CSC 8350 Hands-on Project Report
Car Rental Simulation using DEVSJAVA

Pazhambalacode, Gaayathri Vaidhyanathan
gpazhambalacode1@student.gsu.edu

Alapati, Giridhar
galapati1@student.gsu.edu

Fall 2022

1 Abstract

A car rental system is a service that enables its customers/users to be able to rent a car at their convenience. It is gaining a lot of popularity for the services offered and its affordability. It gives information about the customers booking for the number of days, the quotation based on the time, and the kinds of cars to help customers get a car based on their requirements. The focus of the Hands-on project is to simulate the process of a car rental system using DEVSJAVA. It could be confusing for a car rental service center to manually keep track of the flow in the center and understand the working initially. Hence, a simulation of a car rental system will help us understand and grasp this process flow better.

2 Problem Statement and Background

The Discrete Event System Specification (DEVS) provides a way to define a system in the form of a mathematical object. Using time, inputs, states, functions, and outputs, the next state of the system, along with its functionality and outputs, is determined.

A car rental system is a service that enables its customers/users to be able to rent a car at their convenience. It is gaining a lot of popularity for the services offered and its affordability, which can be observed in Figure 1. With this gaining attention and the necessity for car rentals, it is vital for someone starting this business to have a complete grasp on the working and require a thorough analysis of the demands and functioning of the car rental center.

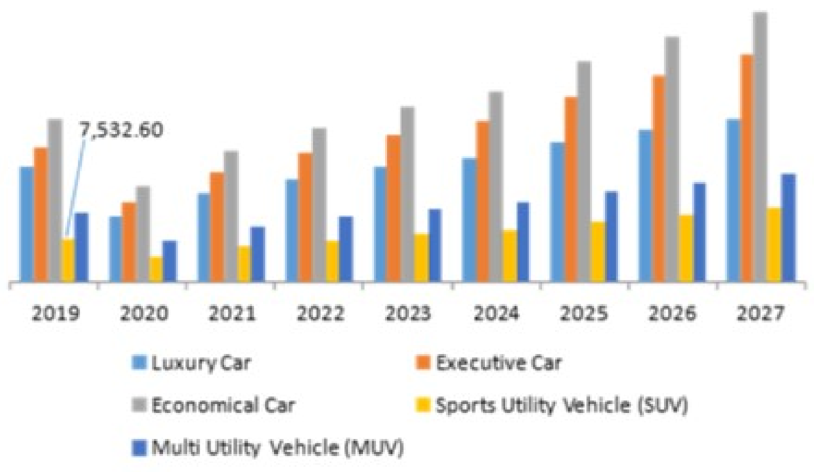


Figure 1: Trend in car rental services over time

The focus of the Hands-on project is to simulate the process of a car rental system using DEVSJAVA. Based on the type of car the customer desires and the number of days the car is to be rented, we run a simulation demonstrating the working of a rental service center upon the arrival of the customer. This also calculates the price details based on the details provided by the customer. There are various stages in a car rental service. This simulation not only helps understand the flow of a car rental service center better but also helps analyze the working of the business to maximize profit and smoother working.

