Upon a Home Assistant Solution Based on Raspberry Pi Platform

Alexandru Florentin IFTIMIE, Claudiu VINȚE
The Bucharest University of Economic Studies
iftimie.alexandru.florentin@gmail.com, claudiu.vinte@ie.ase.ro

Our ongoing research on Internet of Things (IoT) has been focused on a project aiming to creating a proof of concept for a distributed system capable of controlling common devices found in a house such as TVs, air conditioning units, and other electrical devices. In order to automate these devices, the system integrates various sensors and actuators and, depending of user's needs and creativity in conceiving and implementing new commands, the system is able to take care and execute the respective commands in a safe and secure manner. This paper presents our current research results upon a personal home assistant solution designed and built around Raspberry Pi V3 platform. The distributed, client-server approach enables users to control home electric and electronic devices from an Android based mobile application.
***Keywords*:** IoT, Home Automation, Java, Raspberry Pi, Arduino UNO

1 Introduction

The vision of IoT is to create the framework for a network of physical objects-devices, embedded with electronics, software, sensors, and network connectivity that enables the respective devices to collect and exchange data [1].

As the era of technology and information expands at high rates, researchers, scientists and hobbyists have begun to bring technology into home, and integrate it with various electronic or mechanical devices, in order to simplify and secure human living experience within society.

The IoT envisions a technologic environment that sustains the ability for objects to be sensed and controlled remotely, across existing network infrastructure [2], creating opportunities for more unobstructed integration of the physical world into computer-based systems, with results reflected in improved efficiency, accuracy and economic benefits [3], [4]. When IoT is augmented with sensors and actuators, the technology becomes an instance of the more general class of cyber-physical systems, and that is what our research effort intends to tackle.

As the era of technology and information expands at high rates, researchers, scientists and hobbyists have begun to bring technology into home, and integrate it with various electronic or mechanical devices, in order to simplify and secure human living experience within society.

2 Literature Review

The current state of the art development projects and research papers upon smart home applications are focused on some particular aspects of the field. Some of them have attracted a greater interest, such as electricity consumption management, component and device security, health related intelligent monitoring systems. While many researches have been conducted on certain parts of a smart home system, we have been primarily interested in creating a framework that could serve as a developing environment for various integrations, test different components and configurations, find flaws and identify improvements.

Electricity consumption monitoring applications are developed in order to reduce the cost of living. That is from the end user's point of view. Furthermore, from an environmentalist perspective, this type of management can improve the quality of society as less electricity is wasted. For instance, an on-line data power application with consumption data processed in cloud was proposed in [5]. The system consisted in a web server, a website, a hardware interface and the software application for monitoring the electrical switch control. The application generated daily, weekly and yearly reports

DOI: 10.12948/issn14531305/21.2.2017.01

regarding the electricity consumption. They found that this application encouraged households to carefully watch the energy consumption. Another research direction is related to smart grids, for both home and industry appliances [6]. The smart grid may provide customers the tools they need to reduce energy consumption, thus lowering the electricity costs and a more stable electric grid. The authors argue that high performing homes will have energy efficient walls, fenestration and technologies that communicate with customers.

End-user's ability to configure and program features is also a subject of interest in smart home applications. It regards the development of user friendly GUI applications that allow for customizing the system in a preferable way. On the other hand, in [7] the authors observe that the user's daily activities does not map well to programming tasks, thus they do not want to control their devices, but to be in control of their life, time, activities and relationships. End-user programming is typically hard to understand and it is quite rigid, not offering the full control of the system. Among end-user interfaces there are encountered implementations for natural language processing, visual programming, or magnetic refrigerator poetry [8].

The most important aspect in a smart home application is about the security standpoint. Automation systems are attractive targets for attackers [9]. The reasons include aspects such as non-stop internet connection, lack of a permanent system administrator, customer reluctance for system updates and upgrade, device specific vulnerabilities. Also, often the homeowner is misinformed and does not care too much about the security. User's mobile devices can act as a gateway to the home automation system. Moreover, in case of system malfunction, only an expert can actually fix it. There are specific security issues in connection with a neighborhood central control system, if that approach is the case. Although it may appear as cost effective, a minor breach can grant access to all homes connected to it [9].

Important technology players have started to offer smart home packages as well. One of them is Samsung with their SmartThings platform, which has the largest number of apps. In [10] the authors describe performing a number of penetration tests over 499 of their smart home applications and found flaws and bugs in most of the applications.

One of the most innovative smart home applications that we have reviewed is the Vital-Radio developed at MIT [11]. It is a wireless sensing technologies that is able to monitor and track with high accuracy the breathing and the heart rate of multiple users in an indoor environment. The application uses a radar technique called FMCW to separate the reflections arriving from objects into different buckets, based on the distance between a given object and the device. This type of application can relieve the user from carrying inconvenient and intrusive sensors such as the nasal probe or the chest band.

3 A Home Assistant Solution Based on Raspberry Pi Platform

The main objective of this project was to create a proof of concept for a distributed solution to remotely control a wide range of home devices. Specifically, the system should be capable of controlling lights, compute energy consumption, monitor and control doors, windows, coffee machines, AC units, TV sets, web cameras, read sensor values, control a home surveillance robot, alert user about burglary, and be as secure as possible.

