Machine Learning Project

Use a Predictions Algorithms in Smarties - Smart Home

Smarties

Smarties is an Autonomous System project for a virtual home, which manages different actuators as sensors and effectors in order to reduce some energy consumptions, increase house security and take multiple decision according to its configuration. The idea of all these functionalities is give to the end user the full control of his/her house, since Empty House mode, Fresh House mode, Warm House mode, etc, can be activated at any moment.

Smarties is divided into two components such as openHAB Framework and Virtual Home that connect each other using a broker called Mosquitto, which is an Open Source message broker that implements the MQTT protocol versions 3.1 and 3.1.1. For more information about this broker, visit Mosquitto.org.

Further detail about Smarties project, visit this Github repo.

Smart Home Architecture

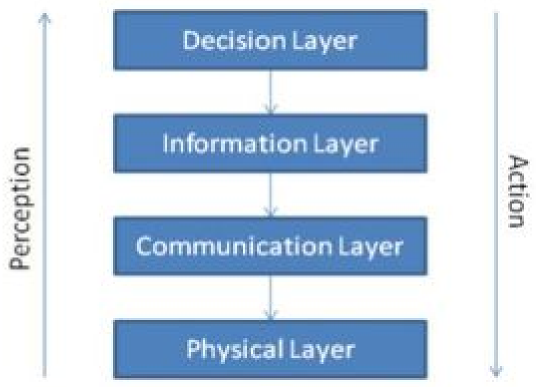
The architecture of a Smart Home can be accurately depicted as four layers. [1]

Physical Layer: This layer contains the basic hardware within the house including individual devices, transducers, and network hardware. [1]

Communication Layer: This layer includes software to format and route information between agents, between users and the house, and between the house and external resources. [1]

Information Layer: This layer gathers, stores, and generates knowledge useful for decision making. [1]

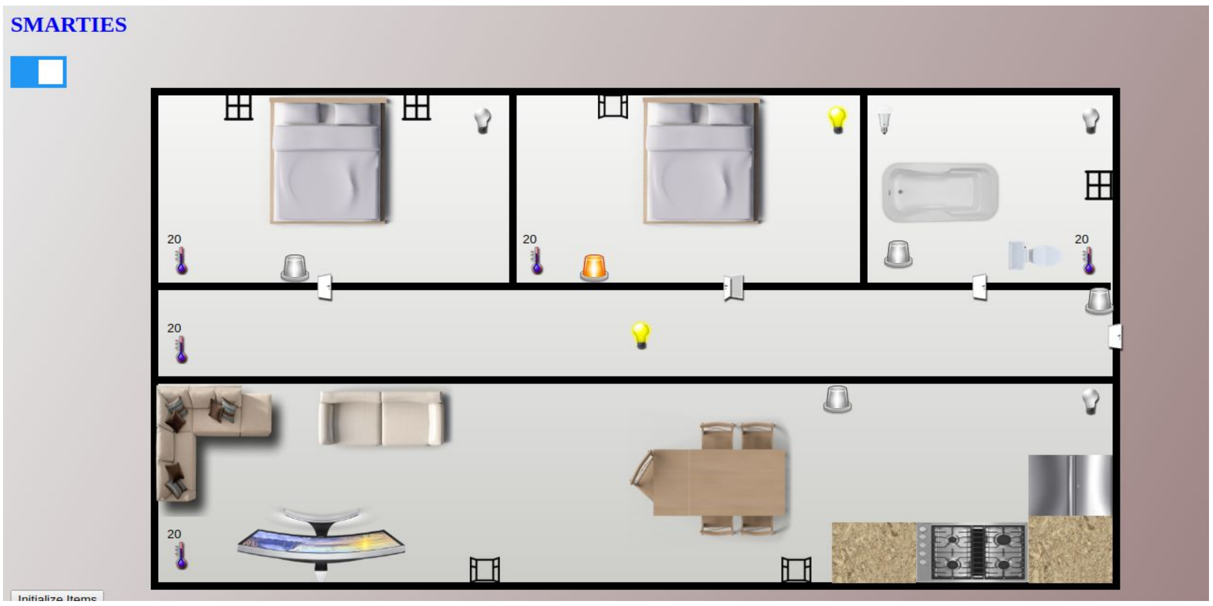
Decision Layer: This layer selects actions for the agent to execute based on information supplied from other layers. [1]



