HomeTweetie: The Smart Home Twitter Application

IN4398 - Internet of Things Seminar

Delft University of Technology
June 22, 2016

Group Members:

Muneeb Yousaf
4411129
m.m.yousaf@student.tudelft.nl

Christopher McGirr
4415302
c.a.mcgirr-1@student.tudelft.nl

Abstract

The implementation of a light weight and scalable system for a Smart Home Gateway that utilizes the Twitter platform to communicate with the Home Owner has shown to be a success. It has even been shown using a simple Stochastic Markov Model that the needs of the Home Owner can be inferred to allow the Home to issue a command based on the inference of meaning from the Home Owner's previous tweets. This also allowed for a simple system for commands to be given to the Home gateway in natural speech. Although, many aspects of the HomeTweetie application lack the sophistication and complexity to make this a robust and usable system in everyday life. It does allow for future expansion and improvement to yield a more concrete application.

1 Introduction

The use of sensors and smart devices in our modern homes is growing every day. This opens the possibility of utilizing these sensors and devices to make the life of the home owner that much easier and simpler.

However, one of the many hurdles to implementing such a system is the ability for the sensors and devices to communicate effectively over the Internet with the home owner. This of course could be done using proprietary communication platforms and servers to send messages bi-directionally between sensors, devices and the user.

To implement a platform where home smart devices and sensors can communicate with the user over the Internet can incur high capital costs for renting and maintaining servers. However, there already exists a number of platforms where small messages can be sent around the world with high reliability. One such system is the Twitter platform.

The idea behind HomeTweetie is to utilize this communication platform to create a deploy-able and lightweight system where home sensors and smart devices can relay information and receive commands from the home user, who may not necessarily be at home. The systems would have an Internet gateway, in the home which would receive and send all the tweets to the outside world. The home user would be able to send commands over the twitter platform to the home and the home devices would react accordingly. Or the home user would receive notifications on the state of the home.

The gateway device can even go further and analyze the home owner's twitter feed and respond to indirect commands. In a way infer from the home owner's tweets through natural language interpretation what the home owner would want. For example, if the recent history of the tweets suggest the home owner finds it cold outside and he is planning to go home, then the home could preemptively switch on the heating for the homeowner. Without the home owner ever asking for it.

This idea can be further extended to take in the home's environment based on geo-location of other twitter feeds. If the home were to monitor tweets that originated from around the home, a form of crowd sensing, it could react to local events such as a robbery by making sure all doors are locked and lights are on if it is the night. To give the appearance the home owner is home.

2 Twitter API

The Twitter API allows for developers to use many of the features found with the use of Twitter. However, there are some limitations that have to be taken into account when using the API. Due to the open nature of the Twitter API to the public, the amount of "GET" requests that can be sent in a time interval is limited. For the use of the REST API, which includes the Twitter feeds of users and Direct Messages of users, it is limited to 15 requests every 15 minutes[6]. Therefore, the access limit has to be taken into account when designing the application. As HomeTweetie will only be able to grab Twitter Feeds and Direct Messages of users once every minute which reduces the ability for the application to respond in real time.

In order to use the API given by Twitter, an application has to be registered [1]. However, this is not the only step. In order for the HomeTweetie application to access the Direct Messages of its account and the Home Owner's twitter account the application settings had to be modified to allow for read and write

[1] Visit https://apps.twitter.com/ to register an application

of direct messages. With the application keys and settings updated we could now begin to program the HomeTweetie application.

3 Software and Hardware Platform

For this project the Java Programming language was chosen due to the flexibility to be ported to any machine that can run a Java Virtual Machine (JVM). Another reason for the use of java was the ability to easily prototype an application and the variety of libraries available.

However, one of the downsides of using Java is the fact that the memory management of the application is down automatically and in most cases it is not optimal. This can lead to an application with large overheads, but due to the amount of data that the application will be processing and passing, it was decided that a Java implementation will be sufficient.

