# COMPSCI 677 Lab 2 Design Document

**Team Members**
Chandra Sekhar Mummidi
Sai Venkata Maneesh Tipirineni

### Overview

The overall design proceeds in three phases: 1. Database Initialization and Device Registration, 2. Leader Election,Physical/Logical Clock Initialization and Clock Synchronization, 3. Event Ordering and Security System Monitoring. We have 2 server processes, server_gateway which interacts with the sensors and the devices and makes decisions, and a server_backend process which creates and maintains the device,sensor and event databases. We have 4 sensors, temperature, motion, door and presence. And the light bulb and outlet devices. Thread Synchronization was achieved through Python GIL.

### Server_Gateway

This is the application tier which interacts with sensors,devices, database and makes security decisions. It also registers the devices and gives them unique device id's and also polls temperature and door sensors regularly. It also starts the leader election process and initializes the vector clocks when all the sensors and devices are registered. It also receives state_change messages from presence, door and motion sensors and handles multicasting of logical vectors. We considered only change_state calls as server_gateway events and hence no other calls would increment the server_gateway logical clock. Polling devices does not start until at least one state change occurs and it is not considered a server process.

### Server_Backend

This is the tier which manages the database. We have 3 database files. One is devices.txt which contains information about the devices and sensors. The others are events.txt which is a log of all the events that occurred and latest_events.txt which contains information about the latest events, the ones required to make security decisions. Splitting the events database into two files helps us differentiate the older events from the latest ones and determine if we need to wait for a new event to happen to determine if the "user left home" or "user entered home". In both those cases, we need both doorsensor data and motionsensor data and the ordering between them. Suppose the user left home and now is coming back and we do not have a latest_event database, we see old motionsensor data in the database from when the user was home earlier and doorsensor trigger when user opens the door and move state to "user entered home". This is wrong as the user just opened the door and is not home yet, we still need to wait for the motionsensor trigger before declaring that the user is home. Having a latest_events database and clearing it after a state change makes it easier to accomplish this. Every write to a database file is counted as an event of server_backend. Note that polling will be considered as an event at the backend as the state is written to events.txt

The schema of our tables is as follows:
Devices.txt: Device_unique_id, Device_type, Device_name (seperated by tabs)
Events.txt: Device_id, Event_description, Device_physical_time, Device_logical_time
Latest_events.txt: Device_id, Event_description, Device_physical_time, Device_logical_time

**Devices and Sensors**

Devices register themselves with the server_gateway when they come alive. Their names and ids are stored both in devices.txt and also in-memory for faster access and decreasing network traffic. Each device also knows the name of the leader once leader election is completed. Each device also gets the list of devices during the lead_election process.
4 sensors were implemented with their respective push/pull/change functionalities.

**Leader Election**

Leader Election is done using bully algorithm. It is started by server_gateway process when all devices and sensors are registered. All devices, sensors, server_gateway and server_backend participate in leader election. This is done via the leader_elect.py interface.