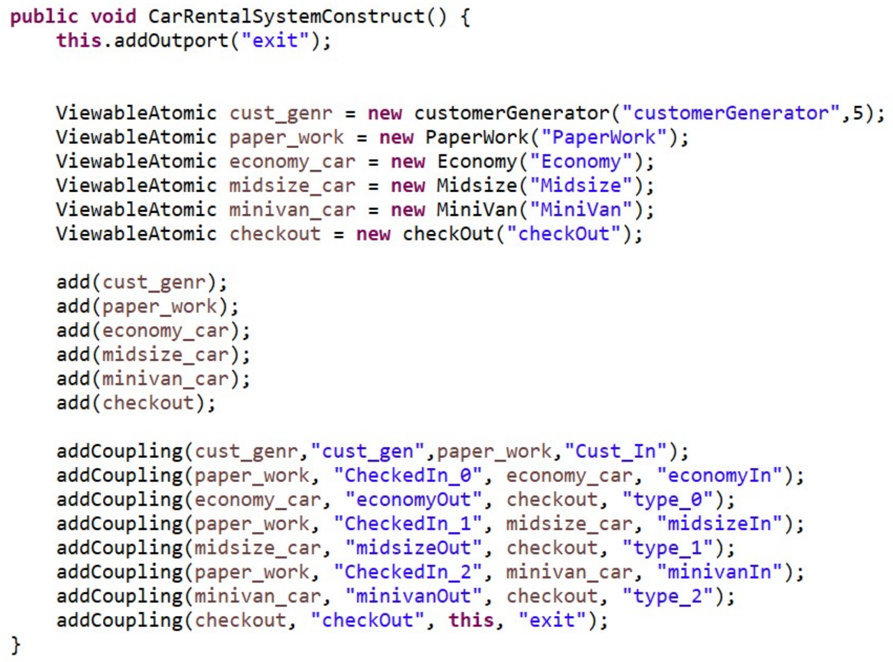
3 Modeling and Simulation Goals

The aim of this DEVSJAVA project is to understand the working of a car rental service center and to be able to accurately simulate this car rental system with the help of DEVSJAVA. The components to be focused upon are as follows. The Customer Generator component's objective is to generate customers randomly. The PaperWork is the component where based on the customer generated, the type of car they want to rent, and the number of days the car is to be rented for, the price quotation and payment, along with document filling, takes place. Then based on the type of car the customer wants, they go to the respective station/component to rent the car. Following this, all the customers assemble at the checkout regardless of the type of car rented. Here, all the documents are verified, the payment receipt is checked, and they finally exit the car rental service center or, in other words, exit the simulation.

4 Model Design and Description

4.1 Model Design

The Car Rental Simulation using DEVSJAVA has the following components: CustomerGeneration, PaperWork, Economy, Midsize, MiniVan, and CheckOut. Each of these components is interconnected/coupled with each other as follows as seen in the code snippet in Figure 2.



```
public void CarRentalSystemConstruct() {
    this.addOutport("exit");


    ViewableAtomic cust_genr = new customerGenerator("customerGenerator",5);
    ViewableAtomic paper_work = new PaperWork("PaperWork");
    ViewableAtomic economy_car = new Economy("Economy");
    ViewableAtomic midsize_car = new Midsize("Midsize");
    ViewableAtomic minivan_car = new MiniVan("MiniVan");
    ViewableAtomic checkout = new checkOut("checkOut");

    add(cust_genr);
    add(paper_work);
    add(economy_car);
    add(midsize_car);
    add(minivan_car);
    add(checkout);

    addCoupling(cust_genr,"cust_gen",paper_work,"Cust_In");
    addCoupling(paper_work, "CheckedIn_0", economy_car, "economyIn");
    addCoupling(economy_car, "economyOut", checkout, "type_0");
    addCoupling(paper_work, "CheckedIn_1", midsize_car, "midsizeIn");
    addCoupling(midsize_car, "midsizeOut", checkout, "type_1");
    addCoupling(paper_work, "CheckedIn_2", minivan_car, "minivanIn");
    addCoupling(minivan_car, "minivanOut", checkout, "type_2");
    addCoupling(checkout, "checkOut", this, "exit");
}
```

Figure 2: Flow/Coupling of the Model

The CustomerGenerator components output acts as input to the PaperWork component. The PaperWork component has three output ports based on the category of car rental the customer desires. Suppose it is of Type 0; then that customer, after making the payment goes to the Economy component; else, if the customer wants a Type 1 car. In that case, it is outputted from the second output and goes to Midsize Component; else, if the customer prefers Type 2 car, they are outputted to the MiniVan component. For the sake of simulation, the types of cars are categorized into three parts. As the business plans to set up, this can be extended to more categories in real time. Finally, from all these categories of cars rented, they finally go as input to the checkout component and exit from there. This design flow can be observed in Figure 3.

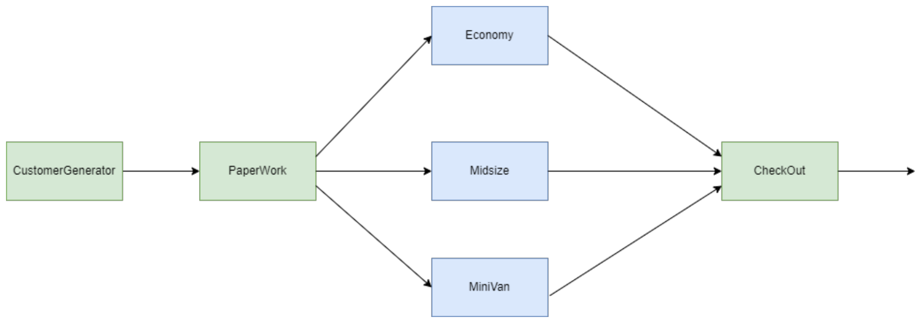


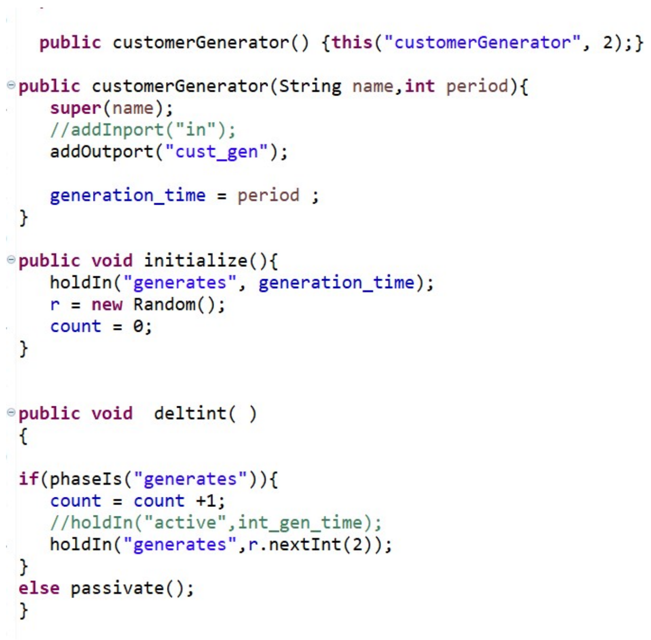
Figure 3: Design of the Car Rental Modeling Simulation

4.2 Model Description

The flow and interconnection/coupling of different components in the Car Rental Simulation has been discussed in the earlier sections. In this section, individual component description and functionality within each component is seen in detail.

4.2.1 customerGenerator

As seen in the code snippet in Figure 4, the customerGenerator component is responsible for the generation of customers at a random interval of time.



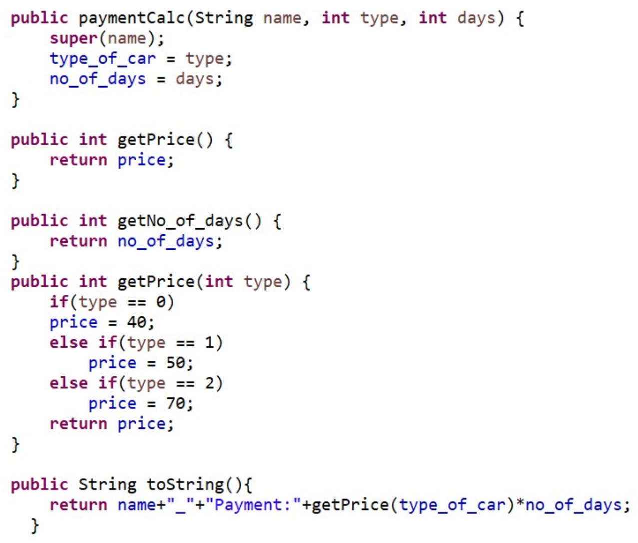
```
  public customerGenerator() {this("customerGenerator", 2);}

public customerGenerator(String name,int period){
    super(name);
    //addInport("in");
    addOutport("cust_gen");

    generation_time = period ;
 }

public void initialize(){
    holdIn("generates", generation_time);
    r = new Random();
    count = 0;
 }


public void  deltint( )
 {

 if(phaseIs("generates")){
    count = count +1;
    //holdIn("active",int_gen_time);
    holdIn("generates",r.nextInt(2));
 }
 else passivate();
 }
```

Figure 4: Code Snippet of customer Generator component

This component outputs a customer entity containing the customer name, the randomly generated type/category of a car the customer wants to rent, and the number of days the customer will be renting the car. This component keeps continuously/actively generating customers, and the output (outport) of this component is the input (inport) to the next component i.e., PaperWork.

4.2.2 PaperWork

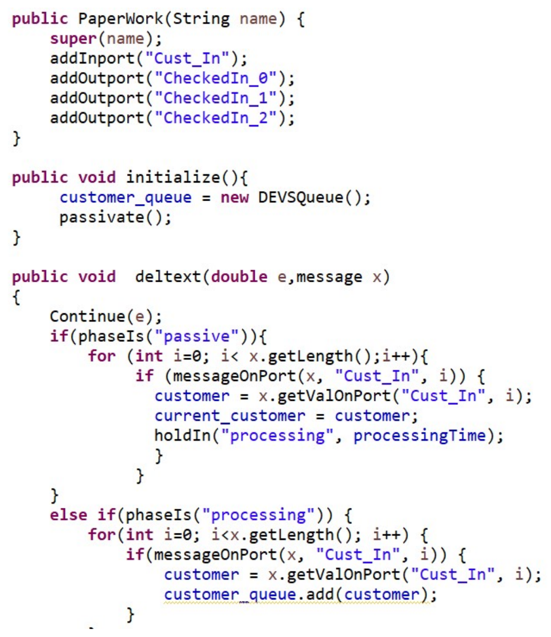
The output customer entity is the input for the PaperWork component. In the PaperWork component, based on the type of car and number of days, the amount of payment to be made is calculated as shown in paymentCalc entity in Figure 5, and the customer undergoes paperwork and payment procedures.



```
public paymentCalc(String name, int type, int days) {
    super(name);
    type_of_car = type;
    no_of_days = days;
}

public int getPrice() {
    return price;
}

public int getNo_of_days() {
    return no_of_days;
}
public int getPrice(int type) {
    if(type == 0)
    price = 40;
    else if(type == 1)
        price = 50;
    else if(type == 2)
        price = 70;
    return price;
}

public String toString(){
    return name+"_"+"Payment:"+getPrice(type_of_car)*no_of_days;
  }
```

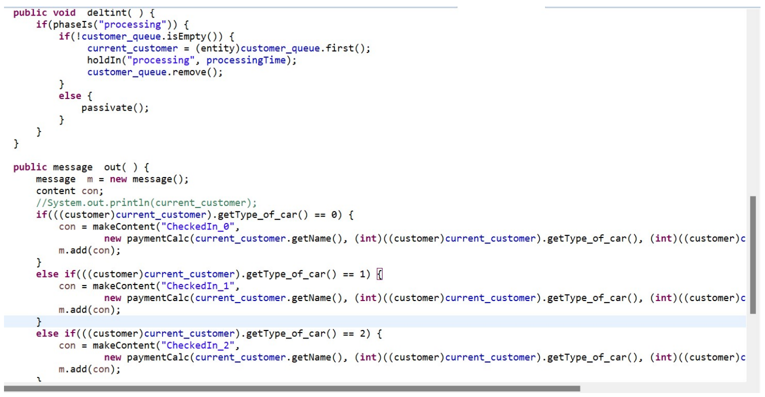
Figure 5: Code Snippet of paymentCalc entity calculating payment