Smart Home agent architecture

Virtual Home

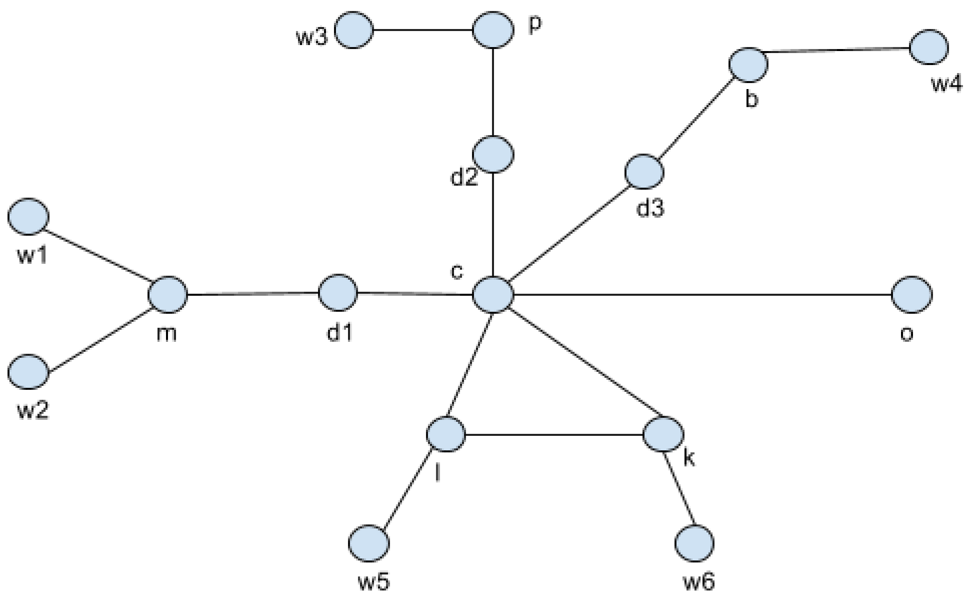
We decided to draw a virtual house as an interface with a series of items such as lights, motion and temperature sensors, windows and doors using HTML5 and CSS3. However, we used the Paho Javascript Client, which is an MQTT browser-based client library that uses WebSockets to connect to the Mosquitto broker.



Smarties - Virtual Home Interface

Mobility Model

The smart home coverage area is partitioned into zones or sectors. Location Management involves keeping track of user movement in such a way when the smart home needs to contact an inhabitant, the system initiates a search for the target terminal device by polling all zones where it can possibly found.
The Smart Home network can be represented by a bounded-degree connected graph, G=(θ,ε), where node set θ represents the zones and edge set ε represents the neighborhood (walls, hallways, windows, doors, etc.) between pairs of zones [1]. Next figure shows the representation of Smarties - Smart Home by a graph.

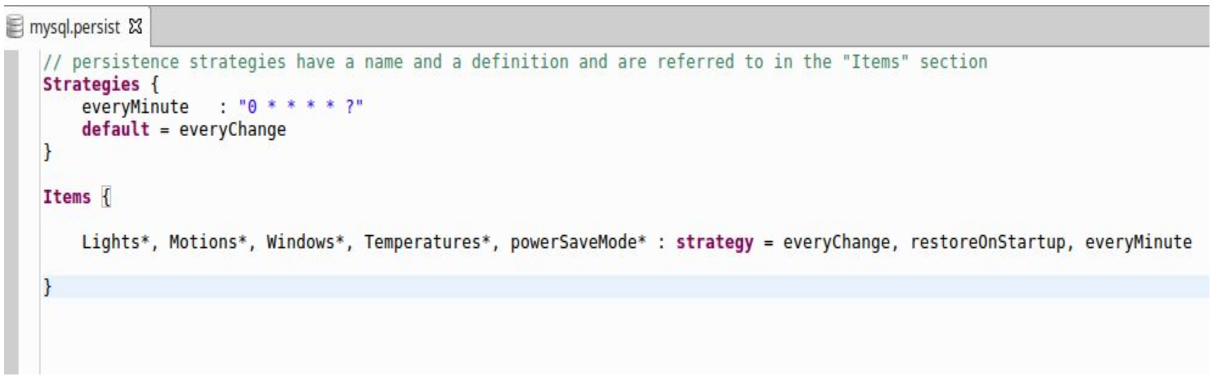


A graph model of Smarties floor plan

Zone	**Activity**
m	Wake up in master room
w1	Open the window 1
m	Back in bedroom
d1	Open the door 1
c	Out of bedroom
d3	Open door 3
b	Go to the bathroom
d3	Back to corridor
c	Out of Bathroom
k	Go to kitchen to make breakfast
l	Go to living room to eat
k	Back to the sink at kitchen
c	Out of kitchen
o	Go to work