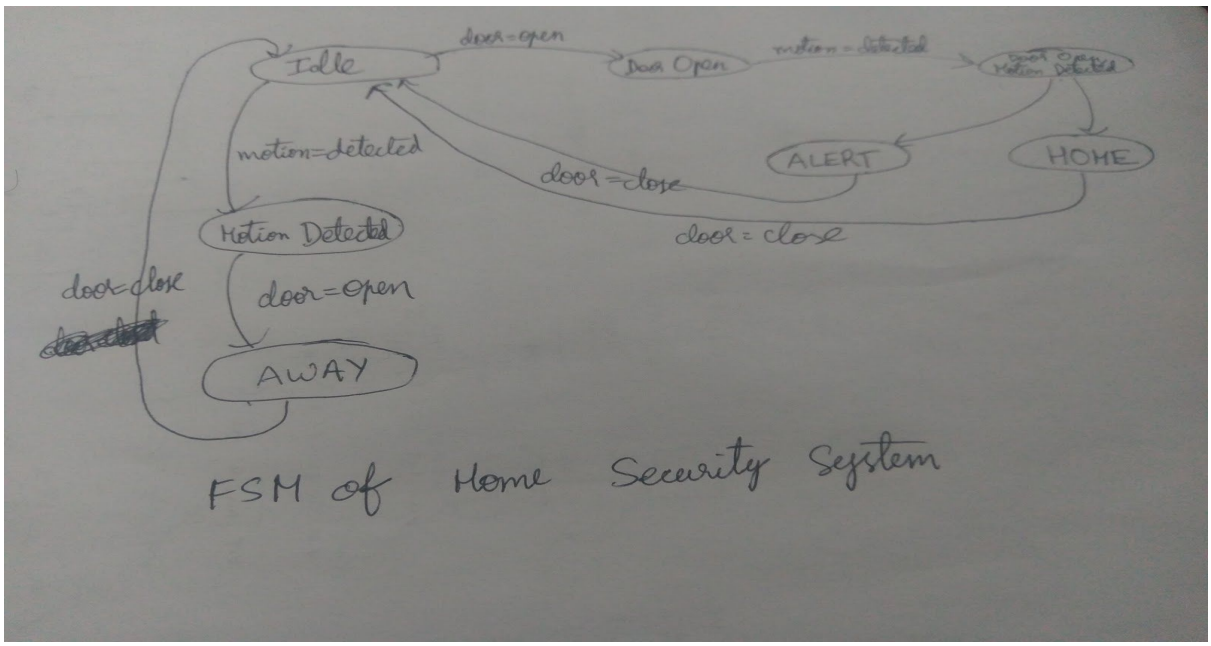
**Physical Clock Synchronization**

Physical clock synchronization starts right after a leader is elected. The leader starts the clock synchronization process. The average offset is calculated and sent to all processes. The synchronization process is run once every 8 seconds. The networks delays are accounted for taking the average roundtrip time of messages to all processes.

**Logical Clock**

We used vector clocks and causally ordered multicast to determine an ordering of events. Server_gateway handles multicasting. It updates its clock whenever it receives a state_change notification (the notification contains the vector clock of the sending process), and multicasts the new vector clock to all other processes.

**Security System Algorithm and Event Ordering**



**Base State Diagram of our Security System algorithm**

We are using the above state diagram and physical/logical times from latest_events.txt to build our home security system algorithm and determine whether the user is Home, Away or ALERT in case of a burglar entering the house. The order in which the transitions occur in the above figure is determined by event ordering.
We used numbers for states in the implementation of our algorithm. The following table maps those state numbers in our implementation to the above states.

| | |
|---|---|
| 0 | Idle |
| 1 | Door open |
| 2 | Door open and motion detected |
| 3 | User Home(presence key detected) |
| 4 | Intruder Alert(presence key not detected) |
| 5 | Motion detected |
| 6 | Door open (user away) |

While implementing event ordering in code, after every message received, server gateway checks for recent event for each sensor, door sensor and motion sensor which are push based.
Synchronized physical time is retrieved for each event from database. For example, physical times for motion detection and door opening are used to make decision about whether user entering into home or leaving home. This data is retrieved from latest entries of each sensor. If an event is used to make decision then that event is removed from latest event entries and if server doesn't have enough data to make a decision it will wait for message to reach it.This is seen in code as the state machine jumping from state 0 to either 2 or 6 that is both motion and door sensor data depending on the order they occur.
If FSM is in door and motion detected in the same order, it should receive presence status from user key(as push mentioned in part 3) within a timespan. For our test case we have taken it as 5 seconds. That is, if server doesn't receive a presence beacon in 5 seconds after door opened it will consider it as an intruder. Otherwise if it is received then server considers it as user reached home.
If FSM in motion and door detected, in the same order then it will be considered as user leaving home.
If the user comes home, both the light bulb and the smart outlet are switched on. If the user leaves, both are switched off. In the event of an intruder alert, the lights are switched on but the smart outlet is switched off.

**User Process**

A user process is used as the driver for the whole system. The user can change the temperature and trigger and change the states of various sensors and devices through this process. The user also sets his preferred clock choice through this program.

## Design Tradeoffs

1. Only one instance of each process can be run.
2. We assumed that the door is closed by the user while leaving or entering home.
3. Vector Clocks are assigned after leader election and therefore no event can occur before leader election.
4. No processes can be added after leader election starts.
5. Number of processes need to be known beforehand.

## Further Improvements

One obvious improvement would be to allow two or more instances of the same process run simultaneously. Another improvement could be adding support for dynamically adding and removing sensors and devices during runtime and handling failure to nodes.