Eric Cao, Kierstin Matsuda, Kevin Garcia
4/15/2019
COP 5614
Group 9 Final Project

Prototype Java Implementation of a Smart Home Management System

The problem is that homes take a lot of work to function properly. For example, every time you turn on a light you need to turn it back off; however, with our smart home management system it will be possible for the light to automatically turn itself on or off based on your preferences. As a proof of concept, our smart home management system prototype will feature a wide range of applications/features such as lighting, security, energy saving, and anything else the user can come up with. For example, the smart home system will have a security setting. This setting will change some devices based on its status. The security status switches between “home” and “away”; if the security status is set to “away” and the user has a camera doorbell registered, it can show the user who is at their door. Whenever the system is set to “away”, it will notify users if any motion sensors have gone off inside the house. All of these features will be customizable and be capable of running autonomously or manually through the interface.

Smart homes connect devices, sensors, and users with each other through the “gateway”, a server that allows data flow among said things. This allows the user to handle all devices in one area as well as certain actions to be performed autonomously. The server can be accessed wirelessly so it can receive commands from users not at home. An example of this function is turning on the lights while away from home.

The system’s server has two tiers: the front tier and the back tier. The front tier server will handle all the queries and commands from the user and will store any changes into the back tier database server. The back tier server will also keep track of all attached sensor and device state changes. The database will keep the prior changes in a text file, with the title of each file being the time it was created. We used the bully algorithm to select the leader for clock synchronization. For clock synchronization, we used the Berkeley algorithm. We also used logical clocks, specifically Lamport clocks. Each node has a clock offset variable called timeOffset, which holds how much each clock has to be adjusted based on the Berkeley algorithm in seconds. The actual clocks are not changed, but the offset is used to change their timestamps.

By having this clock synchronization, we can use this for event ordering. Now when the door is sensed to be open followed by motion sensed in the room, the smart home system will tell the user that the user has entered the house, the alarm system is set to HOME, and the lights will turn on due to motion sensed. On the other hand, if motion was sensed first then the door is opened, then the user has left the house and the alarm system is set to AWAY. Additionally, we will also use the beacon sensor to differentiate this event between the user and a possible intruder; if the beacon is not present when the following event occurs, then the smart home system will notify the user that there is an intruder at home and the lights will not turn on.

This system works by taking advantage of Java’s ability to use multithreading, so we were able to use threads that independently handled each client on the server. This allowed

each thread of the server to communicate with one another using shared memory. The program also uses sockets to communicate with each respective sensor, device, and user.

Users will be able to change states and query periods of individual devices/sensors. The system will have deadlock handling. The management system will keep a database recording all information about each device registered with the system. The database keeps track of every event that happens. The server will keep a log detailing each time a device has changed state, data transfers, connection registration, connection interruption, or when an error occurs. The log will have a built-in backup function. The Smart Home Management System will be written in Java and will use Apache POI for database and log functionality. The database is also saved on an Excel sheet. The user interface of the prototype is written on Netbeans. The user can also create a new device and sensor. The user can name the device and sensor and the true value and false value for devices only.

For testing, we checked every command and checked their output to see if it works. For example, if we query a command, the prototype will tell you the device's or sensor's status. We also checked that the database remember all past commands and events. This was done by entering in commands at a specific time followed by searching the database during that time. We also tested the prototype on both Mac and Windows.

After working on the prototype, we thought of many improvements for the next version. One problem we would like to fix in the future is that the thread created by the server when a user connects will not terminate if the user process abnormally exits. Two possible improvements are making the user know when the server has stopped running and vice versa. This will prevent any hanging executions from continuing to run. That way, the user can keep track of any problems or bugs. In the prototype, we have the motion sensor linked with the smart bulb as an example, but in the future we would like to give the user the ability to link devices with sensors. Another feature we would like to implement would be letting the user disconnect or restart devices and sensors, but not removed from the system. With the prototype on a computer, in the future we would like to implement it into a mobile app.

In order to run the program, you first need to make sure that all programs have the same port and each client has the IP address of the server. The IP address and port of the server can be changed in all client programs by modifying the "SERVER_IP" and "PORT" constants at the top of each class. Also make sure that java and make are installed. As well as making sure that the server is accepting incoming and outgoing requests on that port and IP. This would ensure that the server can receive and send information to the clients. Once that is completed, run the command "make" to compile all the files. Then run the files by using the command "java <Filename>". For example, to run Outlet.java, you need to enter the command "java Outlet". Run the server file first, then run all the clients. Each client can be run on any computer or directory. Next, run the UI file by typing "javac FinalProjectUI.java" and "java FinalProjectUI". Once all programs are running, you should see the interface with options the user can choose, and occasional messages from the server to the user program. The messages will differ based on what options were chosen; for example, you should not see a message about an intruder if the security mode it set to HOME. Once you are done with the program, simply close the window to exit. This will stop the interface, but the other programs will continue running. To stop the other programs from running press Control-C.

With the prototype able to run basic concepts, the Smart Home Management System will help users control all their household appliances with one application.