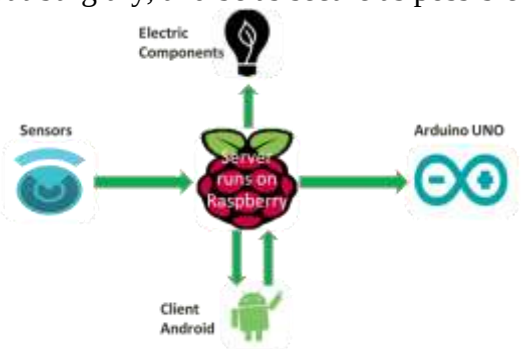

Fig. 1. Project's main components – overall interaction

The prototype that we propose is composed of a collection of circuits having data input sources like sensors and various output devices like LEDs, motors, and other

DOI: 10.12948/issn14531305/21.2.2017.01

consumers controlled by intermediate components like driver chips and relays. The core of the project is represented by the Raspberry Pi development board. The system's main components and their overall interaction are presented in Figure 1.

The chief component, the whole approach is based on, is a Raspberry Pi V3 which plays the role of command center. An Arduino UNO board is employed for controlling a toy car that will serve as a platform for a surveillance robot, and multiple electric components to command.

At the hardware level, on this board will be attached all the electrical components while, at the software level, a server written in Java will run and act as the main controller of all the electrical and electronic components, Figure 2.

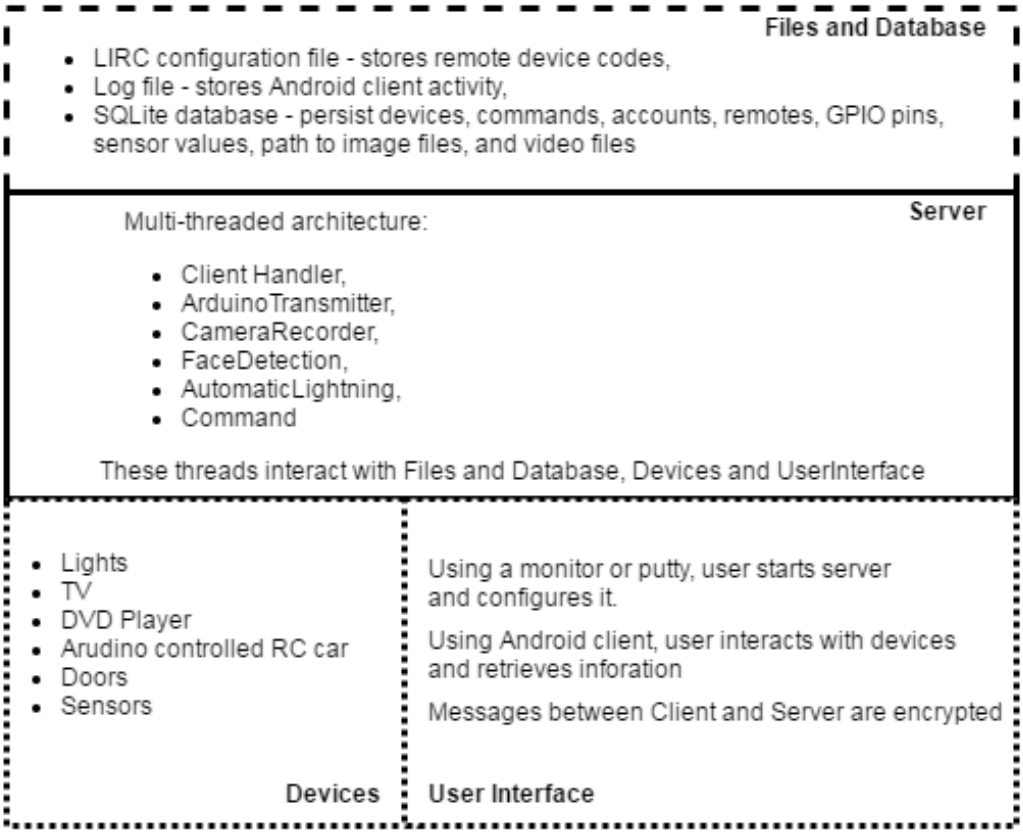

Fig. 2. The layered solution architecture

The current software architecture consists in:

- an application server, running on Raspberry Pi platform;
- a GUI wrapped around the application server, that plays the role of administrator console, as a desktop application accessible from the Raspberry Pi platform;
- a SQLite database, located on Raspberry Pi as well, and serving data persistency required by the application server;
- and an Android application, as a distributed client.

The key components the user will be able to interact with fall in one of the following categories:

- basic electrical devices, such as lights, motors, relays, or transistors;
- IR controllable devices such as air conditioning units, TV sets, video players, video projectors etc.;
- various sensors distributed across the house;
- surveillance cameras.

Each action generated by the client, such as retrieving the list of light sources, represents a request sent to the server, and then the client waits for the server response.

