La Victoria Simulation

Project Team: Fantastic Five

Team Members

Harsh Malewar	*010011521*	*harsh.malewar@sjsu.edu*
Omkar Gudekar	*010026224*	*omkar.gudekar@sjsu.edu*
Sagar Bendale	*010034765*	*sagar.bendale@sjsu.edu*
Amit Dubey	*010018892*	*amit.dubey@sjsu.edu*
Dhaval Kolapkar	*010011625*	*dhaval.kolapkar@sjsu.edu*

Table Of Contents

List of Figures

Introduction

All the real life events, which we observe, are in a continuous flow. However, on analyzing in detail, we realize that these events can be divided into discrete parts, which simplifies the analysis. This is discrete event modeling where continuous real world processes are approximated to non-continuous events defined by the observer. Common scenario of DES event would be a customer arriving at a shop or a new product being launched. The term discrete event suggests representing the system, which is being analyzed as a sequence of actions being performed on entities such as customers, documents, or data packets.

The core concepts of DES are entities, attributes, events, re- sources, queues, and time. Entities have attributes, events, consume resources, and enter queues. Attributes are specific to each entity and events are defined as things that can happen to an entity. A resource provides service to an entity. If a resource is "occupied" when an entity needs it, then that entity must wait, forming a queue.

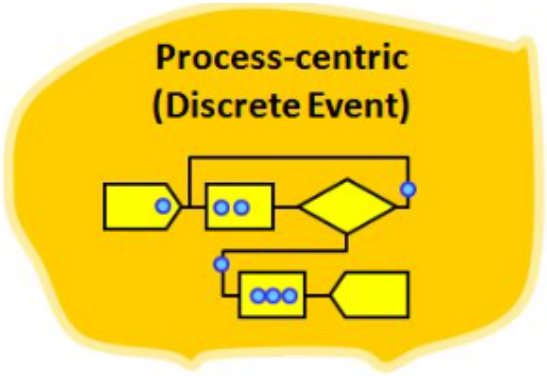

Figure 1: Discrete Event Simulation

La-Victoria

La Victoria, better known as “LA Vic’s” started off as a small, hole in the wall taqueria one block away from San Jose State University in 1998. Over the years, it has evolved in to a very popular go to place in the South Bay. Main activities that happen in it are as follows:
Customer comes to the La-Victoria and waits in the queue, for it’s turn, to place the order.
Once its turn is arrived customer interacts with the cashier at the counter. The customer then places the order and cashier asks for the payment. Once the payment is made customer receives a token number. This takes around 20-30 seconds time to place an order.
Now the customer waits till it's order is getting prepared.
At the same time cashier places the order details to the chef's queue.
As soon as chef receives the order, chef starts to prepare the items. It takes around 6-8 minutes for order to complete.
Once the order is complete, chef places it on the counter platform.
The order, which is placed on the platform, is then picked up by one of the announcer. Announcer then announces the token number about 4-5 times for which the order is prepared. Customer whose token number matches with the announcement then comes to the counter to pick up his/her order.