Inhabitant Movement History example

The location determination of the inhabitant is a purely movement based schema. An update is generated whenever a zone boundary crossing is detected or a distinctly different user activity as compared to the last one is observed [1]. We can achieve this in a proper way since openHAB framework stores its items data according this mysql persistence configuration. We can check it in the next picture.



```
mysql.persist
// persistence strategies have a name and a definition and are referred to in the "Items" section
Strategies {
    everyMinute   : "0 * * * * ?"
    default = everyChange
}

Items {

    Lights*, Motions*, Windows*, Temperatures*, powerSaveMode* : strategy = everyChange, restoreOnStartup, everyMinute

}
```

Mysql Persistence config

The movement history of a user is represented by a String “v1v2v3...” of symbols from the alphabet θ, where θ is the set of zones in the house and vi denote the zone id reported by the ith update. Consider, the movement history of the Inhabitant Movement History example above, during a entire day is generated as mw1md1cd3bd3ckIkco.

The tacit assumption is that an inhabitant movement is merely a reflection of the patterns of his/her life, and those can be learned. This defines the learning phase that in turn aids decision making when reappearance of the patterns are detected [1].

Prediction Algorithms

LZ78
The LZ78 data comprension is an incremental parsing algorithm based on the Markov model. This algorithm has been interpreted as a Universal modeling scheme that sequentially calculates empirical probabilities in each context of the data; the generated probabilities reflect contexts seen from the beginning of the parsed sequence to the current symbol. The LZ78 is used only as a system that breaks up a given sequence (string) of states into phrases [1].

Algorithm:

```
initialize dictionary := null
initialize phrase w := null

loop
        wait for the next symbol v
        if((w.v) in dictionary):
                W := w.v
        else
                add(w.v) to dictionary
        w := null
        increment frequency for every possible prefix of phrase.
    endIf
forever
```

Active LeZi Algorithm

The Active LeZi is an on-demand algorithm that is based on Markov models and primarily stores the frequency of input patterns in a trie according to the compression algorithm LZ78. The amount of information being lost across the phrase boundaries increases rapidly when there is an increase in the number of states seen in the input sequence. This problem can be overcome by maintaining a variable length window of previously-seen symbols [1].

Algorithm:

```
initialize dictionary := null
initialize phrase w := null
initialize window: = null
initialize Max_LZ_length = 0

loop
    wait for the next symbol v
    if((w.v) in dictionary):
        W := w.v
    else
        add(w.v) to dictionary
        Update Max_LZ_length if necessary
        w:=null
    endIf
    Add v to window
    if (length(window) > Max_LZ_length)
        delete window[0]
    endIf
    Update frequencies of all possible window that includes v
forever
```

