货模型下在每个盘点期触发向工厂下单。

◆ Retailer

智能体界面：



状态图：



部分函数及事件：

generateDemand - Event

Name: ☒ Show name ☐ Ignore

Visible: ☒ yes

Trigger type:

Rate:

☒ Log to database
[Turn on model execution logging](#)

Action

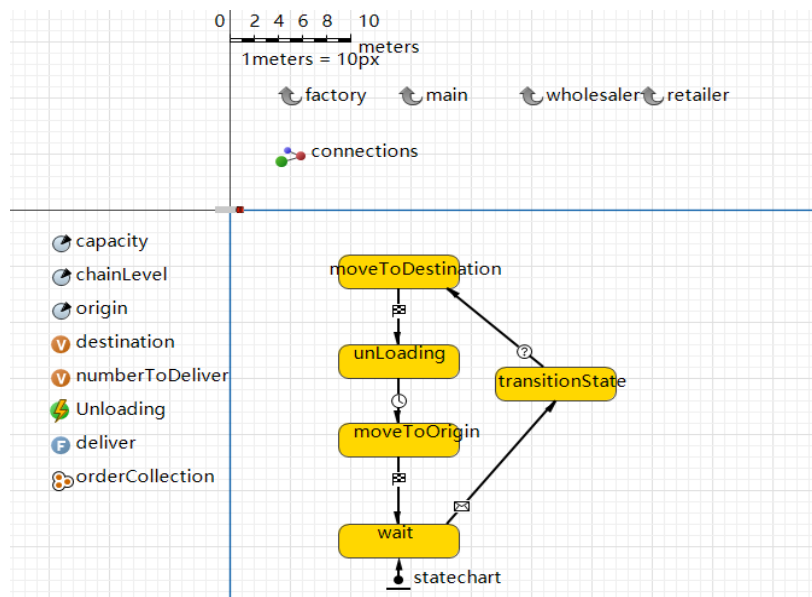
```
double orderNum = normal(devDemand, avgDemand); // 需求量服从正态分布，随机产生
numOfOrderToWholesaler ++;
orderCollection.addLast(new Pair<Double, Double>(orderNum, time()));

if (takeMode == 1) // 分销商送货模式
{
    order.send(new Pair<this, orderNum>());
}
if (takeMode == 0) // 零售商取货模式
{
    take.send(new Pair<this, orderNum>());
}
```

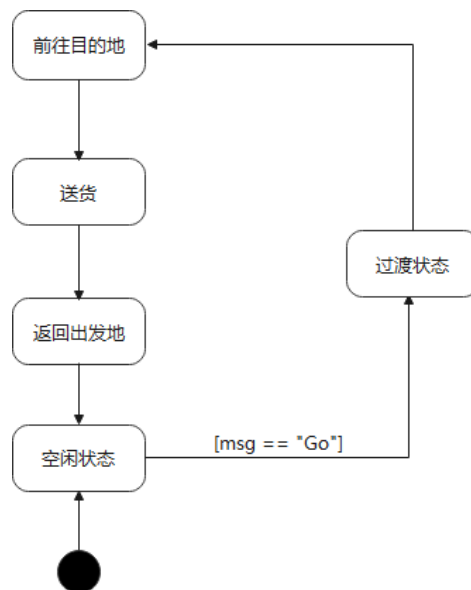
generateDemand 事件的作用是每天随机产生需求，将其加入零售商的需求集合，并在不同模式下通过不同的端口向分销商发送请求。

◆ Truck

智能体界面：



状态图:



部分参数、函数体及事件:

🔄
chainLevel - Parameter

Name:

Visible: ☒ yes

Type:

☒ Show name ☐ Ignore

chainLevel 参数用于区别卡车属于供应链中的哪一个层次：

- 1: Factory
- 2: Wholesaler
- 3: Retailer

在 unloading 状态卡车要向所属的供应链层次发送消息。

deliver - Function

Name: ☒ Show name ☐ Ignore

Visible: ☒ yes

☒ Just action (returns nothing)

☐ Returns value

Arguments

Function body

```
// 如果订单量大于卡车容量，则对订单进行拆分
while (number > capacity)
{
    orderCollection.addLast(
        new Pair<Agent, Double>(dest, capacity));
    number -= capacity;
}
if (number != 0)
{
    orderCollection.addLast(
        new Pair<Agent, Double>(dest, number));
}
// 让卡车开始工作
send("Go", this);
```