DOI: 10.12948/issn14531305/21.2.2017.01

Critical data, necessary for system state consistency and state recovery, in case of a hardware failure, is persisted in a SQLite database located on Raspberry Pi [4]. The system database consists in the following tables:

- *Accounts* - stores the account data for client applications: user ID, password, account status.
- *Commands* - stores the commands that are to be executed when a sensor reading reaches a predefined value, or satisfies certain conditions; a sensor may produce a discrete value between 0 and 1024.
- *Doors* - stores the door locations, the electrical component which opens the door, the electrical component which close the door, and the status of the door.
- *Lights* - stores the current status of the light source, its location, the associated pin code, and the associated time intervals the light source is to be activated or not.
- *Intervals* - stores the start and stop timestamps, along with the status of the light which has to be activated during that time interval.
- *Remotes* - stores the assigned names of the remote controls.
- *RemoteKeys* - names of the buttons and the ID of the remote control.
- *Videos* - an Android client application may record video clips which, rather than being stored on the smartphone, may be sent to the sever which can make them available to the other clients; store the path to the directory where the video file is located, along with name and the timestamp of the recording.
- *Cameras* - stores the location of the camera, its automatically generated ID, and status, ON or OFF.
- *DetectedFaces* - stores the path towards the pictures with the detected faces, when the house security option was activated.
- *GPIOOutputComponents* - stores the pin associated to component, name of the component, and action description; for example MotorRight means starting to turn an electric motor clockwise, and has associated pin 10.
- *GPIOPins* - stores the pin numbers for mapping: physical pin number, BCM pin code, Java 8 library code, current status, and if it is used or not.
- *SPISensors* - name of the sensor, the associated pin, the current value, and sensor description.

The server application is multithreaded, each thread being responsible for a simple and repetitive task. The main thread listens for the client requests, processes them in terms of the request type, then creates new child threads that take care of the specific request, and send the reply to the client, Figure 3.

The `AutomaticLighting` thread is responsible for initially retrieving from the database the list of lights and the operating time intervals associated to them, and then checks periodically, every second, if they are to be on or off. Inserting a new light, along with its associated time interval is achieved within `LightsPanel` class.

DOI: 10.12948/issn14531305/21.2.2017.01

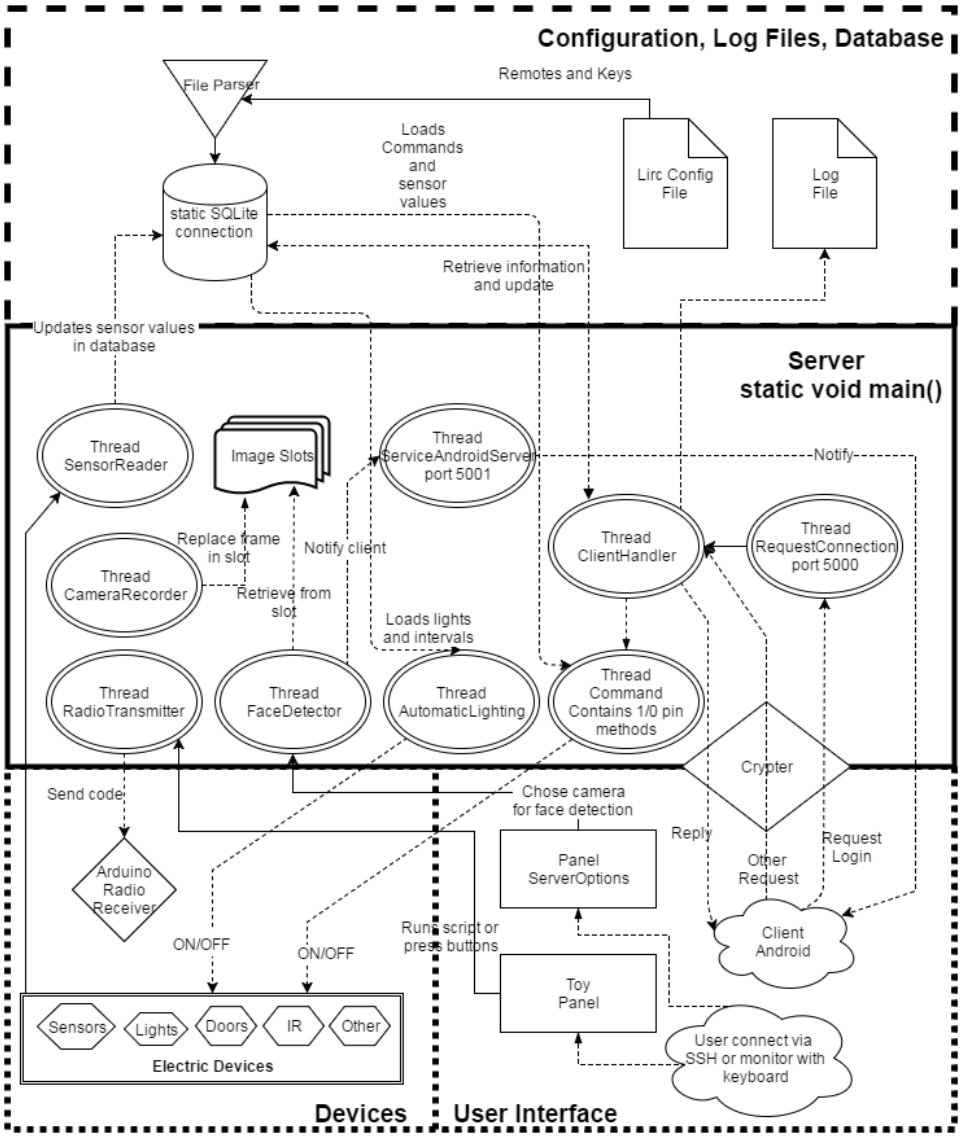

Fig. 3. Detailed component architectures

The Command thread contains methods for sending 1/0 signals to GPIO pins.