La-Vic simulation architecture

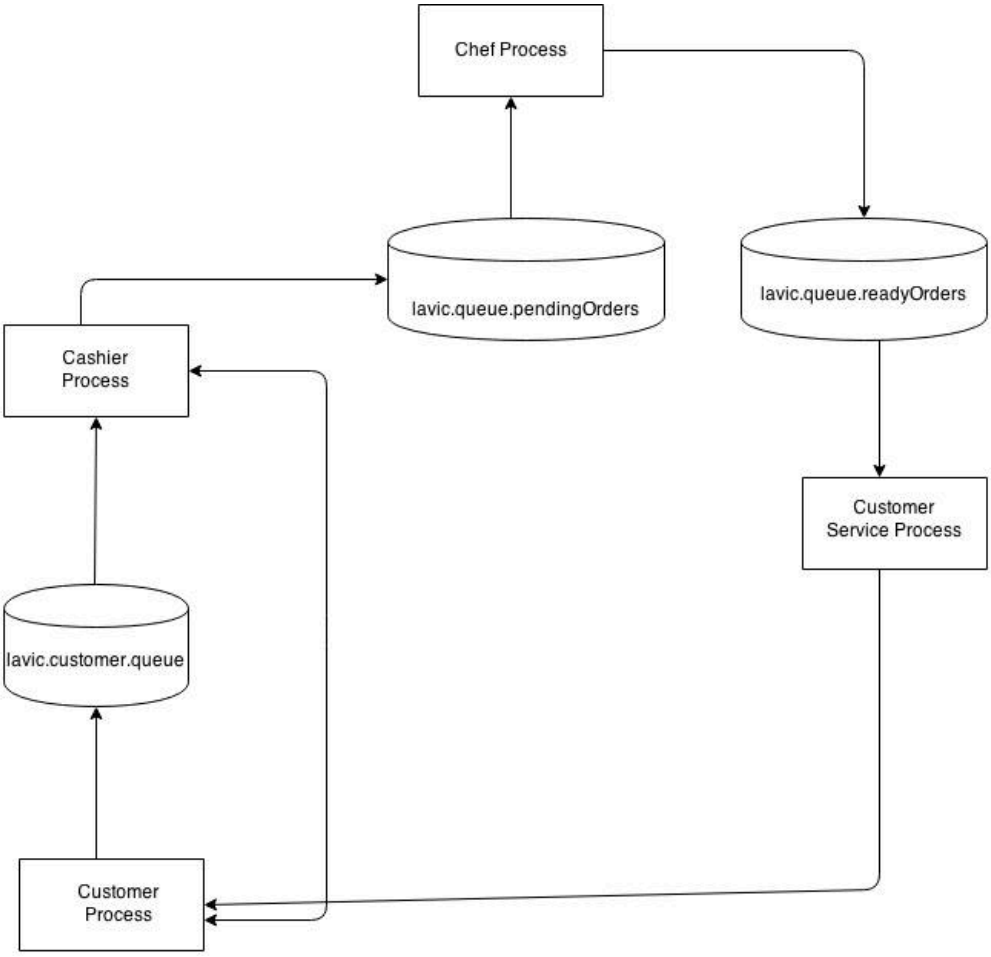

Figure 2: Architecture

To provide a programming environment for mobile robotics and artificial intelligence, which are enabling technology for real life simulation, is provided by Pyro. Pyro stands for **Py**thon **Ro**botics, which is written in Python and has the ability to define different styles of controllers, which are called the robot's brain and create a behavior or event based simulation. Due to the aforementioned reasons, we have used Pyro as the basis of our simulation.

Customer, Cashier, Chef and Customer Service are represented by entities. All these entitles enter do some processing and if any of them is not available to take input, the entity enters the corresponding queue.

Detailed Simulation

Customer Process:

```
def __init__(self, customer_name, speed, queue_to_join):
    print 'initializing customer ', customer_name
    self.customer_name = customer_name
    self.speed = speed
    self.queue_to_join = queue_to_join
```

Figure 3: Customer Init

Customer when wants to go to lavic, enters a queue and waits if the cashier entity is busy with other customers. This is done in the below code:

```
def getOrderDetails(self):
    menuObj = Pyro4.Proxy("PYRONAME:lavic.menucard")
    menu = menuObj.getmenucard()
    menuObj._pyroRelease()
    num_of_items = random.randint(1,3)
    orderDetails = []
    #print 'numer of itesm', num_of_items
    for i in range(num_of_items):
        itemName = random.choice(menu['menu']).keys()[0]
        quantity = random.randint(1,3)
        this_order = {'itemName' : itemName , 'quantity':quantity }
        orderDetails.append(this_order)
    #print 'Customer order: ', orderDetails
    return orderDetails
```

Figure 4: Get Order Details From Menu

When the customer entity reaches the queue front, the cashier welcomes the customer. Customer responds to the greeting and checks the menu card to decide what he would like to have.

Once decided, he requests for the same to the cashier. Cashier then calculates the bill and requests for payment. Customer then makes the payment and gets acknowledgement for the same.

Customer then waits for his order. Here, when the order is processed the announcer entity will announce the token id. Customer will keep on polling and checking if announcer announces his token id. When his token id is announced, he will get the order from the window.

```
def speak(self, to_person, what):
    #print 'speaking to ', to_person
    self.display._print_conversation(self.absoluteName, what['message'])
    respont_to_person = Pyro4.Proxy(to_person)
    respont_to_person.listen('customer', self.customer_name, what)
    respont_to_person._pyroRelease()

@Pyro4.expose
def listen(self, from_person, name, message):
    #print "MESSSAGE RECEIVED"
    json_message = message #json.loads(message)
    message_type = json_message['messageType']
    json_message['fromName'] = name
    #print 'message', json_message
    self.display._print_conversation(from_person, json_message['message'])
    self.request_queue.put(json_message)
    #print 'enqueued'
```

Figure 5: Customer Speak and Listen

All the communication between the customer and cashier is handle through two methods – Listen() and Speak(). Only the listen method is exposed over the network. Whatever message a customer receives through the listen method is processed by a separate thread and then the customer responds to it via the speak method.

Cashier Process:

```
def respond_to_hello(from_person,from_name,msg):
    print from_person+" ["+from_name+"] : "+json.dumps(msg)
    customer = Pyro4.Proxy("PYRONAME:"+from_name)
    customer.listen("Cashier",uri,get_order_question_msg())
    customer._pyroRelease()
```

Figure 6: Cashier Response to Customer

The Cashier entity listens to the customer queue in intervals and in the meanwhile plays mine-craft or chats with someone. If a customer arrives, he waits for customer to decide and then greets the customer with Hello, welcome to Lavic! On greeting, it asks the customer for the order.

```
def respond_to_order_request(from_person,from_name,msg):
    print from_person+" ["+from_name+"] : "+json.dumps(msg)
    customer = Pyro4.Proxy("PYRONAME:"+from_name)
    customer.listen("Cashier",uri,get_payment_request_msg(msg))
    customer._pyroRelease()
```

Figure 7: Respond to Order Request

On receiving the order, it calculates the bill and requests for the payment. Once payment is made it thanks the customer with 'Payment made '. Customer acknowledges with a 'Bye' message.

The cashier service can have multiple cashiers by creating new cashier threads.
Also, if Lavic is closed, this service informs the customer that Lavic is closed now!

Menu card:

```
def __init__(self):
    self.menu_card =json.loads(open("xyz.json").read())
```

Figure 8: Menu Card Init

The La-Vic menu card is simulated in Pyro by an entity named MenuCard.py. The menu is stored in a json file, which is loaded at runtime when the constructor is called.

```
@Pyro4.expose
def getmenucard(self):
    return self.menu_card
```

Figure 9: Get Menu Card

Events which define what happens to an entity, which here is a menu card, are defined by functions: getmenucard() and getItem(itemname). The getmenucard() returns the python object of the menu card.

```
@Pyro4.expose
def getItem(self, lookup):
    for key, value in self.menu_card.items():
        #print key
        for v in value:
            # print v
            for innerKey,innerValue in v.items():
                if lookup in innerKey:
                    return innerValue;
```

Figure 10: Get Single Item

The getItem(itemname) function returns the python object corresponding to the item requested by the other entities.