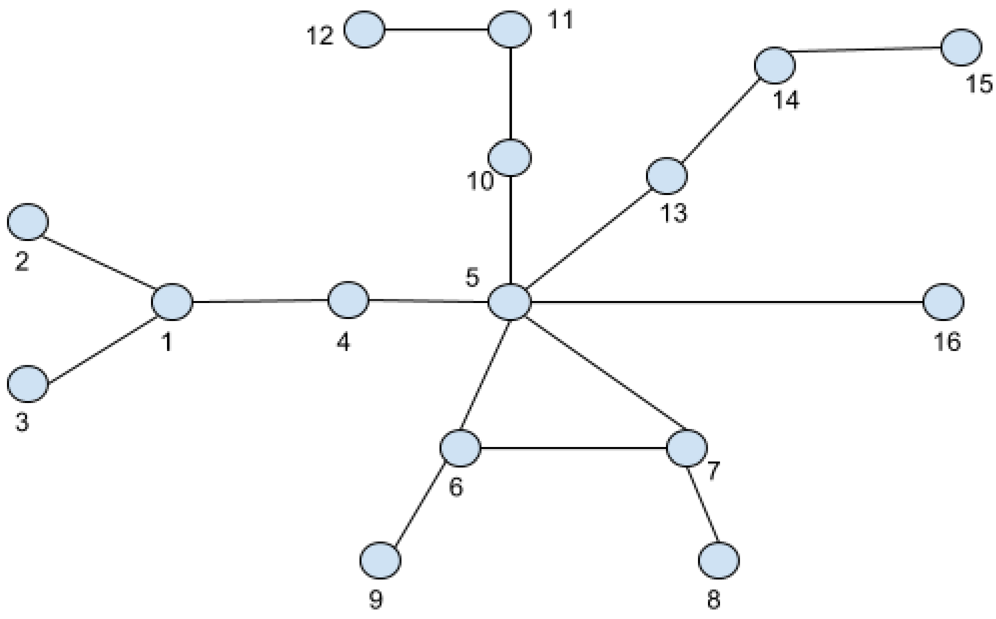
AKOM All-k-Order Markov

Techniques derived from Markov models have been extensively used for predicting the action a user will take next given the sequence of actions he or she has already performed [2].

Now, we are going to explain how we can predict the next inhabitant event using these kinds of algorithms. First of all, we have to performance an algorithm in Smarties Virtual Home that runs randomly values representing events of an inhabitant. Since all these data are stored in a database we can test different prediction algorithms in order to predict the next event.

Algorithm using randomly values

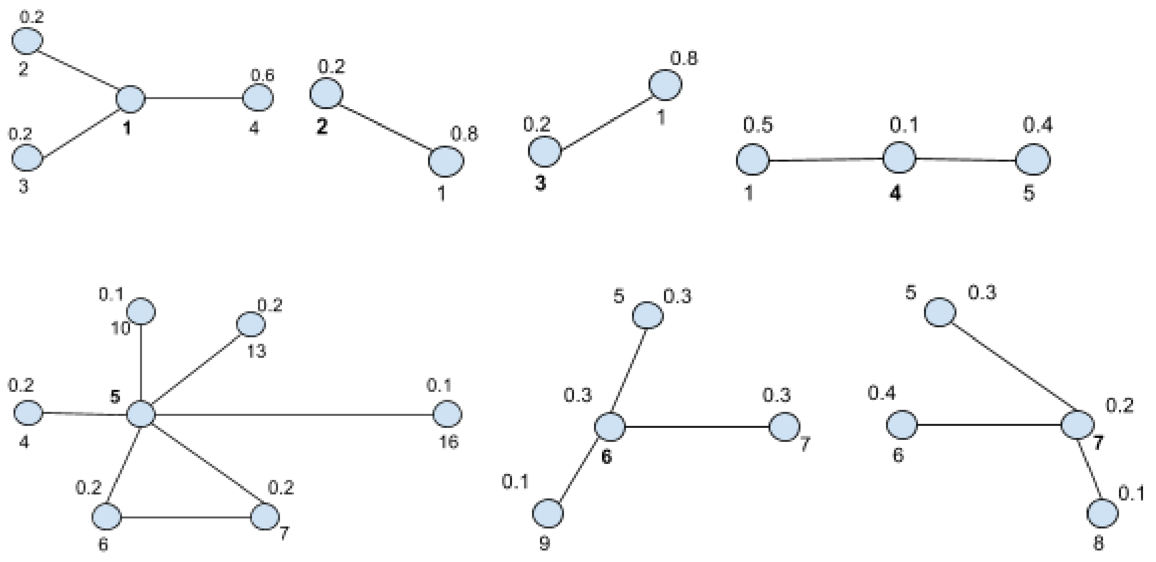
For simplicity, let’s identify each node of the graph with a number in such a way we can add different probabilities along them.

