SmartRoom: A context-aware set of IoT devices for your pets

Federico Gavioli

224833@studenti.unimore.it

Abstract

Taking care of a pet is a time-consuming task for the common working individual. Would it be possible to ease this burden and gain some precious time to spend with our beloved pets? SmartRoom can efficiently connect different Things to provide a comforting environment for your pets while you're at work and automate some pet-related tasks like replenishing food and water bowls. This avoids the issue where your lovely pets are barking or meowing around the house asking for food, thus reducing stress during the hard evening hours and giving you more time to play with them.

1 Introduction and Use Case

During weekdays, we spend most of our time away from home. The remaining non-work hours are usually condensed around the evening, where all the other daily housekeeping tasks are also piled up. In this common scenario a lot of time is wasted to take care of pet related issues, limiting the quality time we can spend with our small friends. Specialized COTS products that automate these tasks exist, but they're usually not open source or flexible enough to be integrated in the environment as this project.

The main objective of this project is to provide some really simple yet efficient devices automatically integrated into a single handling software like Home Assistant.

For simplicity, the use case considered for this project includes a single room house inhabited by a pet and one or more human owners.

This project is entirely open source. The code is available on GitHub [1].

2 Network Stack and Hardware Infrastructure

In this section is presented the hardware infrastructure used in the project and how the communication protocols are implemented to connect the different devices.

2.1 Hardware and architecture

This project is mainly implemented via ESP32 microcontrollers, which are able to electrically interface with their sensors and actuators and connect via WiFi to the same router. Each of these devices then exposes its sensors and actuators to the network with minimal data processing.

On the other hand a Raspberry Pi 3B+ acts as the centralized broker of the network. It runs a Linux distribution and hosts the Home Assistant Web Server.

The architecture of the system, as seen in Figure 1 currently contains three IoT devices and the coordinator. The IoT devices are in brief a food and water dispenser (Smart Feeder), an air conditioning unit (Smart AC) and a pet collar (Smart Collar). The system also supports any

[1] https://github.com/fgavioli/SmartRoom

web-capable general purpose device (PCs, smartphones, tablets, etc...) that can provide configurations and preferences via the Home Assistant Web App and monitor all the data gathered by the connected things.

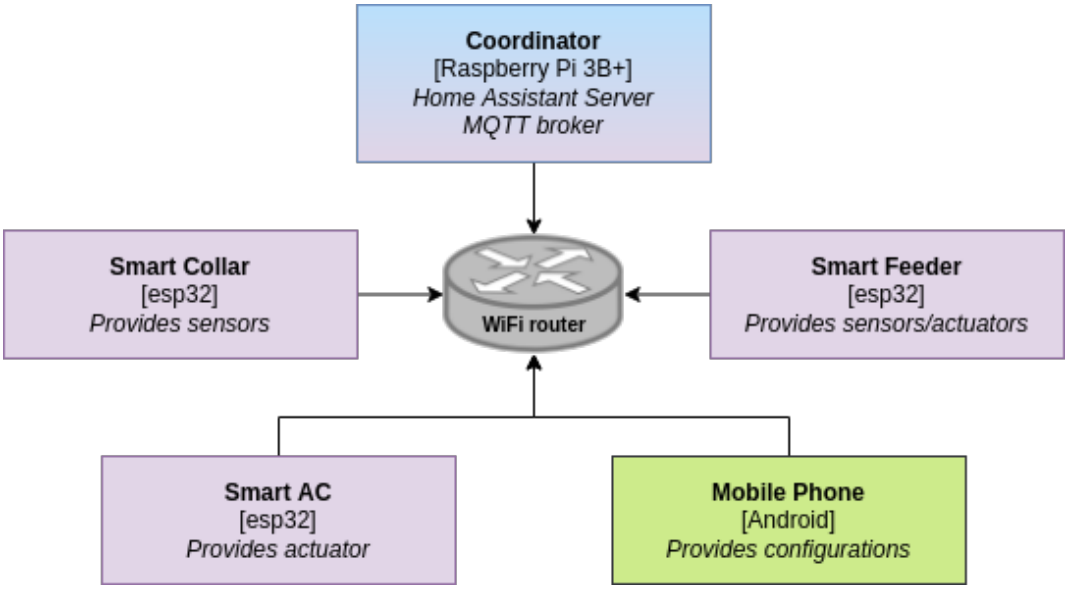
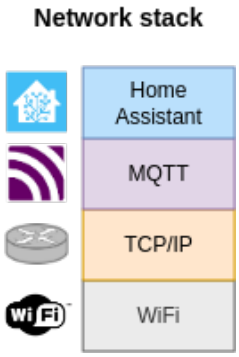


Figure 1: Devices and infrastructure

All the devices are connected via WiFi to a router and exchange data via MQTT, which is then integrated into Home Assistant via the MQTT Integration[2]. In the following subsections are presented the main motivations behind the choice of each part of the stack.



2.2 Network and Transport Layers

Low level communication between devices was achieved using the standard TCP/IP stack for different reasons:

- Given the use case for the project, the communication between things is expected to work in small areas like a small studio, where WiFi can provide more than sufficient performance and coverage
- All the ESP32 boards used in this project natively support WiFi connectivity
- Other protocols like ZigBee or LoRa require specialized network interfaces and gateways to operate, which increases costs

[2]MQTT Home Assistant Integration available at https://www.home-assistant.io/integrations/mqtt/

2.3 Application protocol - MQTT

To provide higher level communication between the things and the coordinator, many of the commercially available application level protocols were checked out. While CoAP and other RESTful protocols (like HTTP) were considered, ultimately the choice fell on MQTT for different reasons:

- MQTT is a well-known and mature standard, with a long history of usage in the IoT industry and wide support in regards of its different integrations and documentation.
- The REST interface used by many protocols requires a synchronous communication between things, which may be offline for different reasons (maintenance, running out of battery, etc...)
  - On the other hand, MQTT offers (out of the box) retained messages to provide state consistency during reboots. These are relevant in this project since some devices may be disconnected for maintenance from the network.
- MQTT offers more flexibility in regard to QoS levels (fire and forget, at least once, exactly once), CoAP offers only two levels (confirmable, non confirmable) while HTTP doesn't implement any QoS capability out of the box.
- To my knowledge, while CoAP *is* used in the industry with Home Assistant[3], it doesn't officially provide a general purpose integration, while MQTT has a very powerful Home Assistant integration that also supports automatic discovery of devices, sensors and actuators[4] [5].
- MQTT is initially offered as a TCP/IP protocol but also supports other lower power communication protocols like Bluetooth and ZigBee (with MQTT-SN) which could be used to better support energetically constrained devices

Among the different MQTT broker implementations, Mosquitto[6] by the Eclipse Foundation was deployed. lightweight and open source, Mosquitto is one of the most used and easy to deploy MQTT brokers.

2.4 Application and Presentation - Home Assistant

To enable the monitoring and automation of the different things the coordinator hosts an Home Assistant server available to all local network devices.

Since all the devices communicate through MQTT, the Home Assistant MQTT Integration was used to map the sensor/actuator topics of MQTT to entities into Home Assistant.

To integrate any type of MQTT entity into Home Assistant, a configuration for each sensor/actuator has to be provided. While configurations can be manually inserted into the `configuration.yaml` file by adding a "Custom MQTT Entity", there is an additional functionality named MQTT Discovery, which allows any device to describe its capabilities to Home Assistant. By using the MQTT Discovery, any device can publish a message containing the configuration string for a sensor or actuator into a discovery topic. Once this configuration reaches Home Assistant via MQTT, it will be processed and automatically added as a card to Lovelace, the default dashboard. Here follows as an example, a configuration for the temperature sensor inside the Smart Collar.

```
1  {
2      "device_class": "temperature",
3      "unique_id": "temperature0",
4      "name": "Temperature",
5      "state_topic": "homeassistant/sensor/smart_collar/temperature/state",
```

[3] Shelly provides a Home Assistant CoAP integration at https://www.home-assistant.io/integrations/shelly/
[4] The Home Assistant MQTT integration at https://www.home-assistant.io/integrations/mqtt/
[5] MQTT Discovery manual at https://www.home-assistant.io/docs/mqtt/discovery/
[6] Mosquitto at https://mosquitto.org/

```
    "unit_of_measurement": " C ",
    "value_template": "{{ value_json.temperature}}",
    "expire_after": 300
}
```

3 Devices

The architecture of the system currently provides four different devices. Each device is implemented using an ESP32 microcontroller connected via WiFi to the main router. The network knowledge of each device is limited to the local hostname of the MQTT broker (`coordinator.local`), so that it's entirely independent on the other devices to offer its services. The discovery and coordination logic is centrally implemented inside Home Assistant via MQTT Discovery and Automations.

3.1 Smart Feeder

The main static device in the ecosystem is the Smart Feeder. It contains a water tank and a food dispenser. The feeder is capable of controlling the flow of water on a closed loop with a pump and to dispense food on request with a servo motor.

The water pump draws a lot of power during operation, so it can't be powered with the 3.3V output pin on the ESP32, so it's controlled with a relay connected to a 5V wall adapter as seen in Figure 2.

The servo motor used to dispense food is controlled via a 50Hz PWM signal with a variable duty cycle (in the 10-20% range) to set its position. This is powered via the 3.3V pins on the ESP32.

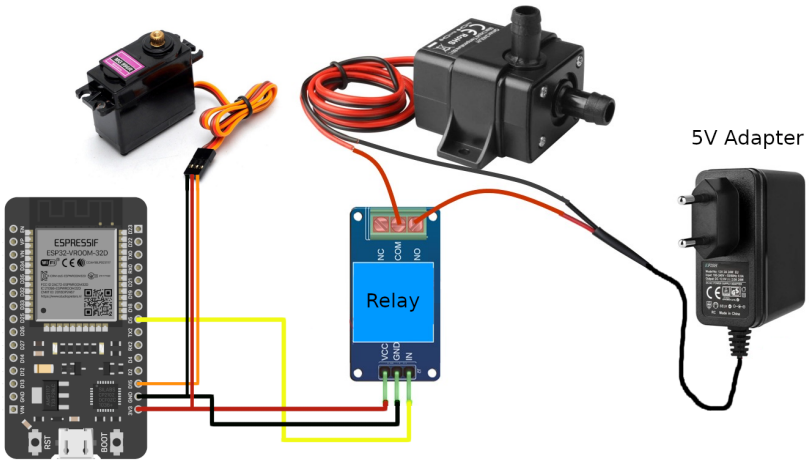


Figure 2: Smart Feeder schematic

The smart feeder makes use of MQTT retained messages on the command topic of the pump to keep the status value saved in case of reboot or loss of connection.

3.2 Smart AC

The Smart AC is a simple air conditioning unit. It exposes on the network the activation control of the cooling unit.

The smart AC makes use of MQTT retained messages on the command topic of the cooling unit to keep the status value saved in case of reboot or loss of connection.

For demonstration purposes, the cooling unit ON/OFF status will be represented with a LED connected to the ESP32.

3.3 Smart Collar

The Smart Collar is a custom pet collar providing to the network the percieved temperature of the environment and its activity status. To achieve this, the Smart Collar is equipped with two sensors:

- The **DHT-11** temperature sensor is able to measure the temperature and humidity of the environment and calculate the Heat Index based on the current measurement. Its data is provided as-is to the network.
- The **GY-521** sensor on the other hand provides an accelerometer, a gyroscope and a temperature sensor.

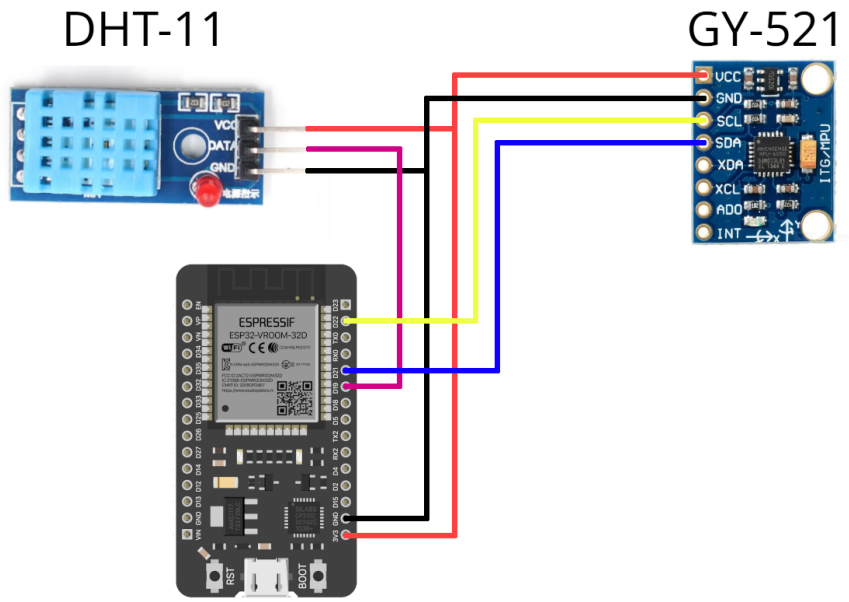


Figure 3: Smart Collar schematic

The first approach for the pet movement detection was to filter the gravity out of accelerometer data from Since the gravity can't be filtered out of the GY-521 (the gyroscope is too noisy to filter out the gravity by rotation) the pet movement detection is achieved by calculating the average jerk based on the accelerometer data in a 5 seconds period, as you can see in the following formula.

$$\text{avg}(j) = \frac{\sum_1^n |j_i|}{n-1} \quad \text{with} \quad j_i = \frac{\sqrt{a_{x_i}^2 + a_{y_i}^2 + a_{z_i}^2} - \sqrt{a_{x_{i-1}}^2 + a_{y_{i-1}}^2 + a_{z_{i-1}}^2}}{\Delta t} \tag{1}$$

In each iteration (occurring at 1Hz) the accelerometer data is sampled from the sensor and used to calculate the total acceleration magnitude. This value is used with the data from the previous iterations to calculate the total jerk (j_i) of the current iteration. This value represents the difference in acceleration of the current iteration. This value is sampled for $n-1$ iterations, and taken as its absolute value to calculate the average jerk. This gives us a measurement of how much is the acceleration changing in time. Since the accelerometer is quite noisy the average jerk at rest is around 0.05 m/s^3. To classify the current jerk measurement as 'moving', a threshold of 0.1 m/s^3 was set to avoid false positives. This binary value ('moving' or 'still') is finally provided via MQTT to the network.

The GY-521 also provides a temperature sensor, but since it's quite inaccurate when compared to the DHT-11, all temperature-related measurement are handled with the latter.

Since this device is energetically constrained (battery), the data is sent at a rate of one update per minute. This rate gets further reduced to one update every five minutes if the connection to the router or broker fails for whatever reason. Since no data is stored between iterations, the hibernation feature of the ESP32 is also used to minimize power consumption during offline time. The device was finally underclocked (from 240 to 160 MHz) to further reduce the energy footprint and the generated heat.

3.4 Coordinator

Since all the devices are integrated into Home Assistant, any web device can monitor the system status via the Home Assistant web application. By connecting to `https://coordinator.local:8123/` while connected with the same local network, a dashboard (named Lovelace) will be presented. This dashboard gives an immediate overview of all the things inside the network with its sensors and actuators. The Home Assistant dashboard also provides some tools to monitor the status history of all entities.

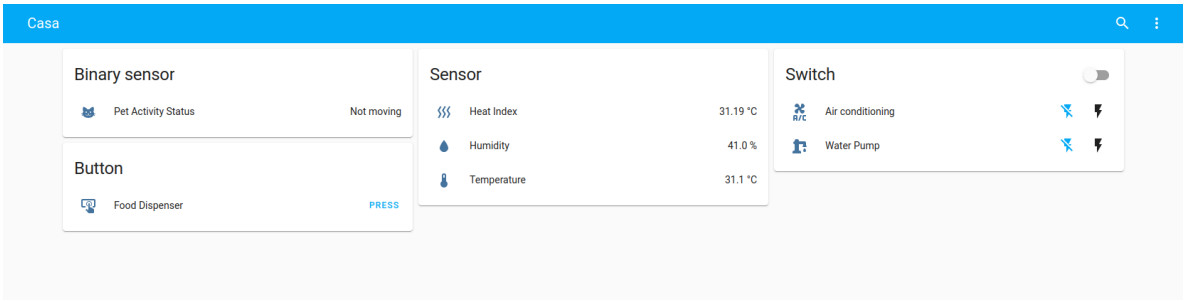


Figure 4: Lovelace interface

4 Automation demos

To provide the automation logic between devices in the network, some example Home Assistant automations are provided below. All automations are configurable via the Home Assistant settings.

4.1 Smart AC Demo

In this demonstration, the Smart AC activates only when a temperature sensor is available and a threshold is reached (average of latest 5m measurement). The AC automatically turns off in case the Smart Collar providing the temperature goes offline or if the target temperature is reached.

4.2 Motion-activated Water Pump Demo

The Smart Feeder pump activates a closed water loop that pets love to drink from. However, keeping the pump active at all times can be noisy and stressful for the pet but also for the owner. In this demo, the water pump activates only if the cat is detected and moving. The pump automatically deactivates when the cat is resting or if it leaves the house.

4.3 Smart Dispenser

The Smart Feeder Dispenser is configured by the user to release some dry food at a certain time of the day. However, to keep the dry food fresh and fragrant, it should be dispensed only if the pet is at home. In this demo, the dispenser gets activated at certain times of the day only if the Smart Collar is available.

5 Future developments

In this sections are presented some possible developments of this project, which could enrich the Smart Room ecosystem.

- **More automations:** The automations presented in the previous sections are just a small fraction of the potential of the system
  - A proximity sensor can be added to the smart feeder to signal via an app notification that the food dispenser is almost empty and that should be refilled
  - Similarly, a water quality sensor could be added to the smart feeder to signal via an app notification that the water inside the dispenser is getting stale and should be replaced
- **Smart Collar**
  - Add a relay to power the Smart Collar sensors only if the device is not in hibernation
  - A LoRa (Long Range) adapter could be integrated into the smart collar to enable the tracking of the pet even outside the house. This would also improve the energy efficiency of the wireless communication
  - With this in hand, a GPS could be used to estimate the location of the pet during the day and relay it back in real time to the LoRa base station at home to keep it monitored in case of emergency