Process Mining | SE4009 | Assignment:02

JANUARY 19, 2024

Daniyal Khan : 20i-1847

Assignment | Process Mining

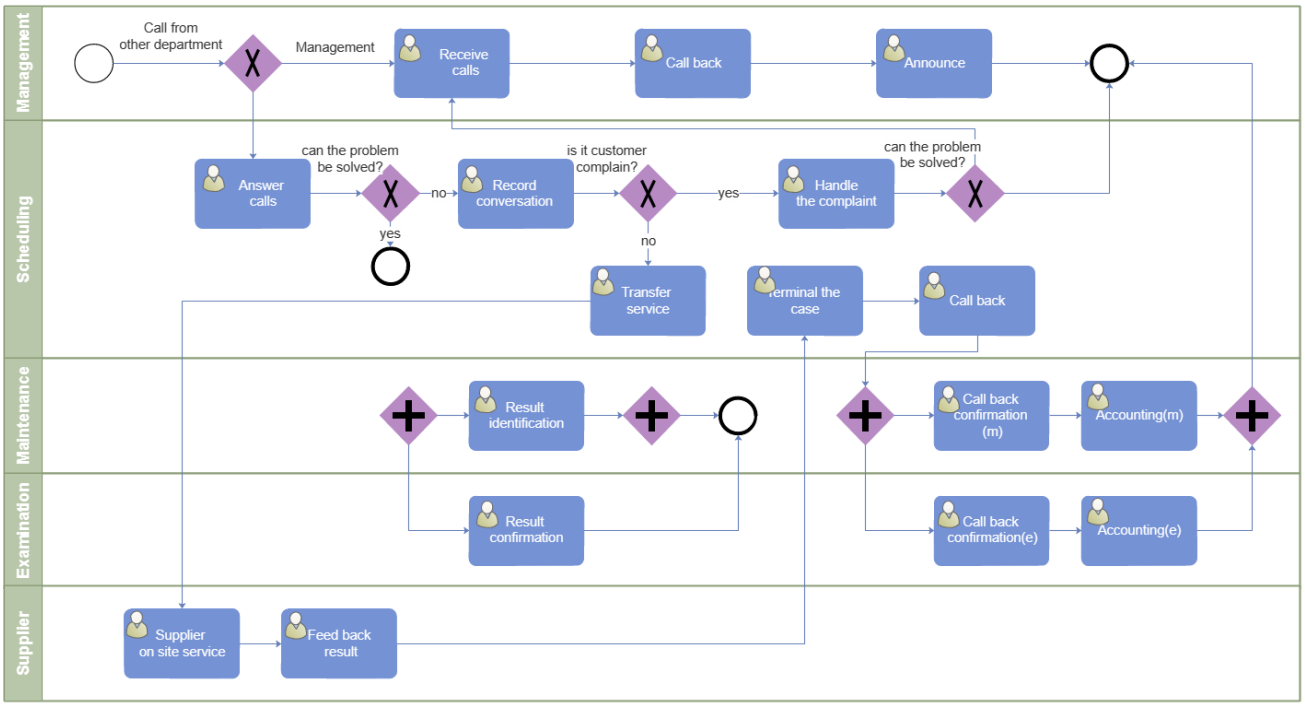
BPMN: Coding

20i-1847 Daniyal Khan

Call Complaint BPMN

Contents

Visual Representation:



Coding The BPMN:

Code: Part 1

A basic code of How a call complaint is dealt:

```
import simpy
import random

RANDOM_SEED = 42
SIM_TIME = 120

NUM_AGENTS = 2
NUM_CALLS = 5

def generate_calls(env, num_calls, call_center):
    for i in range(num_calls):
        call = Call(i)
        call_center.put(call)
```

```
        print(f"Call {call.id} generated at {env.now}")
        yield env.timeout(random.randint(1, 10))

class Call:
    def __init__(self, id):
        self.id = id
        self.arrival_time = 0
        self.service_time = 0
        self.departure_time = 0

class Agent:
    def __init__(self, id, env, call_center):
        self.id = id
        self.env = env
        self.call_center = call_center
        self.busy = False

    def agent_process(self):
        while True:
            call = yield self.call_center.get()
            self.busy = True
            call.arrival_time = self.env.now
            print(f"Call {call.id} taken by agent {self.id} at {self.env.now}")
            yield self.env.timeout(random.randint(5, 15))
            call.service_time = self.env.now - call.arrival_time
            call.departure_time = self.env.now
            self.busy = False
            print(f"Call {call.id} completed by agent {self.id} at {self.env.now}")

def main():
    print("Call Complaint BPMN Simulation")
    random.seed(RANDOM_SEED)
    env = simpy.Environment()
    call_center = simpy.Store(env)

    for i in range(NUM_AGENTS):
        agent = Agent(i, env, call_center)
        env.process(agent.agent_process())

    env.process(generate_calls(env, NUM_CALLS, call_center))
    env.run(until=SIM_TIME)

if __name__ == '__main__':
    main()
```