One of the libraries used is the Twitter4J [5] which is a wrapper library for the Twitter API that allows for easy use of the Twitter API. Rather than coding for functions to parse the JSON objects that are returned by HTTP requests ourselves this API handles all the back-end processing and outputs easy to use Java objects.

For the hardware platform the first choice was the BeagleBone Black which features a 1GHz ARM processor with 512MB of RAM with USB ports, Ethernet Adapter and a variety of GPIO Pins [3]. This allows for Internet connectivity as well with connectivity to sensors. However, using the BeagleBone Black proved to be more time consuming than thought. Though there is a vast pool of documentation and tutorials on-line, the Java Application and connection to the Internet could not be established using the WiFi adapter. Therefore a decision was taken to move application from the BeagleBone Black to the more user friendly, though less powerful Raspberry Pi B.

The Raspberry Pi B is equipped with a single core 700MHz ARM processor with 512MB RAM with 2 USB ports, Ethernet Adapter and a 40 Pin GPIO Port[4]. Again the Raspberry Pi has less computational power than the BeagleBone Black, but with the vast support for applications and linux distributions for the Raspberry Pi allows for quick and simple setup of the application. In this project the Raspbian Linux Distribution was used to run the Java Application which comes with pre-installed versions of Java and driver support for many Wifi USB dongles.

4 Direct Command Communication

The first step in the HomeTweetie Application was to implement communication with Twitter between the Home Owner's Twitter account and the Twitter account of HomeTweetie. This was done by implementing a form of direct commands which the Home Owner can give to HomeTweetie. Given a predetermined format, the application will be able to determine what the Home Owner wants done and executes that task.

The private user chat between HomeTweetie and the Home Owner's Twitter account was used to send commands. The messages sent in this private chat are otherwise known as Direct Messages. We chose this as the private chats between two users are not published publicly as compared to conversations between users using the Twitter feed with the use of "@" replies.

```
HomeTweetie [Action] [Object] [Settings]
```

Above you can see the format of the direct command. Here the homeowner can send a message starting with the word HomeTweetie followed by an action such as "set", then an object such as "lights" with a setting of "on" or "off". To make this easy to use and expandable, the application was created using the Command Design Pattern.