卡车的送货函数。卡车收到送货命令后，决定如何送货。如果订单的量大于卡车容量则对订单进行拆分。每次送货的信息存放在订单集合中。

3.3 智能体交互

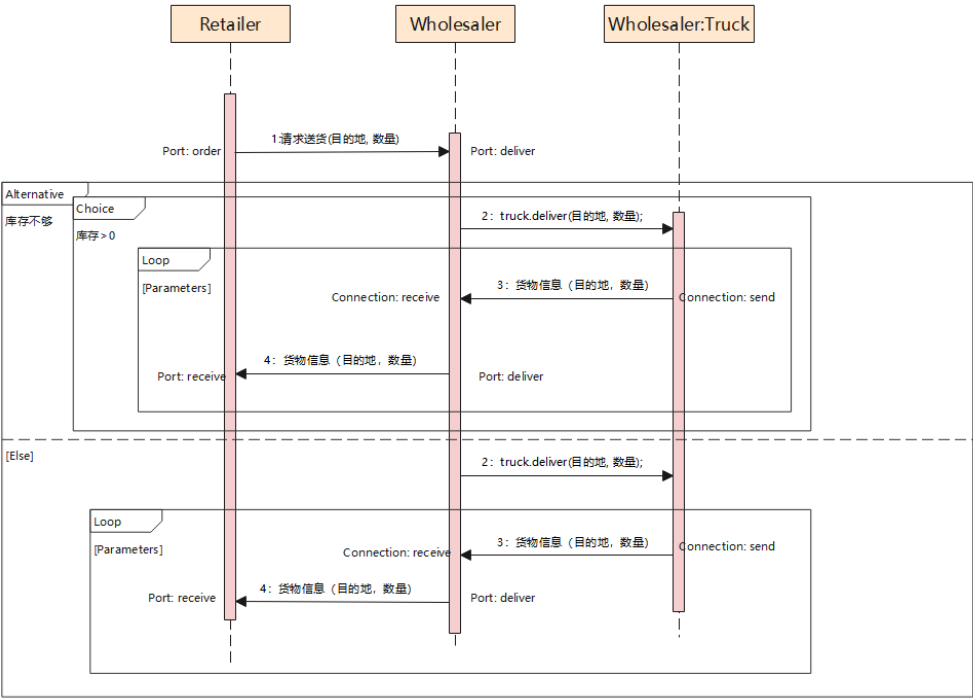


图 3.3 分销商送货模式时序图

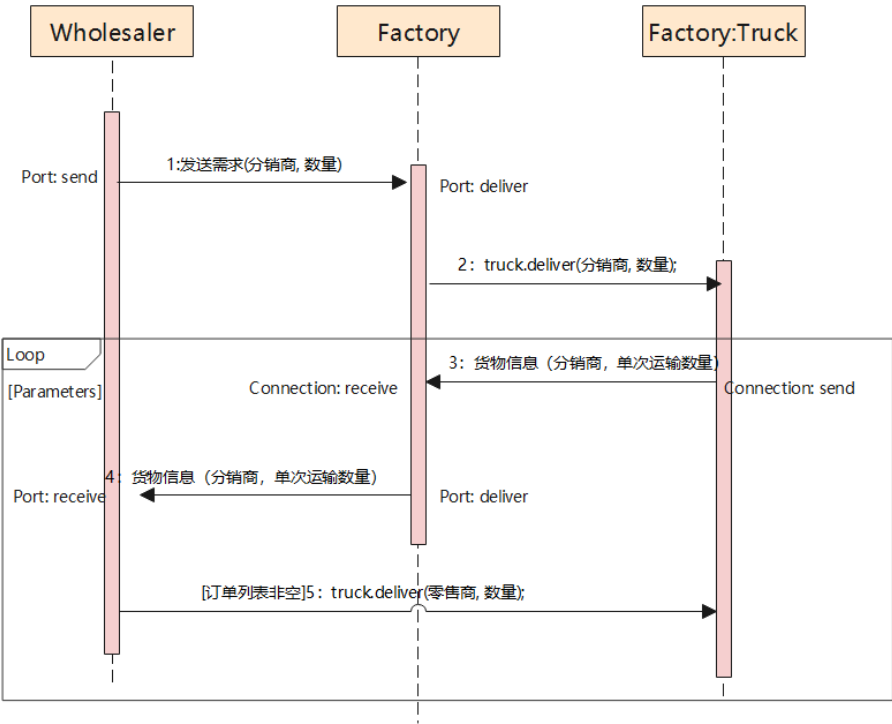


图 3.3 分销商订货模式时序图

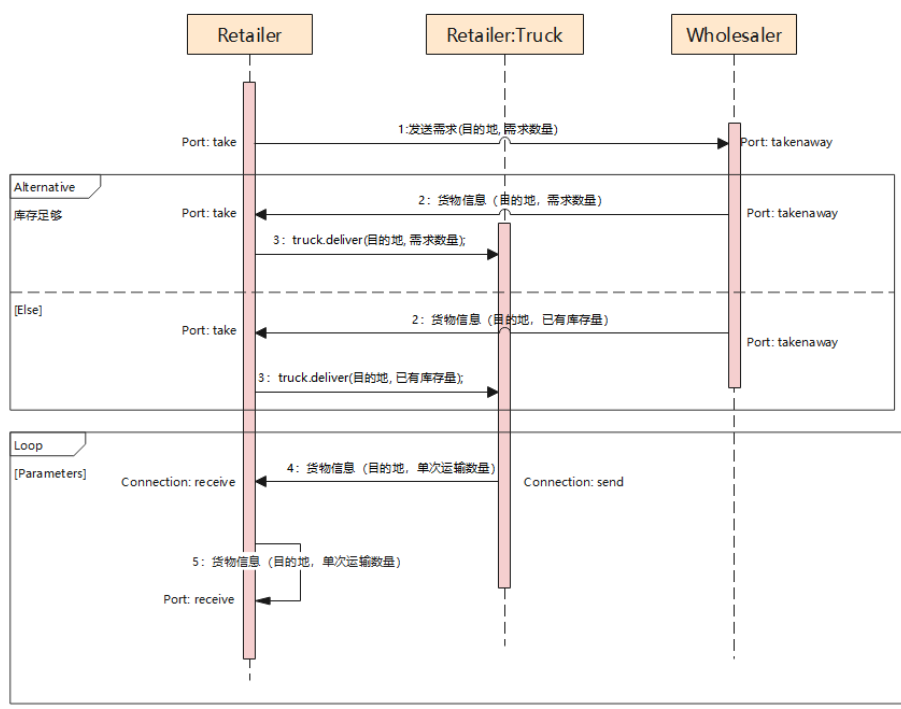


图 3.5 零售商自主取货模式时序图

3.4 输入与输出

◆ 输入：

Wholesaler、Retailer 对应关系及地址初始数据来源：数据库

不同订货、取货模式的选择及初始参数设置：在 Simulation 界面设置后在 Main 窗体中进行初始化。

getTakeModeOfRetailer - Function

Name: ☒ Show name ☐ Ignore

Visible: ☒ yes

☐ Just action (returns nothing)

☒ Returns value

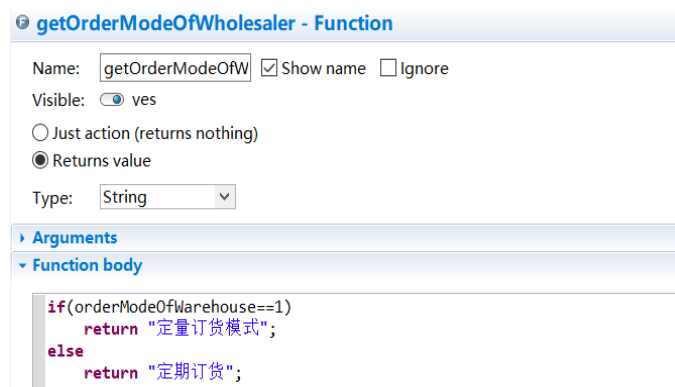
Type:

Arguments

Function body

```

if(takeModeOfRetailer==0)