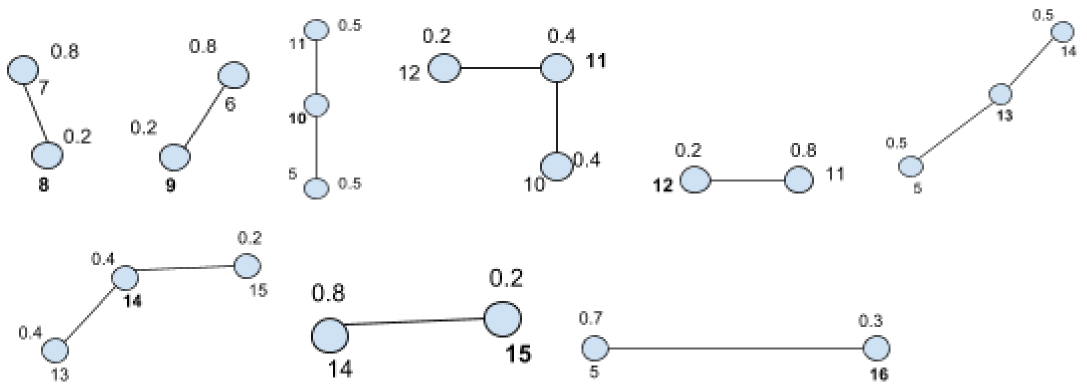


A graph model represented with numbers

Probability for each node

As we know each node represents an sector or zone of the house floor plan and each movement from node A to node B is taken by a probability according this graphs. For example, the node 1 has probability 0.6 to go to node 4, 0.2 to go either to node 2 or 3.

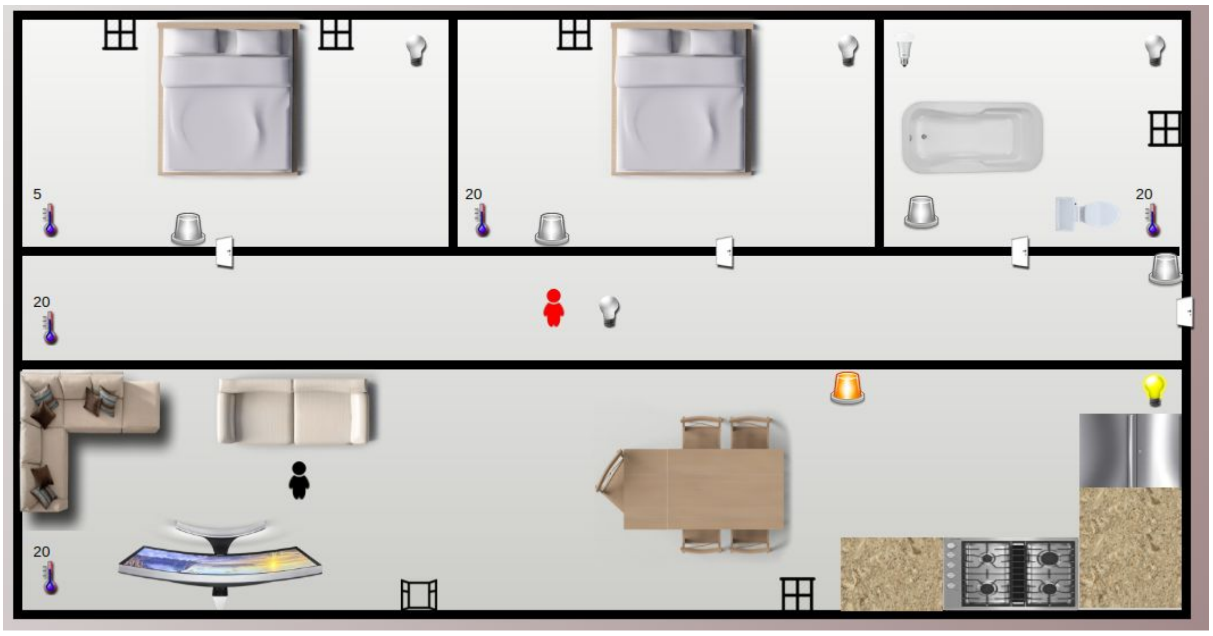




Every movement will be stored in a database as an event. A prediction algorithm will take those data to train it and then generate a prediction of the next event for a sequence of data.

Results

An algorithm was executed during a time of consideration that stores every movement of an inhabitant into a database. The number of data collected is 1615. After this, a new function is executed every 15 seconds in which black icon is moved through the house and another red icon appears in the position predicted for the algorithm. The following image shows the Virtual Home at a certain time.



References

[1] Aditi Dixit and Anjali Naik : Use of Prediction Algorithms in Smart Homes. International Journal of Machine Learning and Computing, Vol. 4, No. 2, April 2014.

[2] Mukund Deshpande and George Karypis: Selective Markov Models for Predicting Web-Page Accesses.