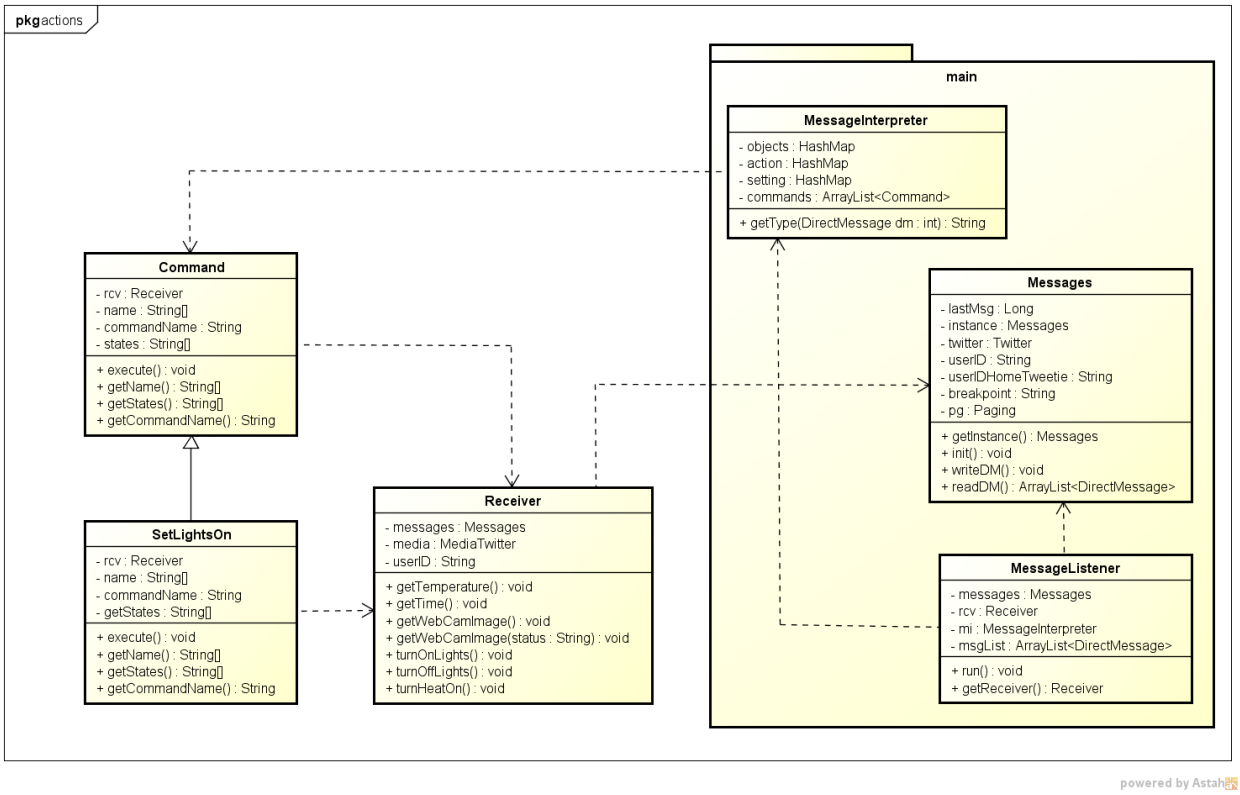

Figure 1: Class Diagram for Direct Command Communication

In Figure 1 the class structure of the Direct Communication Implementation can be seen. The commands available to HomeTweetie are sub classes of the parent class Command so that each command that inherents the parent class has the same format. This makes it simple to execute a command within the MessageInterpreter class since all objects which extend the class Command will have an execute function.

The Receiver Class thus contains all the functionality of HomeTweetie. This is where all the Command objects call when executing their task. Inside the Receiver class there are a number of functions that implements a certain feature of the Home. For example turnLightsOn() which will be called by the Command object setLightsOn() when its execute function is called.

Now another feature of the Command Class is that the array String[] Name contains the set of words that are associated with the direct command in the format Name = Object, Action, Setting. For example SetLightsOn has an array name = "lights", "set", "on". This is useful as during the initialization of the program the commands can be dynamically loaded into a tree structure so that when a message is decoded the tree can be searched to determine whether the command is matching. The MessageInterpreter object takes an ArrayList of type Command and constructs a Hash Map or Tree of the commands, their actions and settings from each Command Objects name array.

When the MessageInterpreter function getType() is called it traverses the tree of strings and checks the words of the message against the tree word for word. If all the required words in the message match all nodes of the tree in order and reach the leaf of the tree than the application executes the function from the Command Object associated at the leaf of this branch.

In addition, the application will be running consistently. Therefore, an object MessageListener was created that extended the class "runnable" so that the function can be run as a thread. As mentioned in Section 2 the application can only grab data from Twitter every 60 seconds therefore the function was scheduled with a fixed period of 70 seconds to avoid any issues.

However, this was not enough. The last message read would also need to be recorded to avoid executing the same command in the next period. Therefore, a break point string was used to mark in the private

conversation with the Home Owner where HomeTweetie has read up to. In this case the string used was "HomeTweetie: has read all your messages :)". This enabled the application to mark where in the conversation it has read to and gave the Home Owner notification that it has read the messages. So if the last message with the Home Owner was the break point than there are no new commands to be processed for HomeTweetie.

5 Command Inference from Tweets

5.1 Inferrence from Tweets

With the ability to receive direct commands from the Home Owner over Twitter we could now begin to implement a system to infer the Home Owner's needs from his or hers Twitter feed. For this portion of the project only a single command was selected to be implemented. The command chosen was the turn on the heating command. Due to the fact that there is limited time not all features or commands were explored.