These methods employs other static methods from Pi4J library [5]. Below are examples of calls for creating a GPIO controller instance and its usage for fetching the 11th pin state.

```
GpioController gpio =
GpioFactory.getInstance();
GpioPinDigitalOutput pin =
gpio.provisionDigitalOutputPin(RaspiPin.
GPIO_11, "", PinState.LOW);
```

The `Command` thread fetches the command lists enrolled in the database, checks the conditions and, if they are satisfied, then the command is executed right away (if the delay time is zero, and the execution time is zero), or it is passed to an auxiliary thread (if the delay and execution times are different than zero).

There can be created commands for various scenarios:

- select the component to be executed;
- set the low and high threshold values, between 0 and 1024;
- select component status when the sensor reading is lower than the low threshold, and the component status when the value returned by the sensor is higher than the high threshold;
- set the delay from the moment the condition is satisfied, sensor reading lower than the low threshold, or sensor reading greater than high threshold;
- set the action duration – for instance, when the light sensor returns 500, then that is

DOI: 10.12948/issn14531305/21.2.2017.01

considered to be too dark and, within 1 second, a light will be turned on for 2 hours.

The `SensorReader` thread is created to monitor sensor activities and, if the user had defined and set a specific command/action for certain readings supplied by sensors, that command is executed right away. Another thread is used to query the SQLite database for certain preregistered events, like an alarm clock. Unlike a simple alarm though, which may only trigger a ring, on this thread can execute a certain user defined command.

The `CameraRecorder` thread is created to record video captures from all started cameras, and stores these files in area from which they may be accessed by threads created with other purposes, such as human face recognition, or live streaming to mobile client application.

The `ClientHandler` thread is created for accepting connections from the client applications, and then for each connection, there is created a `RequestConnection` thread for replying to client requests.

A distinct thread checks for each electrical consumer (light, motor etc.), which has associated at least one interval, if the current time falls into the specified interval and, consequently, if the predefined action has to be executed or not.

Once the client application sends to server the message "away from home", a new thread is created and started to monitor the security related sensors and alert the client if there were any activity in the house while it shouldn't had been.

There is a dedicated thread, `RadioTransmitter`, for sending radio signals to Arduino UNO board. On both ends, for the sender within the server on Raspberry Pi, and for the receiver on Arduino UNO, we make use of API provided by *RCSwitch* library.

An important aspect that has to be factor in is that the Raspberry Pi platform has a fairly modest number of GPIO pins. Using chips like MCP23008 or MCP23017 could supply Raspberry Pi with at least 64 pins, a number which covers most of the needs for sensor connections required by a medium sized house. Also the number of ADC that can be simultaneously connected is limited to 2 converters. That means a quite small number of sensors, respectively 16, as each one has 8 channels. For more ADCs a switch controlled by a GPIO pin should make possible for reading more sensors, but not at the same time, rather by reading one at a time, switching between them. Within the project it has been implemented as well a radio communication between Raspberry Pi and the Arduino UNO board mounted on a toy car chassis, via a Radio Transmitter-Receiver. The goal was to send simple commands like UP, DOWN, LEFT or RIGHT from the client applications in order to control de toy car. We intend to raise the complexity on this research path, in order to build a RC robot as Arduino has 13 GPIO pins and 5 analog pins. For controlling the communication between Raspberry Pi and Arduino UNO we used *WiringPi* library on both sides.

For controlling IR devices we make use of LIRC (Linux Infrared Remote Control), a command line program that listens for IR signals, records them in a configuration file, and sends signals via the Infrared LED. The programs *irrecord* and *irsend*, included of *Lirc*, are used to record keys from remote controls and to send the signals respectively.

In order to install and configure LIRC on Raspbian operating system, the following instructions must be executed:

```
sudo apt-get install lirc
```

Then the following lines must be added into the file `/etc/modules` where `gpio_in_pin` is the pin used for attaching the infrared sensor which will be used for recording the remote buttons, and the `gpio_out_pin` is the pin used to attach the infrared LED which be used for emitting the signal:

```
lirc_dev
lirc_rpi gpio_in_pin=23 gpio_out_pin=22
```

Another file to modify is `/etc/lirc/hardware.conf` where it

DOI: 10.12948/issn14531305/21.2.2017.01

will have the following content in it where the directive `LIRCD_ARGS` will tell which arguments will be loaded by default when the program is launched, the directive `LOAD_MODULES` will tell whether or not to load kernel modules:

```
LIRCD_ARGS="--uinput"
LOAD_MODULES=true
DRIVER="default"
DEVICE="/dev/lirc0"
MODULES="lirc_rpi"
LIRCD_CONF=""
LIRCMD_CONF=""
```

And the last file to modify is `/boot/config.txt` which is necessary if the installed firmware version is 3.18.x in order to load the `lirc-rpi` kernel extension:

```
dtoverlay=lirc-
rpi,gpio_in_pin=23,gpio_out_pin=22
```

After the file has been modified, the following commands must be executed in order to load the changes:

```
sudo /etc/init.d/lirc stop
sudo /etc/init.d/lirc start
```

Before the command line program can be used, first of all the server will start a daemon that is responsible with the actual process of emitting the infrared signal. To start the daemon, the following line of code will be executed when the server starts.

```
Runtime.getRuntime().exec("sudo lircd -d
/dev/lirc0");
```

After the daemon is started, the server is ready for accepting new clients that will request code numbers to emit. The code numbers for the remotes that have been previously recorded into a LIRC configuration file, are parsed from that file and stored into the database so that the user is not responsible with remembering the codes for the remotes. The following line of code will make the daemon start emitting a light signal:

```
Runtime.getRuntime().exec("/home/pi/Desk
top/wiringPi/433Utils/RPi_utils/codesend
"+nr);
```

Regarding the security aspects of the personal home assistant solution, we have experimented with web camera and magnetic sensor. The system can be programmed to alert every client application and send messages containing information about a door status. If the door is open, an action is triggered by the magnetic sensor found on the frame of the door since it is no longer activated by the south pole of the magnet found on the door. Also the system can take photos for a specific amount of time specified by the user. This action is triggered either by sensing the door open or detecting a face. For image processing has been used *OpenCV* library with Java code.

4 GUI Console Features

The system administrator is the only one who can access the server console. The server side administration is achieved through a desktop Java GUI. This Swing application is wrapped around the server and offers access to account table, GPIO connections and other devices. The desktop GUI is organized in multiple panels from which the administrator can configure the system. The first panel controls the *Lights*, which can be turned on and off and added intervals to specify that at a certain part of the day the light should be turned on or off automatically, Figure 4.

The *Doors* panel enables the user to select a particular door, then which component locks the door and which component unlocks the door, and provides as well the ability for choosing the current state of the door.

From the *Server* panel, the user can start and stop the server, provide the port number, and also check the IP cameras.

DOI: 10.12948/issn14531305/21.2.2017.01

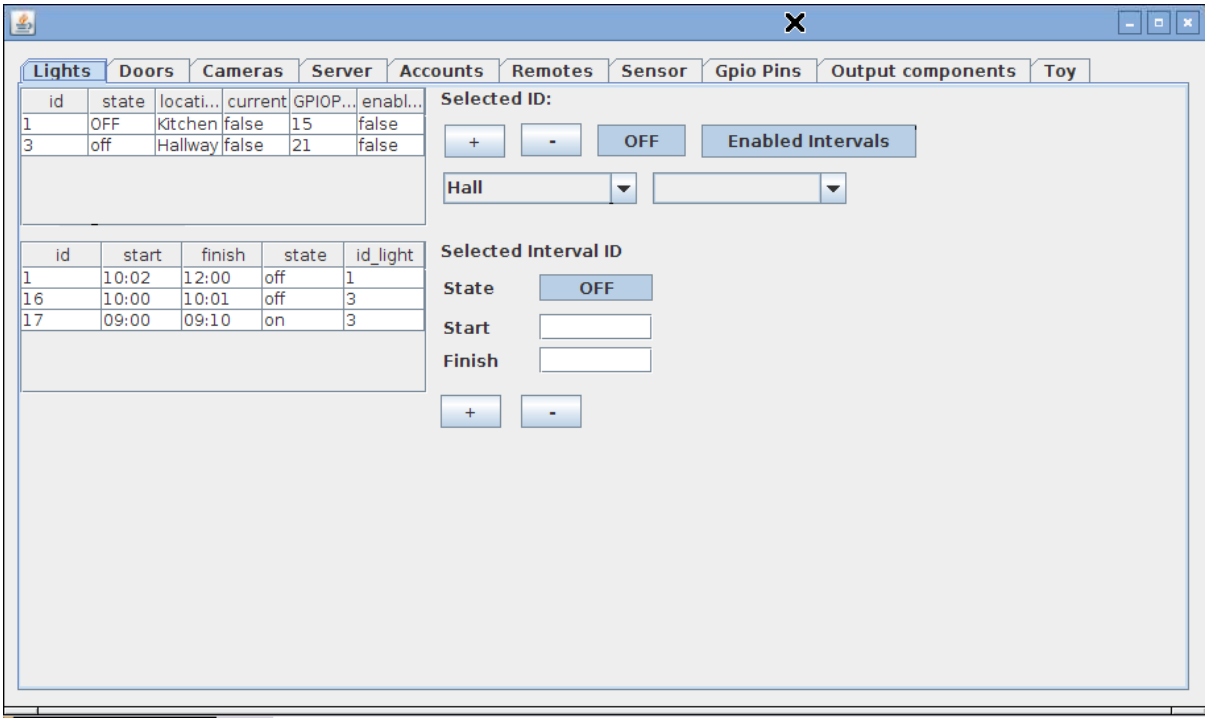

Fig. 4. Desktop client GUI – Lights configuration panel selected

There is also provided a text console for logging the actions that have been executed recently. From the *Remotes* panel, the user is able to add new IR remote controls to the database. As the *Lirc* is a command line program, when the user presses the button for recording new signals, a terminal and an instruction panel are showed.

Within *Cameras* panel, the video cameras connected to the system can be configured and managed. The thread `CameraRecorder` maintains a map of video captures associated to each video camera. This thread insert images in the corresponding location of the map. The `ClientHandler` thread copies images from this map and sends them to the client application. Based on the client request, the `FaceDetector` thread may as well copy an image from the map and process it for face recognition. Due to performance reasons, since for a single frame the system requires about 3 seconds of processing, while the Raspberry Pi processor reaches about 50% of load during this time interval, only one `FaceDetector` thread is launched at a time. From the *Toy* panel the toy car movements can be controlled. When activated, the `RadioTransmitter` thread sends signal codes every 300ms. Our experiments showed that if the time interval between signals is narrower than 200ms the signal is distorted. We have to investigate if the phenomenon occurs due to the Radio Transmitter and Receiver characteristics, or it is cause by electromagnetic interference. The toy car unshielded motors reside in the proximity of the Radio Receiver located on the car. A very important part of the application regards the *Sensor* panel from where the user can program certain actions that are to be executed when a condition is satisfied based on sensor readings, Fig. 5.

DOI: 10.12948/issn14531305/21.2.2017.01

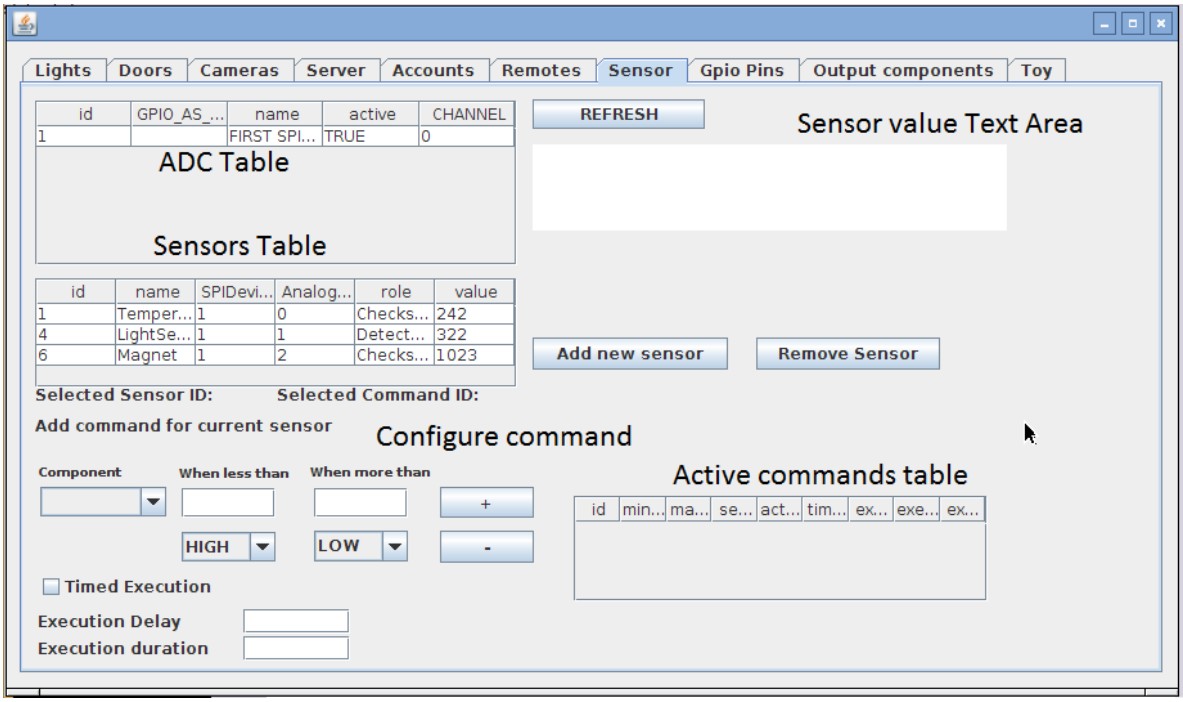

Fig. 5. Sensor panel options

This is the panel where the administrator has the capability to create and manage commands, in the manner that we presented in the server section.

5 Android based client application

In order to control the entire system while away from home, we design and implemented a dedicated mobile client application. The end user has been provided with a personal account which is used to authenticate for the Android application in order to be able to visualize and remotely control home devices. The client application has a setting panel to allow for creating a new account, delete an existing account, update password, add new IP address and Port number to connect to, and display various useful information, Figure 6.