Chef process:

```
def __init__(self, name, speed):
    print "------------------------"
    self.name = name
    self.speed = speed
    print "Chef "+str(name)+" on work!"
    print "------------------------"
```

Figure 11:Chef Init

Chef process takes the input of chef name and speed of the simulation.
Speed is used to increase or decrease the preparation time

```
def check_queue(self):
    print "Monitoring Pending Order Queue"
    while True:
        self.pendingOrderQueue = Pyro4.Proxy("PYRONAME:lavic.queue.pendingOrders")   # pending queue connect
        if self.pendingOrderQueue.getNumberOfOrders() > 0:
            print "-------------------------"
            next_order = self.pendingOrderQueue.getOrder()
            token_number = int(next_order['order']['tokenNumber'])
            print "Order " + str(token_number) + " is getting prepared"
            if next_order is not None:
                if next_order['messageType'] == "ORDER_INCOMPLETE":
                    order_details = next_order['order']['orderDetails']
                    self.prepare(order_details)
                    next_order['messageType'] = "ORDER_COMPLETE"
                    print "Order Completed"
                    self.completedOrderQueue = Pyro4.Proxy("PYRONAME:lavic.queue.readyOrders")  # completed queue connect
                    self.completedOrderQueue.add_order(next_order)
                    self.completedOrderQueue._pyroRelease()                          # completed queue release
                    print "Ready to Serve"
                    self.pendingOrderQueue._pyroRelease()                            # pending queue release
            else:
                print "Null Data"
            print "-------------------------"
        else:
            sleep(1.0/self.speed)
```

Figure 12: Monitor Pending Order Queue

Chef monitors the pending order queue. As soon as chef sees the item in the pending order queue it starts the preparation of the order.

```
def prepare(self,order_details):
    order_details_size = len(order_details)-1
    while order_details_size >= 0:
        quantity = int(order_details[order_details_size]['quantity'])
        item_name = order_details[order_details_size]['itemName']
        print "Preparing "+str(item_name)+" Quantity: "+str(quantity)
        time_to_prepare = self.get_time_from_menu(item_name)
        time_to_prepare *= float(quantity)
        print "Time to prepare: "+str(time_to_prepare)
        sleep(time_to_prepare/self.speed)
        order_details_size -= 1
```

Figure 13: Preparation of Order

As soon as the items are present in the pending order queue chef starts the preparation of the order. Chef also checks the quantity of the requested order and prepares the multiple items in the same order.

```
def get_time_from_menu(self,item_name):
    self.menuServer = Pyro4.Proxy("PYRONAME:lavic.menucard")  # menu-card queue connect
    item_json = self.menuServer.getItem(item_name)
    self.menuServer._pyroRelease()                            # menu-card queue release
    time = float(item_json['prep_time'])
    return time
```

Figure 14: Interaction with Menu Card to get Item Details

Chef interacts with the menu-card service to get the time for preparation of the items.
One chef is responsible for preparing one whole order. We can add multiple chefs in the simulation environment.

Server:

Server takes the name as input for the program to start.

```
def activate():
    server = Server()
    myIp = str(socket.gethostbyname(socket.gethostname()))
    daemon=Pyro4.Daemon(myIp)
    ns=Pyro4.locateNS()
    uri=daemon.register(server)
    global my_uri
    my_uri = uri
    global my_name
    print_success("My Url " + str(uri))
    daemon.requestLoop()
```

Figure 15: Server Registration on Pyro4 Name Server

This is the method, which will register the Server to the Pyro4 name server. From there customers can listen to the announcement and take the order.

```
def serverCustomer():
    clear_screen()
    global my_name
    print_main_title("NEW SERVER " + str(my_name).upper())
    print_success("Time to work!")
    global order_json
    queue_address = "PYRONAME:lavic.queue.readyOrders"
    print_network("Connecting to " + queue_address)
    try:
        orders = Pyro4.Proxy(queue_address)
        while True:
            try:
                if int(orders.get_size()) > 0:
                    parsed_json = orders.get_order()
                    if parsed_json is not None:
                        message_type = parsed_json['messageType']
                        if message_type == "ORDER_COMPLETE":
                            print_info("I've got an order to announce")
                            token_number = parsed_json['order']['tokenNumber']
                            print_title("Announcing token "+ str(token_number))
                            parsed_json['messageType'] = "ORDER_PREPARED"
                            order_json = parsed_json
                            broadcast_customers()
                time.sleep(6)
            except:
                print_error("Some problem getting json")
    except:
        print_error("Could not connect to queue.")
```

Figure 16: Monitoring the Completed Order Queue

Server will keep on monitoring the ready order queue. As soon as any order is present in the queue it will give call to the broadcast customer from the customer process.

```
def broadcast_customers():
        global my_uri
        global order_json
        ns=Pyro4.locateNS()
        customer_list = ns.list("lavic.customer.")
        shout_limit = 3
        count = 0
        for count in range(0, shout_limit):
            if not order_json is None:
                token_number = str(order_json['order']['tokenNumber'])
                print_info("Token number " + token_number + " Please take your order - Announcement: " + str(count + 1))
                for customer in customer_list:
                    pyro_address = customer_list[customer]
                    print_network("Announcing tokens to " + str(customer) + " : " + pyro_address)
                    try:
                        global my_uri
                        customer_obj = Pyro4.Proxy(pyro_address)
                        customer_obj.listen("server", my_uri, order_json )
                        print_success("Sent")
                    except:
                        print_error("Some problem connecting to customers :" + str(customer))
            else:
                order_json = None
                return
            time.sleep(1)
        print_info("No takers. Sorry, disposing order.")
```