    return "    ; 零售商取货";
else
    return "    ; 分销商送货";
  
```



- ◆ 输出：通过反应库存、提前期、订单完成率等统计量的图表输出实验结果，反映不同供应链模型下相关统计量的变化情况。

四、课程反馈

4.1 组内分工

- ◆ 前期讨论及模型思路敲定：全体成员
- ◆ 基础模型建立与改进、项目全框架搭建：封钰震
- ◆ 项目完善及可视化制作：竺瑞航
- ◆ 项目完善及注释：龚海瑜
- ◆ 运行示例及项目报告撰写：武丹宁
- ◆ 项目报告时序图绘制：龚海瑜

4.2 反思与总结

在本次课程设计中，我们首先对基础模型的逻辑进行了梳理，构建了两种订货模型和两种配送模型的仿真建模。利用端口实现信息传递，提高了模型的独立性；利用分区化的可视化方案，及图表对相关数据的记录和反馈，使模型更清晰可读。

在厘清建模逻辑、完成了总体框架的搭建后，小组成员也在不断进行复盘，不断发现问题、改进模型。此次设计可能仍有不足之处，如在模型构建初期我们预想的对顾客需求变化进行仿真，加入口碑、广告等影响最后没有在模型中进行体现。如果增加顾客层次可能对库存控制的仿真分析会更加准确，这也是这个模型改进的方向之一。

五、 附件清单

模型文件: SupplyChainModel .alp

数据文件: Wholesaler .xlsx

Retailer .xlsx