Describing IoT Scenarios formally:

This document talks about describing some decentralized IoT use cases in a formal approach. These use cases are listed in increasing order of their complexity (based on number of devices, frequency of communications). We attempt to define the components, actions, triggers and database design for each of the following IoT Scenarios:

1. Thermostat [room sensor,thermostat]
2. Smart Garage [garage door,light,AC heating]
3. Parking Lot [Ticket dispenser,Ticket exit machine,parking sensor]
4. Room Security System [Bed Occupancy,Light,Main Security]

SCENARIO #1:

Smart Garage : This use case is based on a smart garage based in a home. There are total 3 devices in the system :

1. Garage door
2. Light
3. Thermostat

Each of them performs some action based on change in state of another.

Actions: The possible set of actions that could take place in the system would be:

1. Open garage door
2. Turn On Light
3. Turn On AC
4. Close the door
5. Turn off Light
6. Turn off AC

Triggers: The triggers that govern these actions would be :

1. Opening Garage door
    a. Turn on light
    b. Turn on AC
2. Close Garage door
    a. Turn off light
    b. Turn off AC
3. Turn on light
    a. Turn on AC
4. Turn off light
    a. Turn off AC

Storage (Database design) :

1. Garage Door:
    a. Status bit
    b. Trigger information
2. Light
    a. Status bit
    b. Trigger information
3. AC
    a. Status bit

Each device would also contain list of triggers :
ActionID,RemoteDeviceID,ActionTobeTakenID

Communication:

1. Transfer on any of the state change to its trigger.

Computation:

All the computations are happening on device level itself

Mode of communication:

Devices communicate via exposed API over HTTP

Pseudo Code:

Garage Door:

```
# get the last status
(Select lotnumber,status,timestamp
from sensorDB) laststate

# compare with current state
IF currstatus != laststate AND currentTimestamp > laststate.timestamp{
        UPDATE sensorDB SET status = currstatus, timestamp = currentTimestamp
        SEND_TO_MANGER(lotnumber,staus,timestamp)
}
```

NOTE: The pseudo code for the light sensor and the AC remains the same as all they are doing is reporting the status change to the manager. The event/trigger chaining (if there exists any) is taken care of by the manager.

SCENARIO #2:

Smart Bed : This use case is based on a smart bed frame based in a home. There are total 3 devices in the system :
Bed Frame

Light
Security System
Each of them performs some action based on change in state of another.

Actions: The possible set of actions that could take place in the system would be:
Turn Off Light
Turn On Security System
Turn on Light
Turn off Security System

Triggers: The triggers that govern these actions would be :
Human lying on the bed frame.
Turn off light
Turn on Security System
Human no longer lying on the bed.
Turn on light
Turn off Security System

Storage (Database design) :
Bed Frame:
Status bit
Trigger information
Light:
Status bit
Trigger information
Security System:
Status bit
Each device would also contain list of triggers :
ActionID,RemoteDeviceID,ActionTobeTakenID

Communication:
Transfer on any of the state change to its trigger.

Computation:
All the computations are happening on device level itself
Mode of communication:
Devices communicate via exposed API over HTTP
Trigger Information :
To be stored on the device database. Information of what the device should incase the event causes the trigger routine to execute.

Pseudo Code:

BedFrameSensor:

```
# get the last status
(Select lotnumber,status,timestamp
from sensorDB) laststate

# compare with current state
IF currstatus != laststate AND currentTimestamp > laststate.timestamp{
        UPDATE sensorDB SET status = currstatus, timestamp = currentTimestamp
        SEND_TO_MANGER(lotnumber,staus,timestamp)
}
```

NOTE: The pseudo code for the light sensor and the security system remains the same as all they are doing is reporting the status change to the manager. The event/trigger chaining (if there exists any) is taken care of by the manager.

SCENARIO #3:

Automated Parking Lot Ticketing: This use case talks about creating an automated system for parking lot availability. The type of sensors in the system are:

1. Sensor at entry barricade
2. 1 sensor at each available parking spot
3. Sensor at exit barricade

Using the info from all these 3 types of sensors we can create an automated parking lot system which also informs users about the exact location to park the vehicle and save time in looking for an empty system.