The idea behind inferring commands from the Home Owner was that the Twitter feed can suggest to HomeTweetie what the Home Owner may want. Now natural language inference can be quite difficult and there are many models that attempt to quantify meanings of human speech. However, for this project we took a simple approach.

Here tweets are taken as a Unigram model [7]. A Unigram model assumes that a probability of a sequence of words can be simplified to the product of all the individual probability of those words.

$$p(w_1 w_2 w_3 ... w_n) \approx \prod_i^n p(w_i)$$

However, we still need to define a probability for a single word. Here the Bag-of-Words model was used to quantify a probability of a word from a given tweet. That is given a word the probability in a document, or in this case a tweet, would be the number of occurrences of this word divided by the total number of words. In order to increase the probabilities of the target words for the given command we take the total number of words as the total number of targeted words occurring in the tweet given the command.

$$p(w_i) = count(w_i)/(\sum_1^n count(w_i))$$

The only part left is to associate a number of words with a command so that the words can be matched to a tweet. Since not all tweets may talk about every aspect concerning a command, the command was further broken into states or topics.

For each command a number of states are required for it to be classified as that command. Take for example the command turnOnHeat. Here the states: weather, home, work, going are associated with this command and are required by it to be considered. Then each state has a number of associated words or synonyms with them. To illustrate how a tweet is converted into a probability for a given command take the fictional tweet below.

The **weather** is pretty **cold** outside. I might decide to cut **work** short and **go home**.

Here we have an probability vector of 4 elements or states for the command turnOnHeat since we defined the command turnOnHeat to have four states or topics. The number of words associated with any of the states in this example are 5. In the application we use a Javascript Object Notation (JSON) object to store the command states and the associated words with that state. The full JSON object for turnOnHeat is given in the Appendix Section 10.2.

The number of words occurring for state "weather" are two: weather and cold. While the number of words occurring for state "home" are one that is home. While for the state "work" the occurring word is work and for the state "going" it is the word go. This translates to the vector:

$$p(turnOnHeat) = (weather, home, work, going) = (2/5, 1/5, 1/5, 1/5)$$

Of course it has to be noted that for different commands we have different sized vectors. However, we notice that the sum of the vector always equals one, making this a stochastic vector. Now that there is a method for creating a probability vector for each tweet in relation to a command, a model can be created to take advantage of the Home Owner's Twitter feed.

Here a Stochastic Markov Chain Model is used to model the past probabilities of the Home Owner's tweets. When the application initializes it takes the last 20 tweets from the Home Owner's twitter feed. Then it converts each tweet to a state vector. Which takes a value of 1 or 0 given if a state occurs or not in the tweet given the state words.

$$State(tweet)_t = (1, 0, 0, 1)$$
$$State(tweet)_{t+1} = (0, 1, 0, 1)$$
$$State(tweet)_{t+2} = (0, 0, 1, 1)$$

At which point the application traverses the tweets in chronological order to see how each tweet relates to another and sees whether there was a change in states. Above we have an example. Here we see a change from the three tweets. That is in $t + 1$ the Home Owner no longer talks about state 0 but now talks about state 1 while maintains the topic of state 3. Given just these three tweets this translates to a Stochastic Transition Matrix of:

$$StateTransition = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 2 \end{bmatrix} \quad where \quad ST_{ij} = ST_{ij} + S_t[i]S_{t+1}[j] \quad P = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0.25 & 0.25 & 0 & 0.5 \end{bmatrix}$$

Now the Probability Transition Matrix is computed by simply dividing each row of the State Transition Matrix by the sum of that row. Now with this matrix we can multiply it with the current tweet to get a probability vector that gives the probability about how likely the next tweet will talk about the either of the four states.