Explanation:

1. **Call** Class:
    - Represents a call entity in the call center simulation.
    - Attributes:
        - **id**: Unique identifier for the call.
        - **arrival_time**: The time when the call arrives.
        - **service_time**: The time taken to service the call.
        - **departure_time**: The time when the call is completed.
2. **Agent** Class:
    - Represents an agent in the call center simulation.
    - Attributes:
        - **id**: Unique identifier for the agent.
        - **env**: The simulation environment (SimPy).
        - **call_center**: The call center store where calls are placed.
        - **busy**: A flag indicating if the agent is currently busy with a call.
    - **agent_process()** method:
        - Simulates the behavior of the agent.
        - The agent continuously checks for calls in the call center store.
        - When a call is received, the agent sets the busy flag, records the arrival time, and starts servicing the call.
        - The service time is determined by a random timeout between 5 and 15 minutes.
        - After servicing the call, the agent updates the service and departure times, clears the busy flag, and repeats the process.
3. **generate_calls()** Function:
    - Generates calls and puts them into the call center store.
    - Parameters:
        - **env**: The simulation environment (SimPy).
        - **num_calls**: The total number of calls to be generated.
        - **call_center**: The call center store.
    - For each call, it creates a **Call** object, puts it into the call center store, and prints the call generation time.

- The inter-arrival time between calls is determined by a random timeout between 1 and 10 minutes.

4. **main()** Function:
    - The entry point of the program.
    - Sets up the simulation environment, call center store, agents, and call generation process.
    - Starts the simulation by calling **env.run()** and specifying the simulation end time (**SIM_TIME**).
5. Program Execution:
    - Prints a header message indicating the start of the call center simulation.
    - Sets the random seed for reproducibility.
    - Creates the simulation environment (**env**) and the call center store (**call_center**).
    - Creates the specified number of agents and their corresponding processes using **Agent** class instances.
    - Creates the call generation process using **generate_calls()** function.
    - Starts the simulation by calling **env.run()** and specifying the simulation end time (**SIM_TIME**).

Overall, the code simulates a call center scenario where calls are generated over time, placed in a queue (call center store), and serviced by available agents. The simulation captures the arrival, service, and departure times of the calls, providing insights into the call center's performance and efficiency.

Code: Part:2

A specific code of the current process:

```
import simpy
import random

def customer(env, id, management_queue, scheduling_queue):
    arrival_time = env.now
    print(f'Customer {id} calls at {arrival_time}')

    # Determine which department will handle the call
    if random.random() < 0.5:
        with management_queue.request() as req:
            yield req
            problem_type = determine_problem_type()
```

```
    else:
        with scheduling_queue.request() as req:
            yield req
            problem_type = determine_problem_type()

    if problem_type == "customer_online_complaint":
        solution_exists = scheduling_decides_solution()
        if solution_exists:
            provide_solution(id)
        else:
            record_problem(id)
            handle_problem_and_provide_solution(id)
    else:
        transfer_process_to_supplier()
        report = prepare_report_by_supplier()
        send_report_to_scheduling(report)
        call_customer_and_send_onsite_team(id)
        transfer_process_to_maintenance()

def determine_problem_type():
    return random.choice(["customer_online_complaint", "onsite_complaint"])

def scheduling_decides_solution():
    return random.choice([True, False])

def provide_solution(customer_id):
    print(f'Solution provided to customer {customer_id}')

def record_problem(customer_id):
    print(f'Recorded problem of customer {customer_id}')

def handle_problem_and_provide_solution(customer_id):
    print(f'Handled problem and provided solution to customer {customer_id}')

def transfer_process_to_supplier():
    pass

def prepare_report_by_supplier():
    return {}

def send_report_to_scheduling(report):
    pass

def call_customer_and_send_onsite_team(customer_id):
    print(f'Called customer {customer_id} and sent onsite team')
```

```
def transfer_process_to_maintenance():
    pass

# Simulation
random.seed(42)
env = simpy.Environment()
management_queue = simpy.Resource(env, capacity=1)
scheduling_queue = simpy.Resource(env, capacity=1)

for i in range(10):
    env.process(customer(env, i, management_queue, scheduling_queue))
    env.run(until=env.now + random.expovariate(1 / 5))
```

Explanation:

This code simulates customer calls with random arrival times and random problem types. It uses the SimPy library to model the process and queues. The code doesn't contain a real implementation for some of the tasks, like transferring the process to the supplier, preparing a report, sending a report to scheduling, etc. You can replace the pass statements and dummy return values with real implementations according to your requirements.