Actions:

1. Entry of a vehicle at an entry barricade
2. Exit of vehicle at an exit barricade
3. Removing a vehicle from a parking spot
4. Parking a vehicle at a parking spot

Triggers:

The triggers that govern these actions or change the status of these devices are as follows:

1. Vehicle entering in a parking lot
2. Vehicle exiting a parking lot
3. Vehicle moving position from one spot to another

Storage (Database design) : The storage at each device is as follows:

1. **Entry Barricade**: The entry barricade would maintain 2 types of information : one for the status of itself and other maintaining the status bits of all the other available parking lots.

Status Information:
Vehicle Entering [boolean]
Timestamp [timestamp]
ParkingLotAssigned [int]

Available Parking Lots:
ParkingLotNumber [int]
ParkingLotAvailability [boolean]
Timestamp [timestamp]

2. **Exit Barricade :**
   VehicleExiting [boolean]
   Timestamp [timestamp]
3. **Parking lot sensor:**
   ParkingLotNumber [int]
   IsOccupied [boolean]
   Timestamp [timestamp]

Communication: The local manager for all of these would be the entry parking lot barricade. Any changes to the change in status are communicated to the local manager.

Computation:

All the computations are happening on device level itself

Mode of communication:

Devices communicate via exposed API over HTTP

Trigger Information :

To be stored on the device database. Information of what the device should incase the event causes the trigger routine to execute

Pseudo Code:

ParkingLotSensor:

```
# get the last status
(Select lotnumber,status,timestamp
from sensorDB) laststate

# compare with current state
IF currstatus != laststate AND currentTimestamp > laststate.timestamp{
        UPDATE sensorDB SET status = currstatus, timestamp = currentTimestamp
        SEND_TO_MANGER(lotnumber,staus,timestamp)
}
```

ExitBarricade:

```
# get the last timestamp from db for the vehicle eiting
Select max(timestamp) as lastTimestamp
From exitSensorDB
Where vehicleexiting = True

# compare with the current status

IF currVehicleExiting AND currTimestamp > lastTimestamp
        INSERT INTO exitSensorDB (TRUE,currTimestamp)

        SEND_TO_MANAGER(vehicleExit,currTimestamp)
```

EntryBarricade:

```
# case of vehicle entering the barricade
# update the driver with the available parking slot
Select min(parkingLotNumber) as lotAssigned
From avlParkingLots
Where status = TRUE

INSERT into entryInfoDB (TRUE,lotAssigned,timestampt)

# case of receiving from the any of the sensors
UPDATE avlParkingLots VALUES( lotNumber,status,timestamp)
```

The pseudo code also takes care of a scenario where a vehicle was within a parking lot and moved from one spot to another.

SCENARIO #4:
Smart Thermostat: This scenario talks about an automated thermostat in a room. The types of sensors/devices in the system are :

1. Temperature sensors
2. Thermostat

Using the info from these 2 types of devices we can make regulate the temperature of the room to a pre-set value inputted from a user.

Actions:

1. Turn on the heating
2. Send data to manager
3. Turn off the heating

Triggers:

1. Difference in values of current temperature and value to achieve

Database Design(Storage):

TemperatureSensor:

a. Temperature [float]
b. Timestamp [timestamp]
c. SensorNumber [int]

Thermostat:

The thermostat would maintain data of 2 different types: One received from different sensors, the other one maintaining the temperature values to be maintained in the system

TemperatueSensors:

a. SensorNumber [int]
b. Timestamp [timestamp]
c. lastTemperature [float]

Thermostat:

a. TemperatureToSet [float]
b. CurrentTemperature [float]
c. Timestamp [timestamp]

Pseudo Code:

Temperature Sensor:

```
# get the last temperature value of the sensor
Select SensorNumber,timestamp,lastTemperature
From sensorDB

# check with the current temperature
If currtemp > lastTemperature and currtimestamp > timestamp:
        UPDATE set lastTemperature = currtemp, timestamp = currtimestamp
        SEND_TO_MANGER(sensornumber,lastTemperature,timestamp)
```

Thermostat:

```
        # get the average of all the values from the sensors data
```

```
Select avg(lastTemperature) as roomTemp
From Temperature Sensors

If roomTemp < tempToSet:
        TURN_ON_Thermostat()
Else:
        TURN_OFF_Thermostat()
```

Observations/Questions:

1. In case of manual intervention, should it be able to override the default triggers?
2. Should have chaining of triggers or these should be placed separately?
3. Should the devices be configured to send only for a threshold of change? (Example: Report the temperature changes only when the *delta* is beyond a certain threshold)
4.