CSC 584 --- Software Project Planning and Management

Project Progress Report

Project Title: Simulated IoT Network Security System
Names: Atif Ur Rahman Mohammed
Zubair Ahmed Mohammed

Description of the Work That Has Been Done:

IoT Network Simulation Setup:

- Developed a simulation environment using Python with Simpy and networkx libraries.
- Created basic classes for IoT devices: Sensor, Actuator, and Controller.
- Implemented a MessageBroker class to handle communication between devices.
- Simulated sensors generating different types of data (temperature, humidity, motion).

Device Communication:

- Established a message-passing mechanism where devices communicate through the MessageBroker.
- Sensors send data to controllers, which process the data and send commands to actuators.

Network Graph Structure:

- Created a network graph representing the connections between sensors, controllers, and actuators.
- Added logic to simulate connections: each sensor connected to all controllers, and each controller connected to all actuators.

Basic Device Interactions:

- Simulated sensors generate data and send it to controllers.
- Controllers process sensor data and issue commands to actuators based on simple logic.

Remaining Work to Be Done to Complete the Project Successfully:

Implement Robust Security Features (Due by [4/10/2024]):

- Introduce encrypted communication between devices using symmetric/asymmetric encryption.
- Develop more sophisticated anomaly detection algorithms.
- Implement device authentication mechanisms.

Enhance Network Dynamics (Due by [4/13/2024]):

- Simulate more complex network behaviors such as dynamic device connections/disconnections.
- Implement more realistic data generation and processing in sensors and controllers.

Performance Analysis and Testing (Due by [4/17/2024]):

- Test the network under various scenarios to evaluate the effectiveness of security features.
- Analyze the performance of the network in terms of data throughput, latency, and security.

Final Report and Documentation (Due by [4/21/2024]):

- Prepare a comprehensive final report detailing the project, methodology, findings, and conclusions.
- Ensure thorough documentation of the code and the network setup.

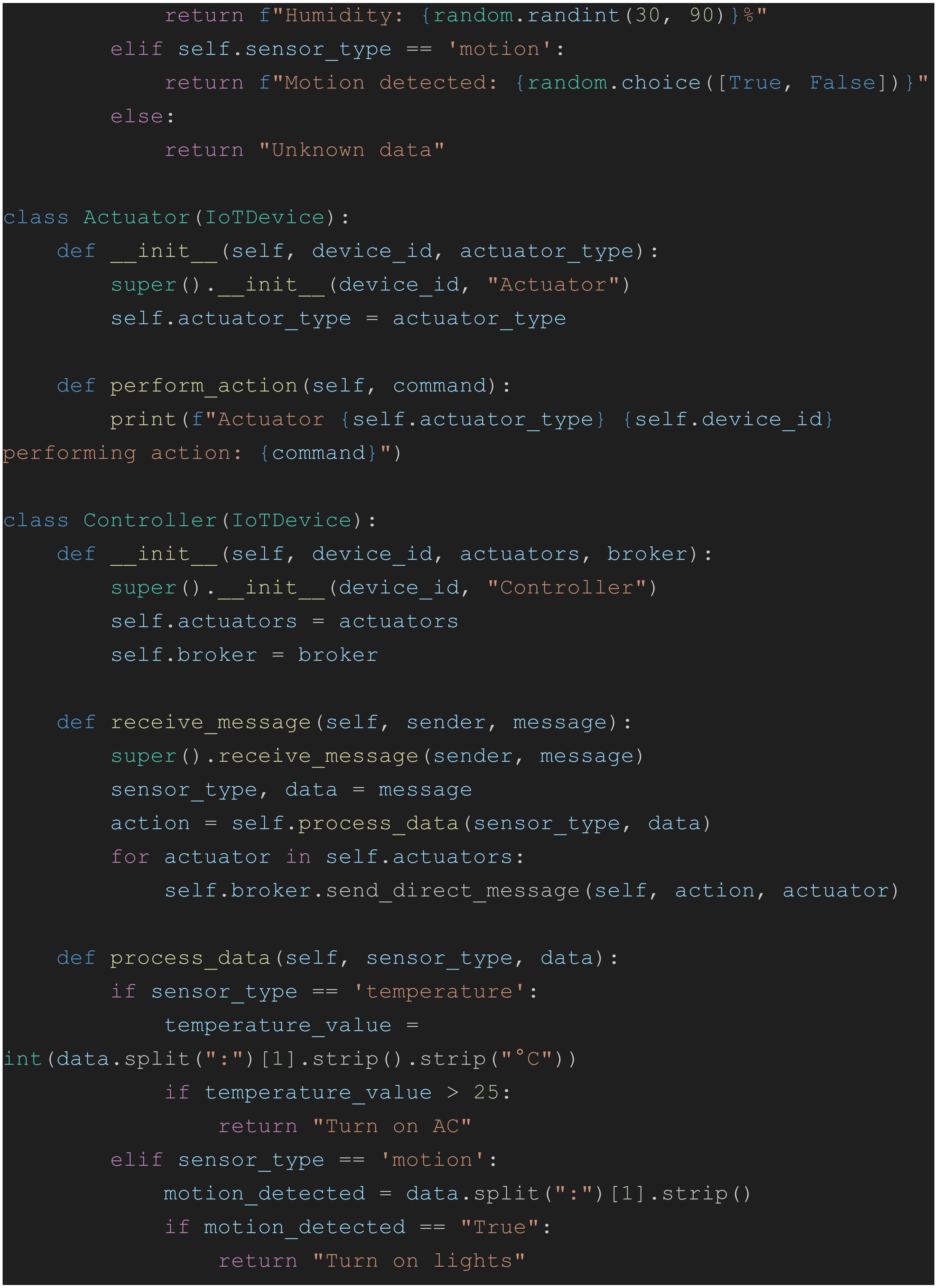
Appendix:

main.py

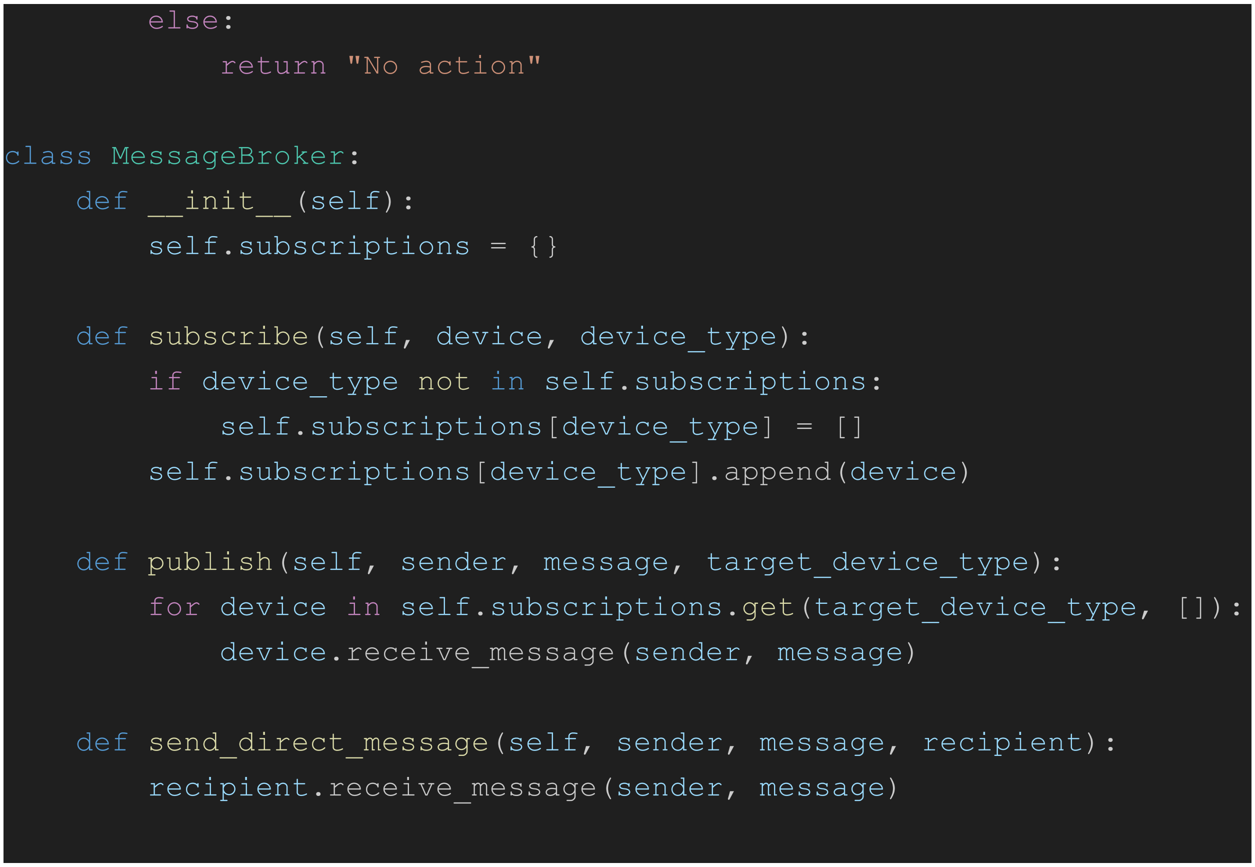
```
import simpy
import networkx as nx
import random
from devices import Sensor, Actuator, Controller, MessageBroker

def sensor_process(env, sensor, broker):
    while True:
        data = sensor.generate_data()
        sensor.send_message(broker, (sensor.sensor_type, data),
"Controller")
        yield env.timeout(10)

def main():
    env = simpy.Environment()
    broker = MessageBroker()

    sensors = [Sensor(i, random.choice(['temperature', 'humidity',
'motion'])) for i in range(1, 11)]
    actuators = [Actuator(i, random.choice(['light', 'lock',
'heater'])) for i in range(11, 16)]
    controllers = [Controller(i, actuators, broker) for i in
range(16, 19)]

    for device in sensors + actuators + controllers:
        broker.subscribe(device, device.device_type)

    network = nx.Graph()
    for device in sensors + actuators + controllers:
        network.add_node(device.device_id, type=device.device_type)


    for sensor in sensors:
        env.process(sensor_process(env, sensor, broker))

    env.run(until=100)

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

devices.py

```
import random

class IoTDevice:
    def __init__(self, device_id, device_type):
        self.device_id = device_id
        self.device_type = device_type

    def send_message(self, broker, message, target_device_type):
        broker.publish(self, message, target_device_type)

    def receive_message(self, sender, message):
        print(f"{self.device_type} {self.device_id} received message from {sender.device_type} {sender.device_id}: {message}")

class Sensor(IoTDevice):
    def __init__(self, device_id, sensor_type):
        super().__init__(device_id, "Sensor")
        self.sensor_type = sensor_type

    def generate_data(self):
        if self.sensor_type == 'temperature':
            return f"Temperature: {random.randint(15, 30)}°C"
        elif self.sensor_type == 'humidity':
```



```
            return f"Humidity: {random.randint(30, 90)}%"
        elif self.sensor_type == 'motion':
            return f"Motion detected: {random.choice([True, False])}"
        else:
            return "Unknown data"

class Actuator(IoTDevice):
    def __init__(self, device_id, actuator_type):
        super().__init__(device_id, "Actuator")
        self.actuator_type = actuator_type

    def perform_action(self, command):
        print(f"Actuator {self.actuator_type} {self.device_id}
performing action: {command}")

class Controller(IoTDevice):
    def __init__(self, device_id, actuators, broker):
        super().__init__(device_id, "Controller")
        self.actuators = actuators
        self.broker = broker

    def receive_message(self, sender, message):
        super().receive_message(sender, message)
        sensor_type, data = message
        action = self.process_data(sensor_type, data)
        for actuator in self.actuators:
            self.broker.send_direct_message(self, action, actuator)

    def process_data(self, sensor_type, data):
        if sensor_type == 'temperature':
            temperature_value =
int(data.split(":")[1].strip().strip("°C"))
            if temperature_value > 25:
                return "Turn on AC"
        elif sensor_type == 'motion':
            motion_detected = data.split(":")[1].strip()
            if motion_detected == "True":
                return "Turn on lights"
```



```python
        else:
            return "No action"

class MessageBroker:
    def __init__(self):
        self.subscriptions = {}

    def subscribe(self, device, device_type):
        if device_type not in self.subscriptions:
            self.subscriptions[device_type] = []
        self.subscriptions[device_type].append(device)

    def publish(self, sender, message, target_device_type):
        for device in self.subscriptions.get(target_device_type, []):
            device.receive_message(sender, message)

    def send_direct_message(self, sender, message, recipient):
        recipient.receive_message(sender, message)
```

Output

```
Controller 18 received message from Sensor 7: ('motion', 'Motion detected: True')
Actuator 11 received message from Controller 18: Turn on lights
Actuator 12 received message from Controller 18: Turn on lights
Actuator 13 received message from Controller 18: Turn on lights
Actuator 14 received message from Controller 18: Turn on lights
Actuator 15 received message from Controller 18: Turn on lights
Controller 16 received message from Sensor 8: ('temperature', 'Temperature: 24°C')
Actuator 11 received message from Controller 16: None
Actuator 12 received message from Controller 16: None
Actuator 13 received message from Controller 16: None
Actuator 14 received message from Controller 16: None
Actuator 15 received message from Controller 16: None
Controller 18 received message from Sensor 8: ('temperature', 'Temperature: 28°C')
Actuator 11 received message from Controller 18: Turn on AC
Actuator 12 received message from Controller 18: Turn on AC
Actuator 13 received message from Controller 18: Turn on AC
Actuator 14 received message from Controller 18: Turn on AC
Actuator 15 received message from Controller 18: Turn on AC
```