5.2 Leveraging other Sources of Information

The Home Owner may not always be talking about the weather, home, work or about going somewhere in their Twitter feed. This is a drawback of the command inference as it may give a low probability and not reach the threshold to be classified as a valid command. Therefore, other sources of information are used to leverage the results given from the inference to better estimate whether the Home Owner would want for example the heating to be turned on.

The data stored within tweets are plentiful. If the Home Owner allows for geo-location of their tweets, the tweet can contain the geo-location information of where the Home Owner has tweeted. Assuming the Home Owner's home location and work location are given this information can be used to estimate whether the home owner is at home or at work if the tweets do not contain any of the key words that involve work. Hence the inference may give zero probability for that state.

In addition, weather data from the Open Weather Map [1] which given the city of the Home Owner can specify the current temperature. Therefore, if the Home Owner does not talk about the weather and how it is cold, the Home will not be able to infer this. Thus, the weather data is used to determine this with a given threshold, in this case 18 degrees Celsius. Although the temperature in the home may be more relevant, the sensor could not be implemented in time and so the weather data was used as a substitute to show as a proof of concept.

For the command turnOnHeat if both the geoLocation data and weather data show the Home Owner is at work and it is cold outside then the inference probability is checked to see if it is above a certain threshold.

6 HomeTweetie Functionality

Twitter also allows for not just for sending text over the Internet, but media as well. Something of interest often for a Home Owner is that when the owner is not at home, if someone for example visits the door or a pet is alone at home, then the Home Owner would want to be notified of any changes in the home.

That is why a function has been added to the HomeTweetie application that when motion in the home has been detected a image is sent to the Home Owner. The motion sensor used was a standard Passive Infrared (PIR) Sensor. When the sensor detects motion an image is taken from a USB Webcam and posted to the HomeTweetie Twitter feed with the Home Owner tagged.

The PIR Sensor was implemented on the Raspberry Pi using the standard GPIO pins provided. The sensor simply gives a High 3.3V signal when motion is detected and 0V when none is detected. The library used to access the GPIO of the Raspberry Pi was the Pi4J Library [2].

The implementation of the webcam also proved to be a little cumbersome. At first, a Java Library for Webcams was used called "Webcam Capture Api". However, due to bugs present in the Raspberry Pi set of drivers this proved to be incompatible. Therefore, as a work around an external script was written to save the image of the Webcam when motion is detected and to save this image to the file system. After which the HomeTweetie application would read the image from file and post it to the Twitter feed.

Of course this method is very insecure. The use of direct messages would provide for a better implementation in terms of security and privacy for the Home Owner. However, with the current Twitter API, applications cannot send media data over Direct Messages to other Twitter users. Therefore, the image had to be posted to the HomeTweetie's Twitter feed.

To avoid constant posting of images to the feed, a timer was included in the motion detection thread to reset the thread again in 5 minutes so that the number of images posted does not exceed 12 in an hour.

The HomeTweetie application also implemented the smart Light Bulbs from Phillips to show that indeed the platform can be interfaced with other smart home devices. This was done quite easily as APIs for the Phillips Hue Light Bulb existed and allowed for easy interfacing.

7 Command Inference from Direct Messages

Given the inference algorithm that has already been implemented the direct commands from the Home Owner sent through Twitter's Direct Messages can also be applied so that the Home Owner can give natural language commands.

Since all the commands from the direct messages are considered to be completely independent of one another and since each command from a home owner contains all the necessary information. The inference of the command given past messages is unnecessary. Therefore, the inference algorithm was simply modified to allow for an Identity Transition Matrix and with only the current message probability vector we can apply it to a set of commands.

The only component left to implement is to create similar dictionaries (JSONs) for each command that contains states names and associated words with each state. Thus, each command becomes associated with a set of states and words that need to be fulfilled in order for the command to be valid.

For each direct message received from the Home Owner it is first checked to see if the command matches the default command pattern if it is not, then it is applied to the inference mechanism with no history. This allows for simply commands such as "Hometweetie turn on the lights" and "Could you switch on the lights" to be interpreted correctly as "HomeTweetie set lights on". Since all three states are contained within the commands.