There is a queue maintained here when a particular customer's paperwork is being processed the incoming customers from the customerGenerator component are added into the queue and wait for their turn.This component has three kinds of outports. Here, based on the type of car the customer wants to rent, the outport from which the content/customer comes out differs. If the customer wants type_ car i.e., Economy, they come out of the outport CheckedIn_0. If the customer wants to rent type_1 car i.e., Midsize car, they come out of the outport ChechedIn_1. Else, if the customer desires to rent a type_2 car i.e., MiniVan, they come out of the outport CheckedIn_2. The code snippet with these logics for the PaperWork component is provided in Figures 6 and 7.



```
public PaperWork(String name) {
    super(name);
    addInport("Cust_In");
    addOutport("CheckedIn_0");
    addOutport("CheckedIn_1");
    addOutport("CheckedIn_2");
}

public void initialize(){
     customer_queue = new DEVSQueue();
     passivate();
}

public void  deltext(double e,message x)
{
    Continue(e);
    if(phaseIs("passive")){
        for (int i=0; i< x.getLength();i++){
             if (messageOnPort(x, "Cust_In", i)) {
               customer = x.getValOnPort("Cust_In", i);
               current_customer = customer;
               holdIn("processing", processingTime);
               }
             }
    }
    else if(phaseIs("processing")) {
        for(int i=0; i<x.getLength(); i++) {
            if(messageOnPort(x, "Cust_In", i)) {
                customer = x.getValOnPort("Cust_In", i);
                customer_queue.add(customer);
            }
```

Figure 6: Part 1 Code Snippet of PaperWork Component



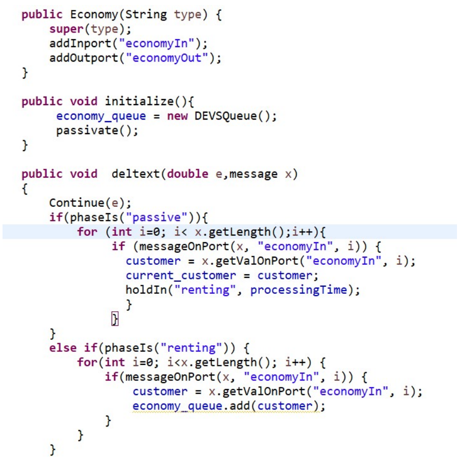
```
public void  deltint( ) {
    if(phaseIs("processing")) {
        if(!customer_queue.isEmpty()) {
             current_customer = (entity)customer_queue.first();
             holdIn("processing", processingTime);
             customer_queue.remove();
        }
        else {
            passivate();
        }
    }
}

public message  out( ) {
    message  m = new message();
    content con;
    //System.out.println(current_customer);
    if(((customer)current_customer).getType_of_car() == 0) {
        con = makeContent("CheckedIn_0",
                new paymentCalc(current_customer.getName(), (int)((customer)current_customer).getType_of_car(), (int)((customer)c
        m.add(con);
    }
    else if(((customer)current_customer).getType_of_car() == 1) {
        con = makeContent("CheckedIn_1",
                new paymentCalc(current_customer.getName(), (int)((customer)current_customer).getType_of_car(), (int)((customer)c
        m.add(con);
    }
    else if(((customer)current_customer).getType_of_car() == 2) {
        con = makeContent("CheckedIn_2",
                new paymentCalc(current_customer.getName(), (int)((customer)current_customer).getType_of_car(), (int)((customer)c
        m.add(con);
```

Figure 7: Part 2 Code Snippet of PaperWork Component

4.2.3 Economy

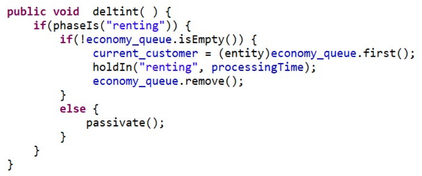
This is the type_0 category of car. The input for this component is the content outputted from the outport CheckedIn_0 of PaperWork Component. This model allows each of the customer arriving at the inport to rent a car. While a customer is renting a car, the other customers wait in the queue. The order here is first-come-first-serve (FCFS) in the queue. The outport of this is directly coupled to the inport of CheckOut component.

The code snippet for this component is in Figures 8 and 9.



```
public Economy(String type) {
    super(type);
    addInport("economyIn");
    addOutport("economyOut");
}

public void initialize(){
     economy_queue = new DEVSQueue();
     passivate();
}

public void  deltext(double e,message x)
{
    Continue(e);
    if(phaseIs("passive")){
        for (int i=0; i< x.getLength();i++){
             if (messageOnPort(x, "economyIn", i)) {
               customer = x.getValOnPort("economyIn", i);
               current_customer = customer;
               holdIn("renting", processingTime);
               }
             }
    }
    else if(phaseIs("renting")) {
        for(int i=0; i<x.getLength(); i++) {
            if(messageOnPort(x, "economyIn", i)) {
                customer = x.getValOnPort("economyIn", i);
                economy_queue.add(customer);
            }
        }
    }
```

Figure 8: Part 1 Code Snippet of Economy Component



```
public void  deltint( ) {
    if(phaseIs("renting")) {
        if(!economy_queue.isEmpty()) {
             current_customer = (entity)economy_queue.first();
             holdIn("renting", processingTime);
             economy_queue.remove();
        }
        else {
            passivate();
        }
    }
}
```

Figure 9: Part 2 Code Snippet of Economy Component

4.2.4 Midsize

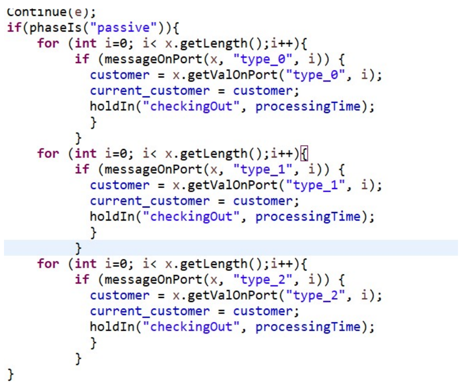
This is the type_1 category of car. The input for this component is the content outputted from the outport CheckedIn_1 of PaperWork Component. This model allows each of the customer arriving at the inport to rent a car. While a customer is renting a car, the other customers wait in the queue. The order here is first-come-first-serve (FCFS) in the queue. The outport of this is directly coupled to the inport of CheckOut component. The code snippet for this component is same as Economy.

4.3 MiniVan

This is the type_1 category of car. The input for this component is the content outputted from the outport CheckedIn_1 of PaperWork Component. This model allows each of the customer arriving at the inport to rent a car. While a customer is renting a car, the other customers wait in the queue. The order here is first-come-first-serve (FCFS) in the queue. The outport of this is directly coupled to the inport of CheckOut component. The code snippet for this component is same as Economy.

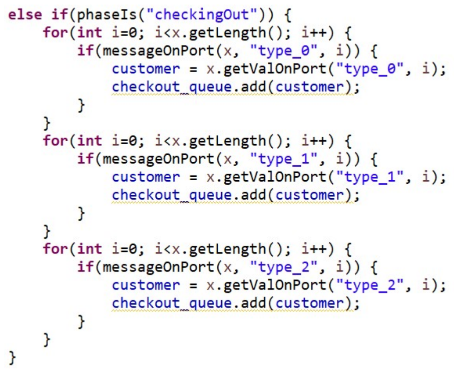
4.3.1 CheckOut

Outputs from the different categories of cars finally arrive here as input where the checkout of the customer is done. Since there are 3 different categories of cars, this component has 3 inports. This model allows each of the customer arriving at the inport to checkout. While a customer is checking out, the other customers wait in the queue. The order here is first-come-first-serve (FCFS) in the queue. Car condition, customer payment receipt and all these details are verified here and finally the customer checks out and exits the simulation/car rental service center. Figures 10 and 11 show the code snippet of external transition function of checkout component. The rest of the code is similar to other models.



```
Continue(e);
if(phaseIs("passive")){
    for (int i=0; i< x.getLength();i++){
         if (messageOnPort(x, "type_0", i)) {
           customer = x.getValOnPort("type_0", i);
           current_customer = customer;
           holdIn("checkingOut", processingTime);
           }
         }
    for (int i=0; i< x.getLength();i++){
         if (messageOnPort(x, "type_1", i)) {
           customer = x.getValOnPort("type_1", i);
           current_customer = customer;
           holdIn("checkingOut", processingTime);
           }
         }
    for (int i=0; i< x.getLength();i++){
         if (messageOnPort(x, "type_2", i)) {
           customer = x.getValOnPort("type_2", i);
           current_customer = customer;
           holdIn("checkingOut", processingTime);
           }
         }
}
```

Figure 10: When phase is Passive for CheckOut



```
else if(phaseIs("checkingOut")) {
    for(int i=0; i<x.getLength(); i++) {
        if(messageOnPort(x, "type_0", i)) {
            customer = x.getValOnPort("type_0", i);
            checkout_queue.add(customer);
        }
    }
    for(int i=0; i<x.getLength(); i++) {
        if(messageOnPort(x, "type_1", i)) {
            customer = x.getValOnPort("type_1", i);
            checkout_queue.add(customer);
        }
    }
    for(int i=0; i<x.getLength(); i++) {
        if(messageOnPort(x, "type_2", i)) {
            customer = x.getValOnPort("type_2", i);
            checkout_queue.add(customer);
        }
    }
}
```

Figure 11: When phase is Active for CheckOut

5 Simulation Model

Figure 12 is the simulation of the car rental system using DEVSJAVA and simview.

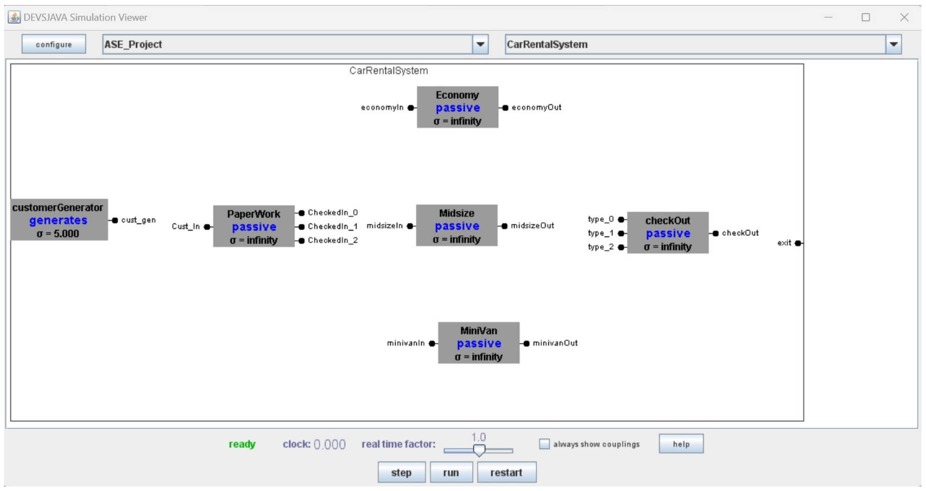
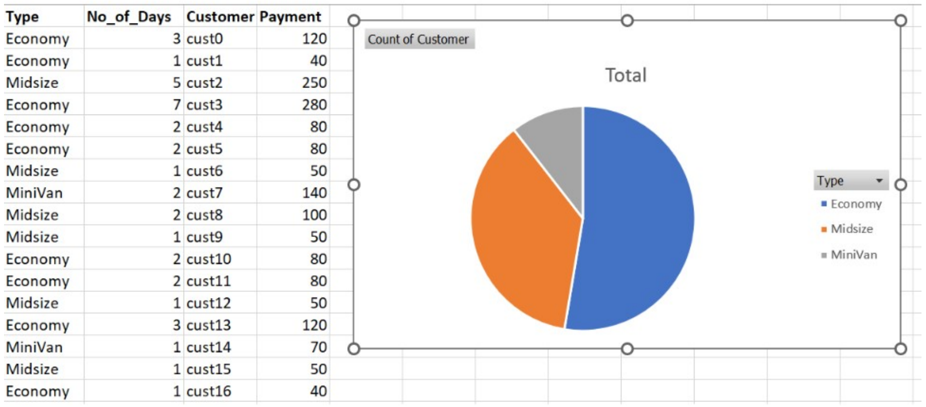


Figure 12: Simulation of Car Rental System using DEVSJAVA

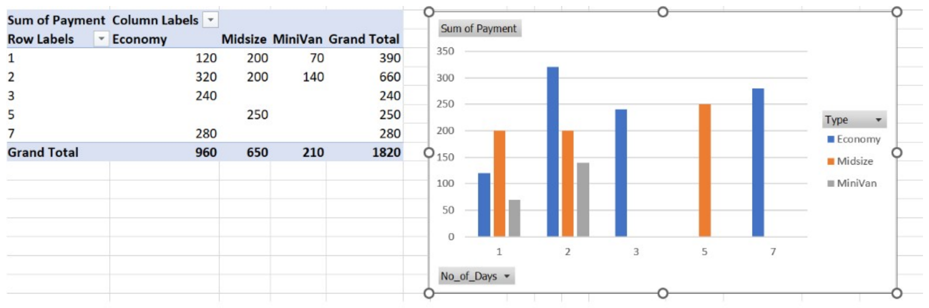
6 Observation and Result

The out messages sent from one atomic model to the other were collected to observe and understand the functioning better. The analysis performed can be observed in Figures 13 and 14. From Figure 13, it can be concluded that the majority of the customers opt for renting an economy car, followed by Midsize, and MiniVan is the least rented category of cars. This helps the person starting a rental car service center get an idea of the demand for each car category. Hence, he can make a better-informed decision concerning it. Similarly, a business can make calls to maximize their profits using analysis similar to 14. This way, a simulation replicating the close properties to the real-world help analyze and improve businesses' decision-making.



Type	No_of_Days	Customer	Payment
Economy	3	cust0	120
Economy	1	cust1	40
Midsize	5	cust2	250
Economy	7	cust3	280
Economy	2	cust4	80
Economy	2	cust5	80
Midsize	1	cust6	50
MiniVan	2	cust7	140
Midsize	2	cust8	100
Midsize	1	cust9	50
Economy	2	cust10	80
Economy	2	cust11	80
Midsize	1	cust12	50
Economy	3	cust13	120
MiniVan	1	cust14	70
Midsize	1	cust15	50
Economy	1	cust16	40

Figure 13: Distribution of the category of car preferred by customers



Sum of Payment	Column Labels			
Row Labels	Economy	Midsize	MiniVan	Grand Total
1	120	200	70	390
2	320	200	140	660
3	240			240
5		250		250
7	280			280
Grand Total	**960**	**650**	**210**	**1820**

Figure 14: Number of days car rent vs price for different categories of car

7 Conclusion

A DEVSJAVA simulation helps visualizing a process flow for better understanding of the process. In the case of the Car Rental Simulation, such a simulation would come handy to analysis the situation real-time when starting up a car rental service center. It helps understand the demand for each category of car. The flow also helps analyze to run a center such that the profit can be maximized. It also helps hiring employees according to time at each center or enhancing components according to demand and time frame at each component/stop. In conclusion, using simulation results, a business can also make better-informed decisions with respect to its business model.