ECE/CSC 792/791/591: Internet of Things: Architectures, Applications, and Implementation

---

Report and Analysis
04/27/2022

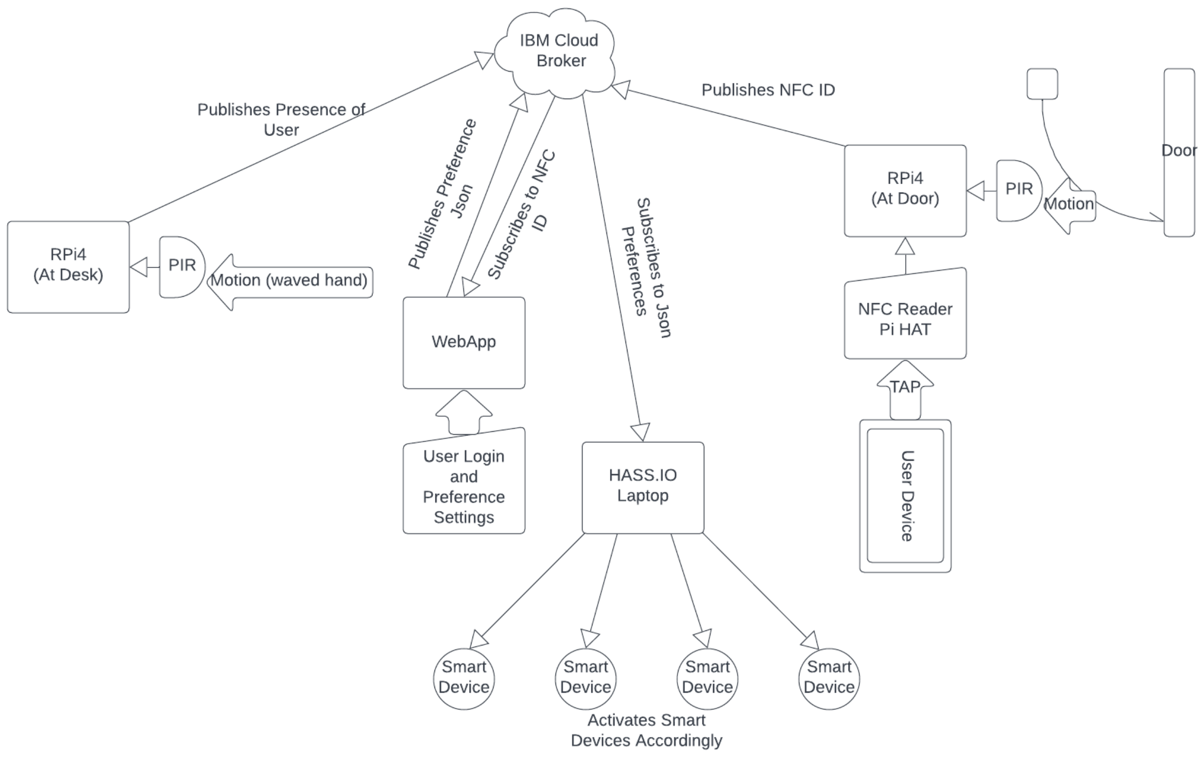
Xander Carruth
Arnab Datta
Timothy Boushell

Introduction:

Smart homes are becoming more and more common as a way of providing increased comfort and functionality in a home. However, many smart homes still do not take into account the needs of individual users, opting to rather have the whole house set to the same settings on one central user account. The ability to deliver smart home settings based on the specific user that is in a room would further increase the comfort and functionality that smart homes provide. Thus, the aim of our project is to provide a smart home interface that automatically caters device settings based on user preferences. This works through the user setting their preferences in a web app and the application connecting their set preferences to the devices in the smart home via MQTT whenever the user scans the NFC tag linked to their account. This would enable rooms to automatically update whenever a user walks into a room instead of having to manually change device settings in an app.

Design:

Figure 1: Block Diagram



Raspberry Pi 1 is attached to a NFC Reader and a PIR Sensor. The PIR sensor senses when the front door of the house has been opened and ‘wakes up’ the Raspberry Pi which waits for an NFC signal. The user scans an NFC tag against the reader (you could keep the tag in the back of your phone case or attached to your key ring) and this action pushes the NFC tag ID to the web application running on IBM Cloud. The web application compares the NFC ID against its database and retrieves the user profile which is linked to the NFC ID. This user profile has a setting for 4 appliances (whether they should be turned on or off). This number could be increased easily, and we have chosen four for the purpose of demonstrating this project. It publishes the status of the four appliances (on or off) to the **preferences** topic on an MQTT broker running on the IBM Cloud.