Figure 17: Broadcast to Customers

Broadcast customer takes all customer proxy object uri. Server broadcasts the token to all the customers. Customer whose token matches will take order. Server will announce up to 3 times before he dispatches the order.

```
class Server(object):
    @Pyro4.expose
    def take_order(self):
        global order_json
        response_json = order_json
        order_json = None
        print_title("Hey Thanks! Please take your order!")
        return response_json
```

Figure 18: Take Order

Take order method is exposed to the name server, so that the customers can call the method to get their order. Once the customer gets the order it is removed from the queue. And the next order starts broadcasting.

Queues

Customer queue:

```
custQueue=CustomerQueue()
myIp = str(socket.gethostbyname(socket.gethostname()))
daemon=Pyro4.Daemon(myIp)
ns=Pyro4.locateNS()
uri=daemon.register(custQueue)
ns.register("lavic.queue.customer", uri)
print "Customer Queue Started on : ", uri
daemon.requestLoop()
```

Figure 19: Customer Queue- Pyro4 Name Server Registration

It is a service that is registered on the pyro4 name server

```
class CustomerQueue(object):

    @Pyro4.expose
    def register(self, name):
        print name + " Joined the Customer Queue"
        queue.put(name)

    @Pyro4.expose
    def getNextCustomer(self):
        return queue.get()

    @Pyro4.expose
    def get_size(self):
        return queue.qsize()
```

Figure 20: Customer Queue

Method register is used to add the customers to the queue. Get next Customer gives the customer in FIFO order. And get size is used to check current size of the queue.

Pending order queue:

```
myIp = str(socket.gethostbyname(socket.gethostname()))
orderQueue=PendingQueue()
daemon = Pyro4.Daemon(myIp)                 # make a Pyro daemon
ns = Pyro4.locateNS()
uri = daemon.register(orderQueue)             # register the greeting object as a Pyro object
ns.register("lavic.queue.pendingOrders", uri)
print "Ready. Object uri =", uri      # print the uri so we can use it in the client later
daemon.requestLoop()                  # start the event loop of the server to wait for calls
```

Figure 21: Pending Order Queue- Pyro4 Name Server Registration

Pending order queue is a name service created using pyro4 name service loop.

```
pendingQueue = Queue();
class PendingQueue:

    @Pyro4.expose
    def addOrder(self, order):
        pendingQueue.put(order)
        print "-------------------------"
        print order
        print "-------------------------"

    @Pyro4.expose
    def getOrder(self):
        if(pendingQueue.not_empty):
            return pendingQueue.get()
        else:
            return 0

    @Pyro4.expose
    def getNumberOfOrders(self):
        return pendingQueue.qsize()
```

Figure 22: Pending Order Queue

Methods to add to the queue, get queue size and get item from queue are exposed. It takes the JSON data as input.

Completed order queue

```
order = OrderQueue()
myIp = str(socket.gethostbyname(socket.gethostname()))
daemon=Pyro4.Daemon(myIp)
ns=Pyro4.locateNS()
uri=daemon.register(order)
ns.register("lavic.queue.readyOrders", uri)
print("Order Queue started on : " + str(uri))
daemon.requestLoop()
```

Figure 23: Completed Order Queue- Pyro4 Name Server Registration

This is the code for completed order queue registration on the name server.

```
class OrderQueue(object):

    @Pyro4.expose
    def get_order(self):
        return ordersQueue.get()

    @Pyro4.expose
    def add_order(self, json_message):
        print("Order Ready : "+json_message['order']['tokenNumber'])
        ordersQueue.put(json_message)

    @Pyro4.expose
    def get_size(self):
        return ordersQueue.qsize()
```

Figure 24: Completed Order Queue

It has methods to get order, add order and get the current size of the order queue.

Example of Simulation

Output of the Simulation:

Environment:

Set Up Environment:

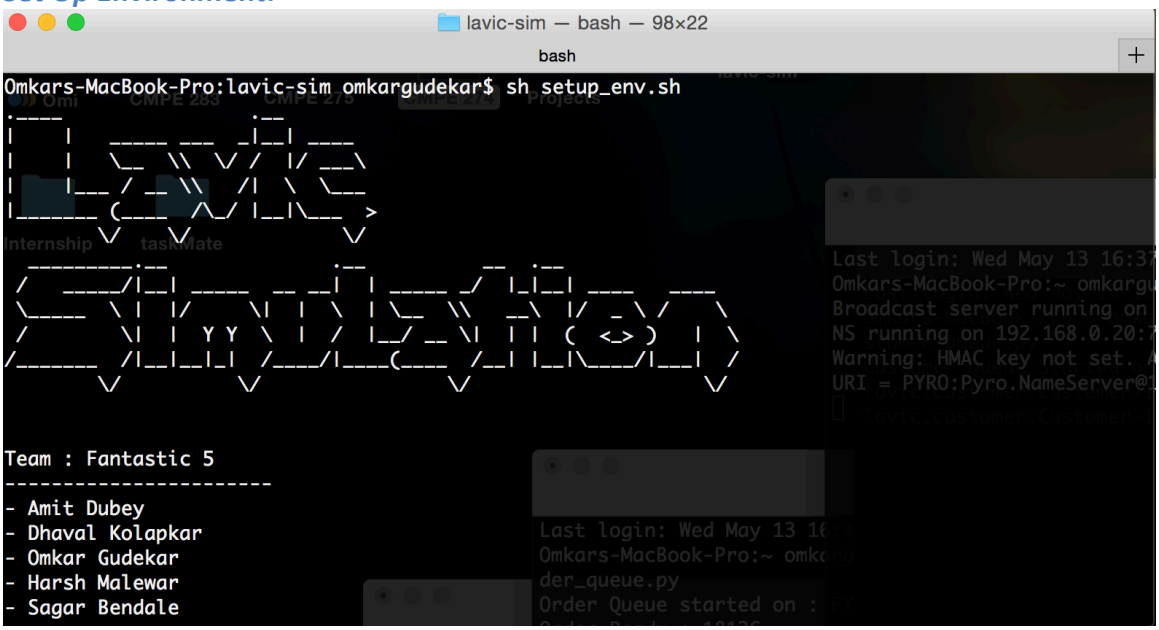

Figure 25: Setting up the environment

Name Server:

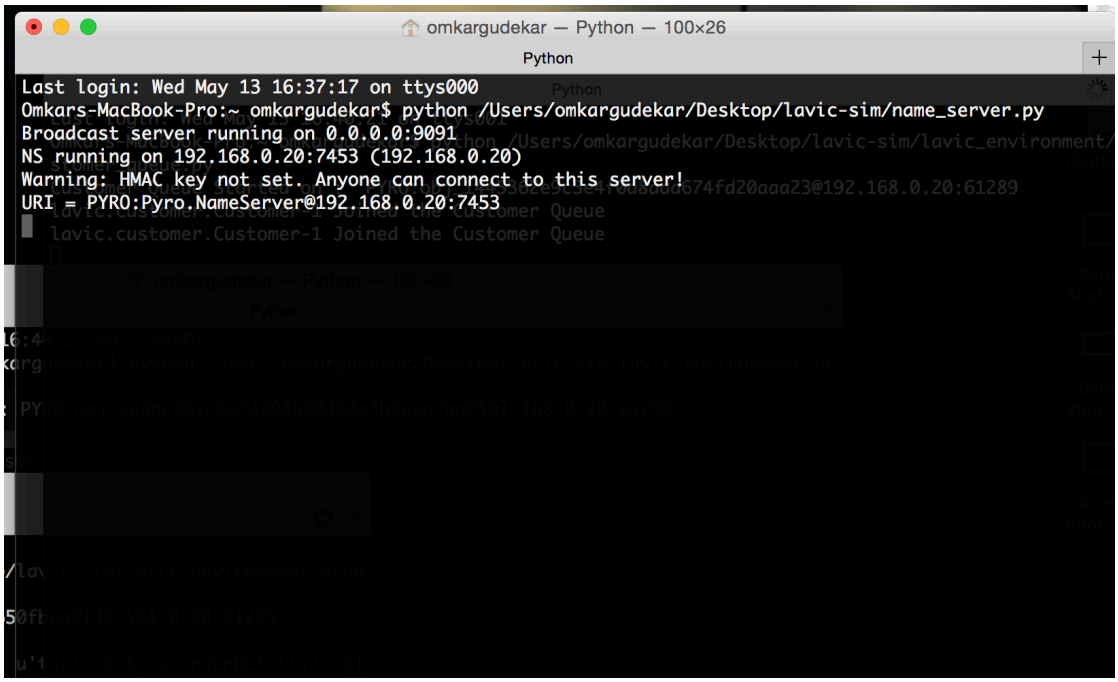

Figure 26: Pyro4 Name Server

Customer Queue:

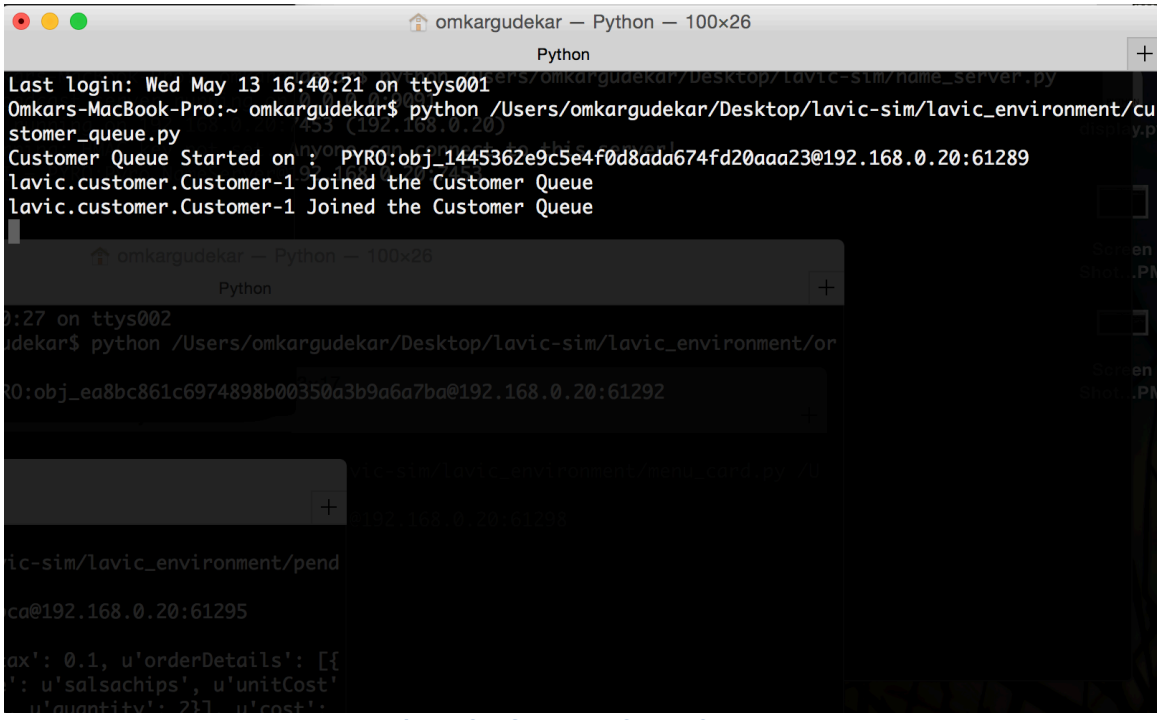

Figure 27: Customer Queue Output

Pending Order Queue:

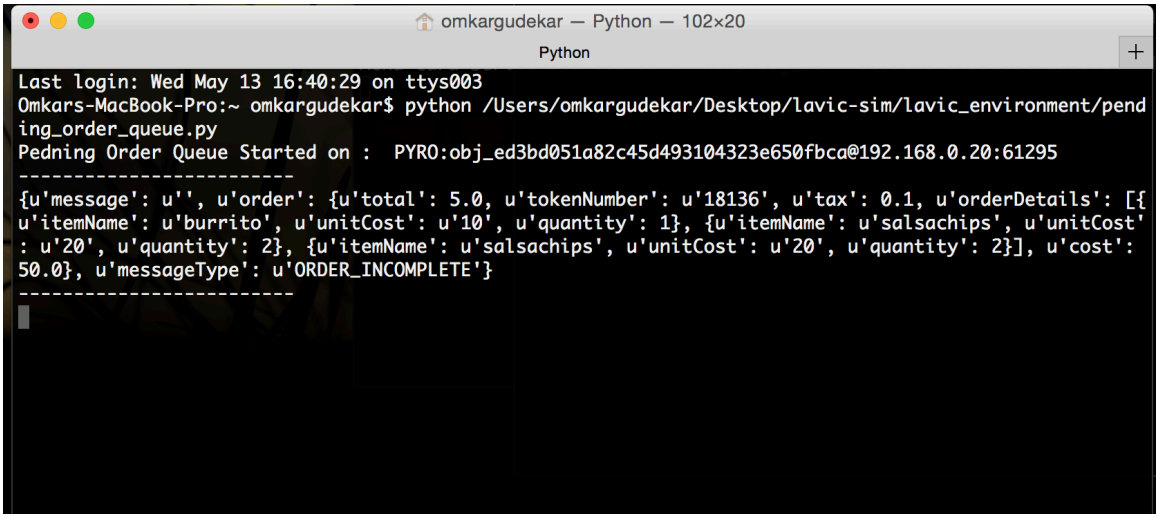

Figure 28: Pending Order Queue Output

Ready Order Queue:

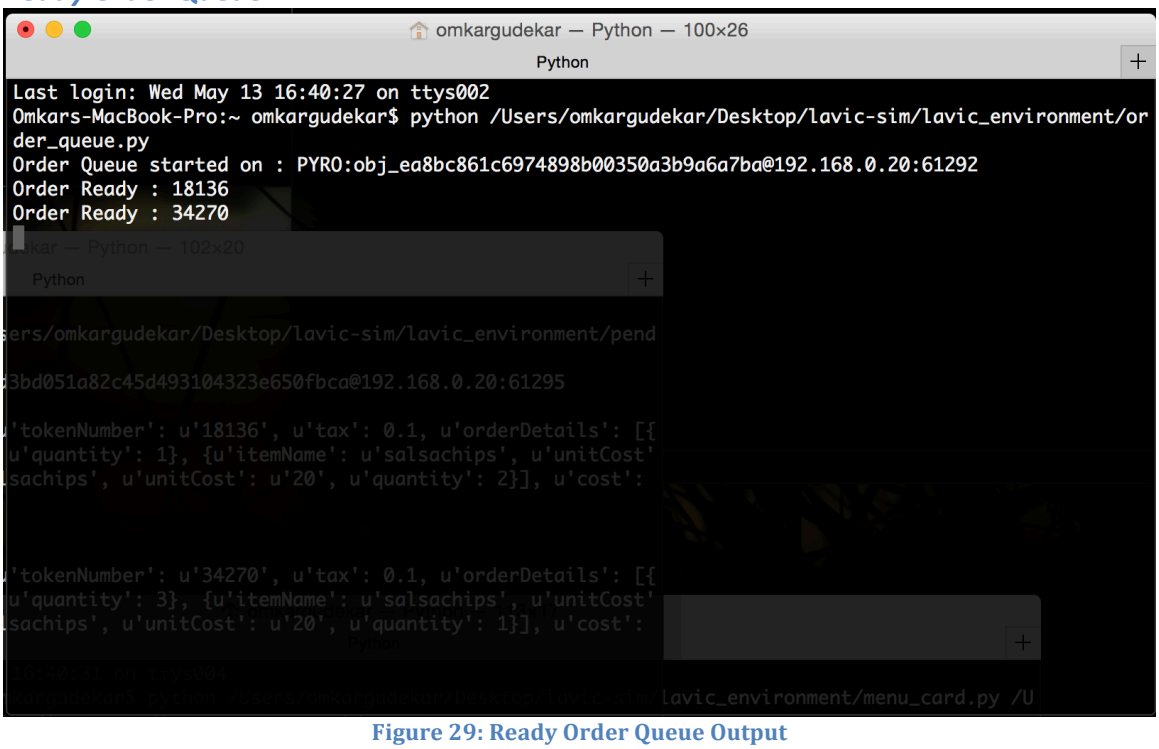

Figure 29: Ready Order Queue Output

Customer:

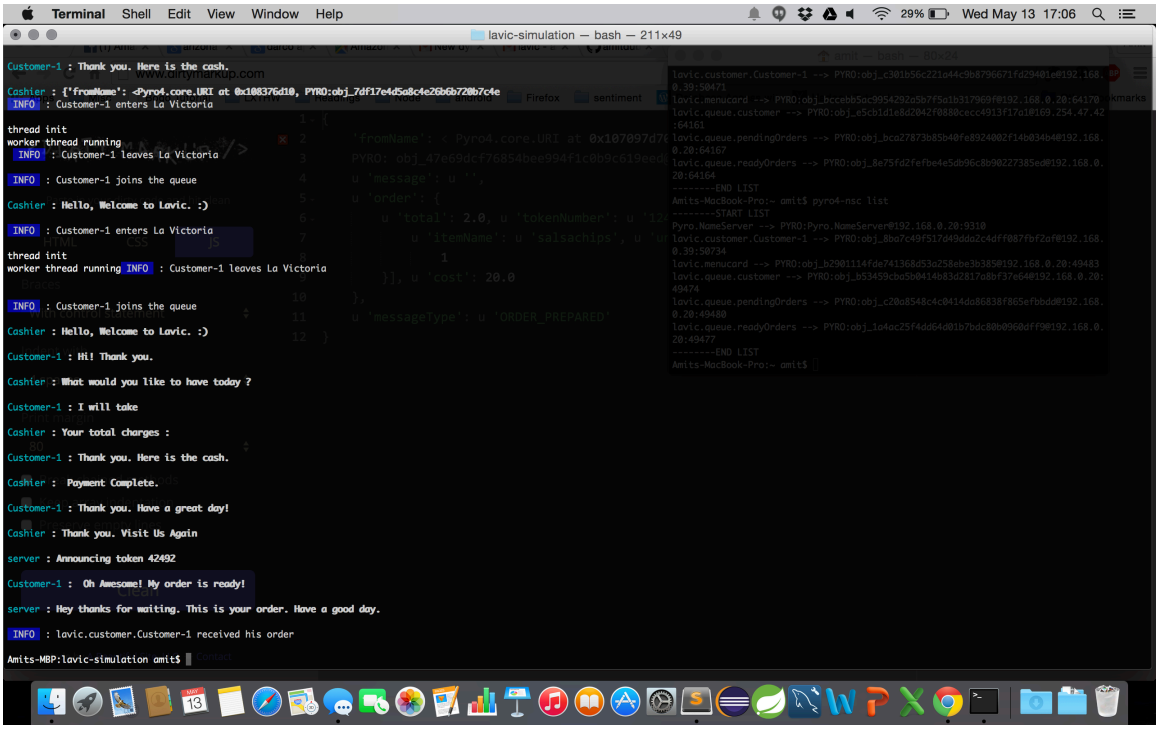

Figure 30: Customer Output

Cashier:

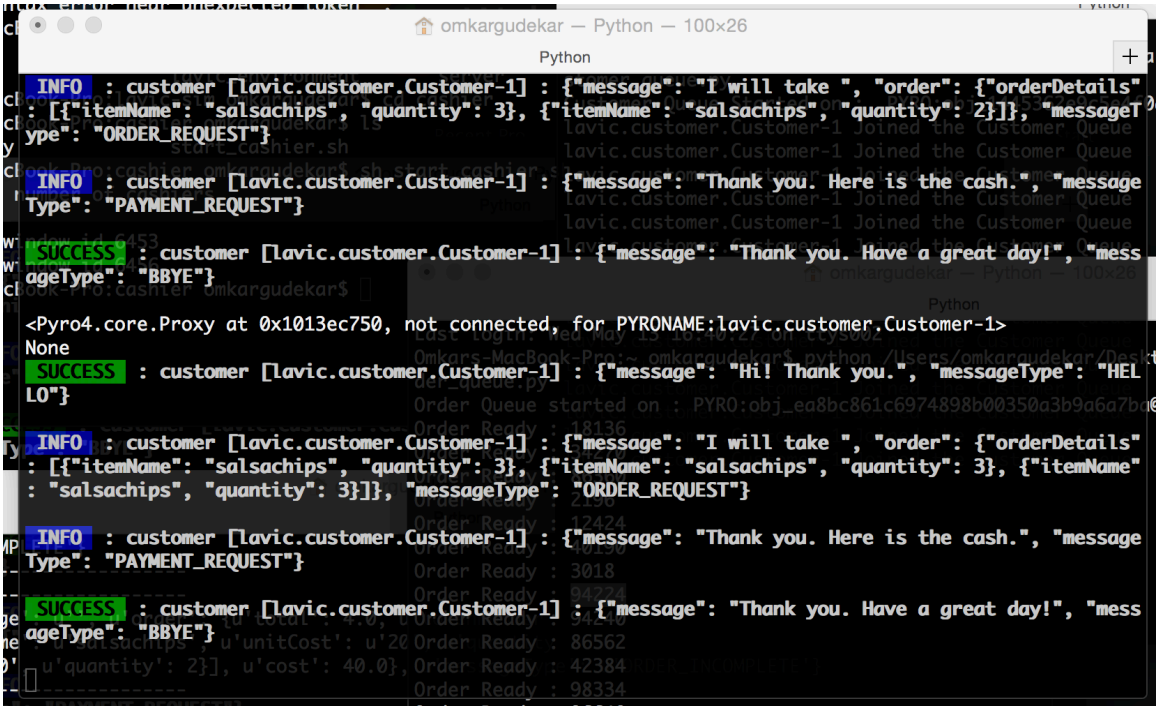

Figure 31: Cashier Output

Chefs Working:

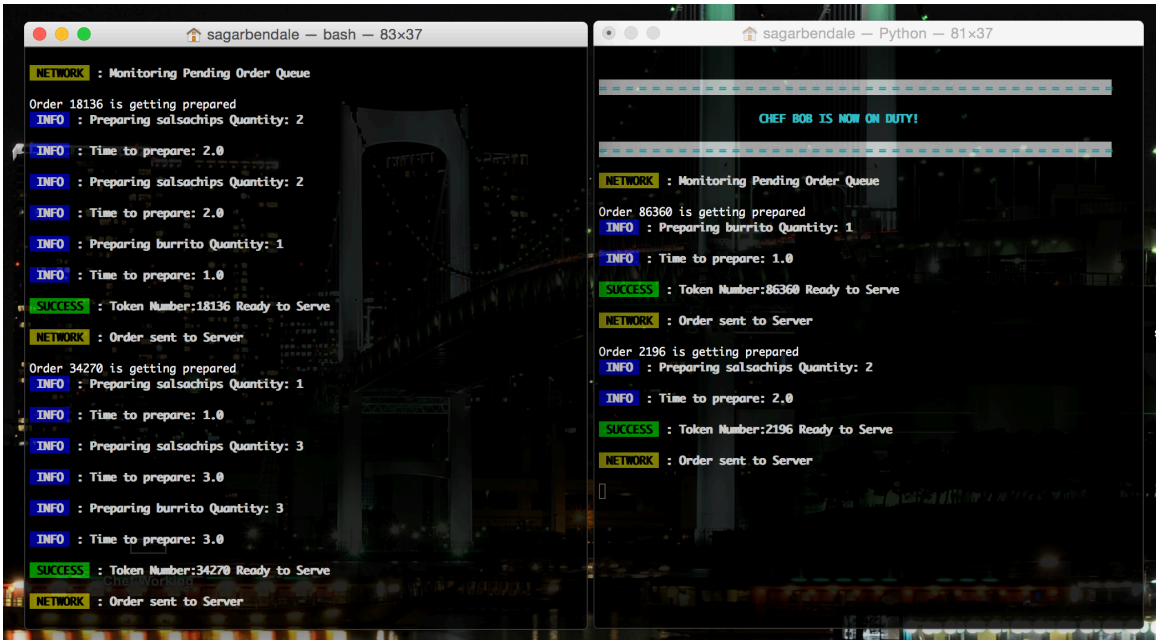

Figure 32: Chef Output

Servers:

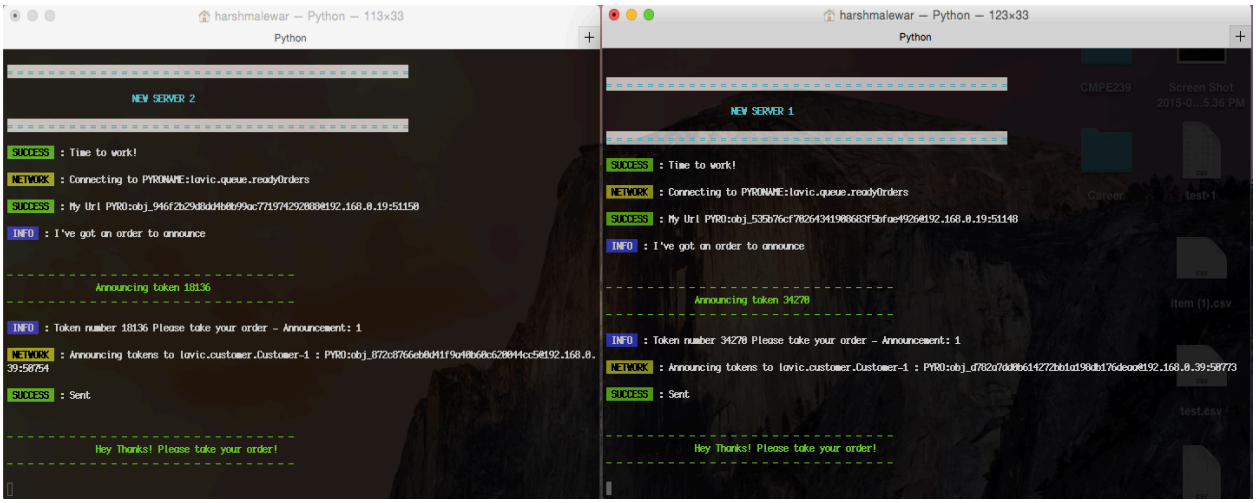

Figure 33: Servers Output

References

[1]http://www.anylogic.com/discrete-event-simulation
[2]http://www.lavicsj.com/LaVictoria.html
[3]http://www.ispor.org/workpaper/Modeling_Methods/Modeling_using_Discrete_Event_Simulation-4.pdf
[4]https://pythonhosted.org/Pyro4/
[5] http://stackoverflow.com/questions/tagged/python
[6]https://docs.python.org/2/library/queue.html
[7]http://pyrorobotics.com
[8]https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man1/osascript.1.html