The client application is able to send and receive messages to and from the server. Between the mobile client application and server the messages passed are encrypted Java `String` instances. The class `Crypter`, contained in `java.crypto` package provides implementation of AES algorithm and the methods for encrypting, respectively decrypting Java String.

The messages coming from the client have the following CSV format:

```
user_name,action[,values, …]
```

For example, the following messages represents request toward the server with the meaning provided as Java comment:

```
Alex,GetCameras // request for the list
of active surveillance cameras
Alex,StartCamera,Garage // start
receiving video frames from the garage
camera
Alex,GetLights // request for the list
of installed lights
Alex,UpdateLights,Kitchen,ON // turns on
the kitchen light
Alex,GetIRDevices // request for the
list of enrolled IR controlled devices
Alex,GetKeys,SonyDVD // request for the
key of a certain remote control
Alex,UpdateIR,SonyDVD,KEY_VOLUMEDOWN //
request for lowering the volume of a
Sony DVD player
```

DOI: 10.12948/issn14531305/21.2.2017.01

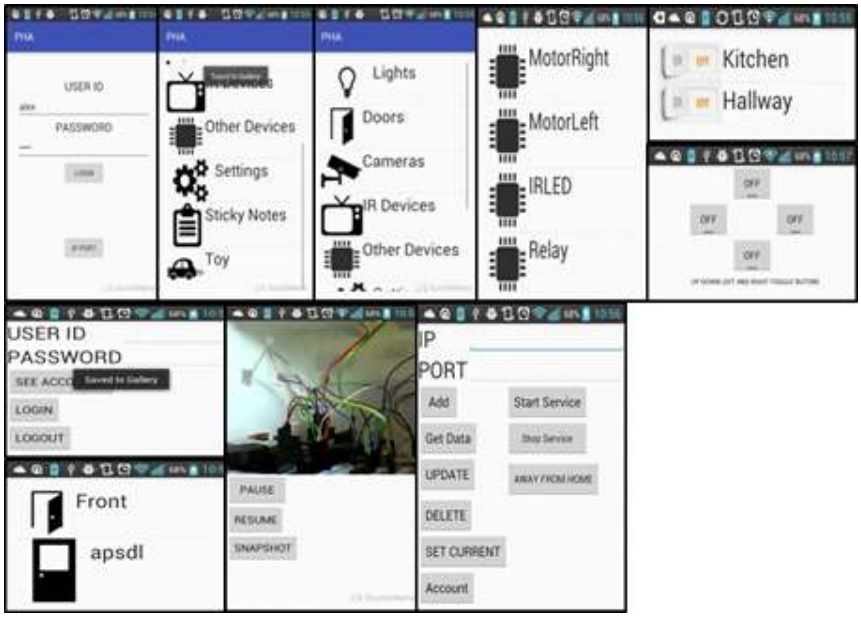

Fig. 6. The Android based client application – various screens

The messages sent by the server have following generic format:

- number of devices concerned (lights, motors, relays etc.), followed by a list of pairs:
  - device name (identifier),
  - device status.

Once an update request is sent to the server, the server does the following actions:

- sends a command to the corresponding GPIO, if necessary, for actuating the physical device;
- updates the device status in the database;
- sends and update to all client applications with the new status of the device concerned for maintaining consistency within the distributed system.

Each message received by the server from client is recorded in a log file. If a mobile phone has been lost or stolen, having the client application running on it, the administrator can delete the user data from the database and then the client application is automatically logged out from the server.

For example, the client could record a video message or write a text message and send it to the server without being necessary to use local storage. The reason behind implementing such a function derives from the need of having the server communicate in a consistent manner, RESTfully, with all registered accounts, family members respectively. As home security is of high interest for all the family members, the message will be stored on the server side until every registered client consumes it, or until the administrator decides to remove it.

6 Security Aspects

We believe that security is the most important feature of a home assistant solution. No potential user will trust an insecure application when it comes to a real-estate property. Apart from the basic aspects of security such as user authentication, we implemented methods for different scenarios.

Technology solution pitfalls come from both software and hardware components. When discussing about software weaknesses, those concern main server accessibility, and the communications between server and client applications.

Starting from the easiest break-in method that is using another computer to disable the system or take control over all devices in the house, the current system's version included the change of port number for the Secure Shell login on the server. As there are at least 60000 possible ports available, an attacker will have lower chance of breaking into the system.

Other measures for limiting the access to the server via Secure Shell include the creation and allocation of public and private keys for the users of the system. This method will

DOI: 10.12948/issn14531305/21.2.2017.01

lower the chances even more of allowing an unknown user from accessing the server. Even if the user will access the server by using a broken account, the account's privileges are very limited so that the system will not be fundamentally affected. In addition, the root login for the same reason mentioned earlier.

The software component that has been developed for accessing the devices and receiving connections from client applications can also present a set of exploitable features. Therefore, we used the implementation of Advanced Encryption Standard (AES) found in Java `Crypto` class for encrypting all messages that are exchanged between the client and the server. The only messages that are not encrypted consist of web camera live transmission. As a single frame from the camera has at least 1.5 MB in size, the encryption and the decryption of the bytes would have taken too much computation time, along with a network traffic increase since the algorithm introduces several extra bytes.

It was chosen to have the encryption key required by the algorithm hardcoded, because it would have been too much overhead for the user to remember an encryption key.

Regarding the hardware, one of the components that potentially may pose security vulnerabilities is the magnetic sensor for the door. As the magnetic sensor checks whether in the near proximity there is a magnetic field of south orientation, this can easily be fooled if the hostile person has knowledge about such a sensor by simply keeping a large magnet around the sensor while lock-picking the door.

As a second layer of protection, we implemented a facial detection heuristics that alert the user when it is not at home. In order for the application to not give false alarms, if the detection algorithm was trained enough, the user must first announce the server about not being at home.

The last level of security implemented within the proposed prototype is user delayed authentication from the mobile application. When the user is accessing the application, the first operation is to send to the server a message that contains the username and the password. When the server receives this message, it waits for a while before checking the database, and then continues with sending back the approval or denial code.

7 Conclusions and Further Research

The solution that we have presented in this paper is the result of a particularly applicative research. At the current stage of our research project upon a personal house assistant solution based on Raspberry Pi platform we have successfully implemented most of the objectives that we aimed for. The solution, although it is inexpensive, as it cost less than 200 Euros, it is quite rough and fairly complicated at this point for a non-technical end user to understand the whole process and make personal adjustments. Our ongoing research on IoT is focused in the near future on improving the level of security for the current home assistant solution, along with tweaking the implementation regarding AC units control via IR interface, monitoring windows, and building an autonomous surveillance robot. There are also considerations that have to be taken into account regarding the integration of such a solution within an existing house interior, since most of the sensors and actuators are wired to the Raspberry Pi platform. An entirely wireless approach, particularly based on WI-FI, would be marginally more expensive, and would require encrypted messages for ensuring security, but would offer a much greater flexibility with regard to the solution architecture design, installation and long term maintenance. Our main focus is to design and implement a coherent and comprehensive API for interconnecting generic WI-FI based distributed controllers with application services residing on Raspberry Pi platform.

References

[1] Internet of Things Global Standards Initiative, ITU July 2015, http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx/, (April 29, 2016)

[2] Internet of Things: Science Fiction or Business Fact, 2014 A Harvard Business

DOI: 10.12948/issn14531305/21.2.2017.01

Review
[3] Analytic Service Report, https://hbr.org/resources/pdfs/comm/verizon/18980_HBR_Verizon_IoT_Nov_14.pdf/, (April 29, 2016)
[4] Ovidiu Vermesan, Peter Friess: Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems, http://www.internet-of-things-research.eu/pdf/Converging_Technologies_for_Smart_Environments_and_Integrated_Ecosystems_IERC_Book_Open_Access_2013.pdf/, 2013 River Publishers, (April 29, 2016)
[5] R.G. Marvin. Garcia, Hannah R. B. Chan, Benilda E. V. Comendador , Grant B. Cornell, Christopher D. Celestial, and Arc E. P. Mercolesia: Smart Home Electricity Management System Using Cloud Computing (SHEMS). In: Journal of Advances in Computer Networks (2013)
[6] Kara Saul-Rinaldi, Robin LeBaron and Julie Caracino: MAKING SENSE OF THE SMART HOME
[7] S. Davidoff, M.K. Lee, C. Yiu, J. Zimmerman, A.K. Dey. (2006) Principles of Smart Home Control. In: Dourish P., Friday A. (eds) UbiComp 2006: Ubiquitous Computing. UbiComp 2006. Lecture Notes in Computer Science, vol 4206. Springer, Berlin, Heidelberg
[8] K.N. Truong, E.M. Huang, & G. D. Abowd. CAMP: A magnetic poetry interface for end-user programming of capture applications for the home. In: Proceedings of Ubicomp (2004)
[9] Arun Cyril Jose, Reza Malekian: Smart Home Automation Security: A Literature Review. In : Smart Computing Review (2015)
[10] Earlence Fernandes , Jaeyeon Jung , Atul Prakash: Security Analysis of Emerging Smart Home Applications. In : Security and Privacy (SP), IEEE Symposium (2016)
[11] Fadel Adib Hongzi Mao Zachary Kabelac Dina Katabi Robert C. Miller: Smart Homes that Monitor Breathing and Heart Rate

Alexandru IFTIMIE is student in the final year of The Faculty of Economic Cybernetics, Statistics and Informatics within The Bucharest University of Economic Studies. His domain of interest and research include artificial intelligence, neural networks, computer vision and robotics. His major achievements include first prizes at student scientific competitions, hackathons, and innovative projects competitions.

Claudiu VINŢE graduated in 1994 The Faculty of Economic Cybernetics, Statistics and Informatics, with the leading overall average of 9.91/10. He is member of The Department of Economic Informatics and Cybernetics. Claudiu holds a PhD in Economics from The Bucharest University of Economic Studies. His domains of interest and research include heuristic and metaheuristic algorithms, middleware components, trading technologies, algorithmic trading, and data analysis.

DOI: 10.12948/issn14531305/21.2.2017.01