Now we have **Raspberry Pi 2** which is located on the user's desk in his room. It is connected to a PIR sensor. This allows certain appliances to turn on in the user's room by detecting 'when he is sitting at his desk'. If he is sitting at his desk, these appliances will be on or off depending on the preferences stored in the web application. If he is not sitting at his desk, these appliances will be turned off. Raspberry Pi 2 detects whether the user is at his desk or not and publishes that information to the **userDeskTag** topic on the IBM Cloud MQTT Broker.

Now we have **Laptop 1** which is running a MQTT broker locally using **mosquitto**. It is also running a Python script which subscribes to the IBM cloud MQTT broker. It subscribes to the **preferences** and **userDeskTag** topics. Based on the information, it decides which appliances should be turned on or off. The local MQTT broker has one topic for each appliance. This script takes into account the **userDeskTag** information for controlling certain appliances. The python script (iot_project.py) publishes the on/off information to each topic respectively (using paho-mqtt).

We also have a **Home Assistant OS (Hassio**) which is running on **Laptop 1** (we could also run this on a Raspberry Pi). The Home Assistant OS is connected to 4 TP-Link Smart switches which are placed in the house. These switches simulate the different appliances. We could also connect other smart lights/other devices which have integration support from Home Assistant. The Home Assistant hub connects to the local MQTT broker and turns these switches on or off depending on the information published to each topic.

Finally we have the web application running on the IBM Cloud which allows us to set up new users (create a profile, pair it with an NFC ID and store preferences) and update existing user preferences if required. The web application is built using Flask with a React Bootstrap front end and a PostgreSQL database to store user preferences and the NFC ID. It is subscribed to the **userDeskTag** topic to receive the NFC ID when it is published. When a new NFC ID is published, it is held for 60 seconds, during which the NFC ID can be used to pair to a user until it is erased. When an NFC ID that has already been saved to a user is sent again, the web application publishes the user preferences associated with the NFC ID to the **preferences** topic.

Implementation:

PIR.py - interfacing with PIR sensor, reading NFC, publishing to broker
DeskPIR.py - sending status is user is present at desk or not
publisher.yaml - used for letting DeskPIR.py connect to IBM Cloud

Iot_project.py - subscribes to preferences and user1DeskTag topics on IBM Cloud and publishes appliance on/off commands of 4 separate topics (app1, app2, app3 ,app4)
subscriber.yaml - used for letting iot_project.py connect to IBM Cloud

Web app: Ran in containers using docker-compose, has a Flask app backing a React Bootstrap front end and as PostgreSQL database
docker-compose.yml - builds the flask and postgresql containers
Dockerfile - helps run the commands to build the web app
requirements.txt - holds the pip install requirements for app
__init__.py - initializes the app
app.py - holds all the main functions of the app (including MQTT subscribe and publish)
config.py - holds the configuration for the app to initialize on
forms.py - sets up all of the forms used in the app
models.py - sets up the User and Preferences objects for the database
webApp.yaml - used for letting the web app connect to the MQTT broker on IBM Cloud
login.html - login page for the web app
settings.html - settings for the web app and where the NFC ID is paired
signup.html - sign up page for Users
static.html - the landing page for the website that also allows users to change their preferences once they have set an NFC ID
style.css - styles all of the web pages on the web app

Results and Discussion:

Through this project, we were able to demonstrate the feasibility of our idea for using an NFC ID reader to associate a user with their preferences for a room in a smart home and update their room's devices accordingly. We faced a few key challenges in this project that we had to work around in order to reach our end goal. We initially had planned for the scanner to read the NFC ID from the user's phone. However, after realizing that the phone NFC ID was changed every time it was read and encrypted with an undisclosed algorithm, we opted to instead use a NFC tag which could be slipped into the case of a user's phone and would send out the same NFC ID on every scan. We also originally wanted to control all of our devices through Google Home. After looking into the Google Home API, we found that we would not be able to edit preferences from the web app so all of the Google Home devices would have to be controlled by playing audio from a speaker. We found that this was not the optimal solution so we decided to use hass.io to control all of the devices instead. We also had wanted to integrate an Amazon Echo instead of Google Home as well but we decided that this was out of the scope of the project after we started using hass.io. For the web app, there were initial issues creating the page that allowed forms to be dynamically updated with the NFC ID after pairing. However, removing the form altogether and combining an AJAX POST with a GET from within the Flask app allowed the NFC ID to be paired and displayed to the user.

Despite the aforementioned challenges, we were still able to produce results that met our goals for the project. Future work with this project would include finding a way to standardize the NFC IDs sent from phones, adding in a smart home device for increased functionality, and more features such as analytics hosted in the web app. These additional improvements upon our work thus far would greatly enhance the comfort and functionality that smart homes provide and cater the smart home to individual users.