8 Improvements

The amount of improvements that can be made to this project are numerous. First is the algorithm for inferring the command from the Home Owner. Here a simple Stochastic Markov Model is used. However, much more sophisticated learning algorithms could be employed as the Markov model assumes that the current probability encapsulates all past states. However, this may not always be the case.

The way images are posted to the Twitter feed allows for a very insecure and public display of the Home Owner's private information. This could be improved in a number of ways. Another image hosting service could be used to privately post the image to the Internet and a link to the Home Owner could be sent over Direct Message in Twitter. An alternative would be to develop a Twitter Application for mobile devices that decrypts images and messages HomeTweetie may post to its Twitter Feed or send through Direct Message. This would involve encrypting the data on the Gateway side as well.

Another Improvement is to use a more sophisticated natural language model. Here grammar and word order does not matter. The occurrence of the key words are only counted. This could lead to false positives as negation is not detected in this language model. Additionally, the occurrence of all key words may not necessarily equate to the desired meaning of the tweet.

9 Conclusion

A gateway using the Raspberry Pi Hardware was implemented to run a HomeTweetie application which utilized the Twitter Platform as a communication channel. Although the inference of the Home Owner's needs and commands may not be incredibly accurate due to the model used and the fact that often the content of the Home Owner's tweets do not reflect that of the home, the application did prove to function correctly when give the correct set of data.

Although the given application can be improved in many areas it can be easily scaled to a wider audience due to the fact that only a low cost Linux based machine running a JVM with an Internet connection is needed. It was also shown that HomeTweetie can easily be interfaced with other smart devices such as the Phillip Hue Smart Light Bulbs that are Internet connected.

10 Appendix

10.1 List of Hardware

Below is the list of hardware compenents used in the final implementation of the HomeTweetie project.

- Raspberry Pi B Model with Raspbian Linux Distribution running version September 2015.
- 16GB SD Card to store Raspbian Distribution for the Raspberry Pi
- Logitech Webcam C170
- PIR Sensor DSN-FIR800
- Micro-USB Charger with 2A output used to power the Raspberry Pi
- WLAN 11n Micro USB Adapter 1T1R (WiFi USB Dongle)

10.2 Dictionary for Command turnOnHeat

Below is the JSON file for the command turnOnHeat. Here we associate a number of words with the states of a command. The structure we have is a command with a name and a set of words with each object containing a name and an array of words associated with it. In the application this JSON is parsed and a Hash Map is constructed which is parsed when a tweet is analyzed.

```
{
  "command": {
    "name" : "turnOnHeat",
    "words" : [
                {
                        "main" : "weather",
                        "associated": ["weather","cold","chilly", "cool",
                           "brisk", "crisp", "frigid", "frosty", "frozen",
                            "icy", "snowy", "arctic", "wintry"]
                },
                {
                        "main" : "home",
                        "associated" : ["home", "apartment", "family", "
                           household", "house"]
                },
                {
                        "main" : "work",
                        "associated" : ["work", "working", "office"]
                },
                {
                        "main" : "going",
                        "associated" : ["going", "heading", "get", "back"
                           ,"driving", "walking", "biking", "cycling"]
                }
        ]
  }
}
```

11 References

[1] Openweathermap, . URL http://openweathermap.org/api.

[2] The pi4j project: Connecting java to the raspberry pi, . URL http://pi4j.com/.

[3] Beaglebone black, 2015. URL http://beagleboard.org/BLACK.

[4] Rpi hardware, 2015. URL http://elinux.org/RPi_Hardware.

[5] Twitter4j, 2015. URL http://twitter4j.org/en/index.html.

[6] Rate limits: Chart, 2015. URL https://dev.twitter.com/rest/public/rate-limits.

[7] Dan Jurafsky. Language modeling: Introduction to n-grams. URL https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf.