IoT Project: Final Report

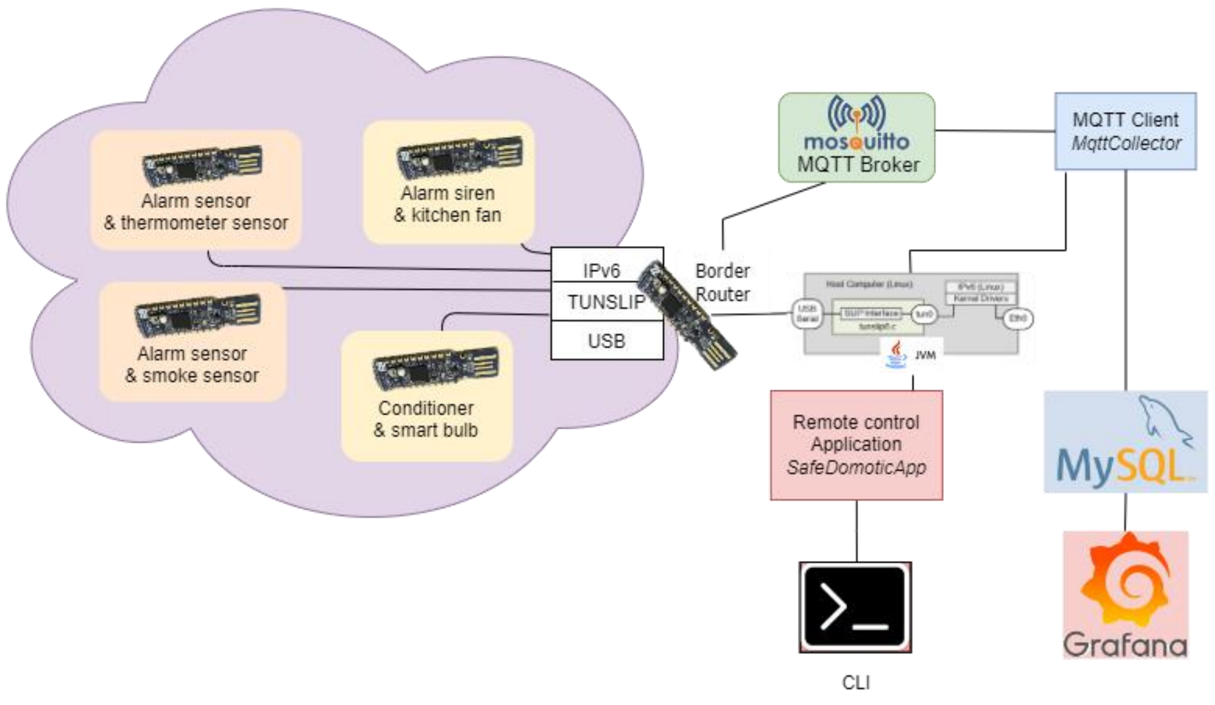
SafeDomoticHome

Lorenzo Ceccanti, matricola 564490, A.A. 2022/23

Introduction:

In this project I have developed an IoT telemetry and control system for a smart home.
This kind of application could be useful to keep the *SafeDomoticHome* warm in winter and cool during summer acting on the A/C system also when you're not at home. This IoT system ensures a greater level of security: the alarm motion sensors and the siren are linked to the IoT system and when an intrusion is detected you can easily notice it. The alarm system is equipped with two zones: the blue and the red one. During the installation of a motion sensor it's possible to select which zone the alarm sensor belongs to. *SafeDomoticHome* also could reduce the risk of fires in the kitchen: a smart fan will automatically turn on if the smoke concentration becomes larger than a certain threshold.

Architecture scheme and deployment:


Alarm sensor & thermometer sensor
Alarm siren & kitchen fan
Alarm sensor & smoke sensor
Conditioner & smart bulb
IPv6
TUNSLIP
USB
Border Router
mosquitto MQTT Broker
MQTT Client MqttCollector
JVM
Remote control Application SafeDomoticApp
MySQL
Grafana
CLI

In violet there's the WSN: in yellow are highlighted the CoAP actuators, in orange are highlighted the MQTT sensors.

Statical CoAP discovery: The list of CoAP actuators is stored in the MySQL database, specifically in the table called `resources`. The table expresses the association between the mote IPv6 address and the resource to which the User Application refers to. I have also made a tiny Java Application that fills the table `resources` after converting the Serial Number of each nrf52840 dongle (put by the user as input in the CLI) in their IPv6 address. Such application can be found in "./statical-coap-discovery/" and it must be launched every time CoAP actuator codes are flashed onto different nrf52840 devices. Important: before launching the statical-coap-discovery application, you have first to run the SQL script provided inside a MySQL instance.

MQTT topics:

Topic name	Who publishes/what is published to this topic
`zone_0`	The MQTT motion sensor programmed to belong to the zone #0 will publish its updates to the Cloud Application/MQTT collector.
`zone_1`	The MQTT motion sensor programmed to belong to the zone #1 will publish its updates to the Cloud Application/MQTT collector.
`temp`	The MQTT thermometer embedded on one of the alarm sensor will publish the updates about temperature in the living room to the Cloud Application.
`conditioner`	The Java class acting as MQTT collector publishes to this topic to implement a close-loop control logic between the thermometer and the Air Conditioning system. The thermometer sensor by subscribing to this topic could react to changes applied on the AC remote (represented by the CLI user inputs).
`voc`	The MQTT VOC sensor, used to detect cooking smokes, will publish the updates about the percentage of smoke in the kitchen to the Cloud Application and to trigger automatically the CoAP fan positioned above the hob.
`fan`	The Java class acting as MQTT collector publishes to this topic to implement a close-loop control logic between the VOC sensor and the smart fan in the kitchen.

CoAP resources:

The following resources are exposed by the conditioner & smart bulb CoAP actuator dongle:

`GET /light`	Retrieves the status of the smart bulb. Parameters: none. Response content-type: `application/json` Code 200: `{"bulbStatus":0}` or `{"bulbStatus":1}` if the bulb is off/on Code 406: `Supporting content-type application/json only`
`PUT /light`	Changes the status of the smart bulb: it will turn on if the int is 1, off is the int is 0 body: `application/json`: `{"command":int}` Response content-type: text/plain Code 200: `"[Light]: ON"` or `"[Light]: OFF"` Code 400: `Bad request` Code 406: `Supporting content-type application/json only`
`PUT /clima`	Implements the smart remote of the Air Conditioning system. Turns ON/OFF the AC, sets the temperature and the speed of its fan. body: `application/json`: `{"power":char[2],"selTemp":int,"fanSpeed":int}` For `power` only `"ON"` or `"OF"` values are accepted. For `selTemp` only values in

	[160, 340] are accepted, for `fanSpeed` only values in [1,5] are accepted. Response content-type: text/plain Code 200: `"[Conditioner]: OFF" or "[Conditioner]: ON, Temperature: int, Fan Speed: int"` Code 400: `[Conditioner]: temp must be in [16,34] and sp in [1,5]` Code 406: `Supporting content-type application/json only`
`POST /clima`	Sets the status of the Air Conditioning system to work as cooler or warmer. body: `application/json:` `{"warming": 0, "cooling": 1} or {"warming": 1, "cooling": 0} or {"warming": 0, "cooling": 0}` Response content-type: text/plain Code 200: `Done` Code 400: `Bad request` Code 406:`Supporting content-type application/json only`

The alarm siren & kitchen fan CoAP actuator dongle expose the following set of resources:

`PUT /siren`	Turns the siren on/off with respect to one or more zones. body: text/plain `zone="zone"&mode="mode"` For the `zone` PUT parameter only the following Strings are accepted: `alarm_0` or `alarm_1`. For the `mode` PUT parameter only the following String are accepted: `ON` or `OFF`. Response content-type: text/plain Code 200: `Done` Code 400: `Bad request`
`PUT /fan`	Changes the speed of the fan positioned above the hob. body: application/json `{"fanSpeed": $int}` Response content-type: text/plain Code 200: `[FAN]: Speed set to $int` Code 406: `Supporting content-type application/json only`

I have chosen to use mainly JSON data encoding due to its simplicity in serialization/deserialization both in C and JAVA language.
In C I have used `sprint()` and `sscanf()` for serialization/deserialization, in Java I have used the GSON library. Also, with respect to other data encoding format, JSON is lightweighted and so most suitable for constrained devices like the nrf52840 dongle that I have used for this project.
JSON is also used as encoding language for MQTT payloads.

Data collection:

Data is collected by the Java Cloud Application and stored in the MySQL database *SafeDomoticHome*
In the database are collected information about:

- The status of alarm sensor attached at the entrance door and at the balcony door:
MOTIONZ0(timestamp: TIMESTAMP, isMagnetOn: INT)
MOTIONZ1(timestamp: TIMESTAMP,isMagnetOn: INT)

- A record for each time the alarm siren rang:
ALARMLIST(timestamp: TIMESTAMP, zone:VARCHAR(45))
- The information about arming, disarming and zone exclusion:
ALARMCMDLOG(timestamp:TIMESTAMP,zone0:INT,zone1:INT)
- The log of thermometer during its sensing activity:
THERMOMETER(timestamp: TIMESTAMP,celsiusDegree: VARCHAR(45))
- The log of VOC meter during its sensing activity
VOCMETER(timestamp:TIMESTAMP,smokePercentage: INT)

Important: Make sure to set the timestamp to UTC:

```
USE SafeDomoticHome;
SET @@global.time_zone = '+00:00';
```

Deployment, use-cases and interaction with dongles:
After deploying the *rpl-border-router*, you need to run the Java Application statical-coap-discovery putting the SN of nrf52840 dongle, paying attention not to confuse a SN for another.
When "Changes saved" appears the static CoAP discovery procedure is completed and you can proceed with next steps.

Deploying the sensors and actuators:
After connecting an alarm sensor to an USB port, the device will wait for a zone selection. Pressing the button will make you select the zone. If only the yellow led is on, a zone has not yet selected for that mote. To select a zone, follow this scheme [fig.1]:

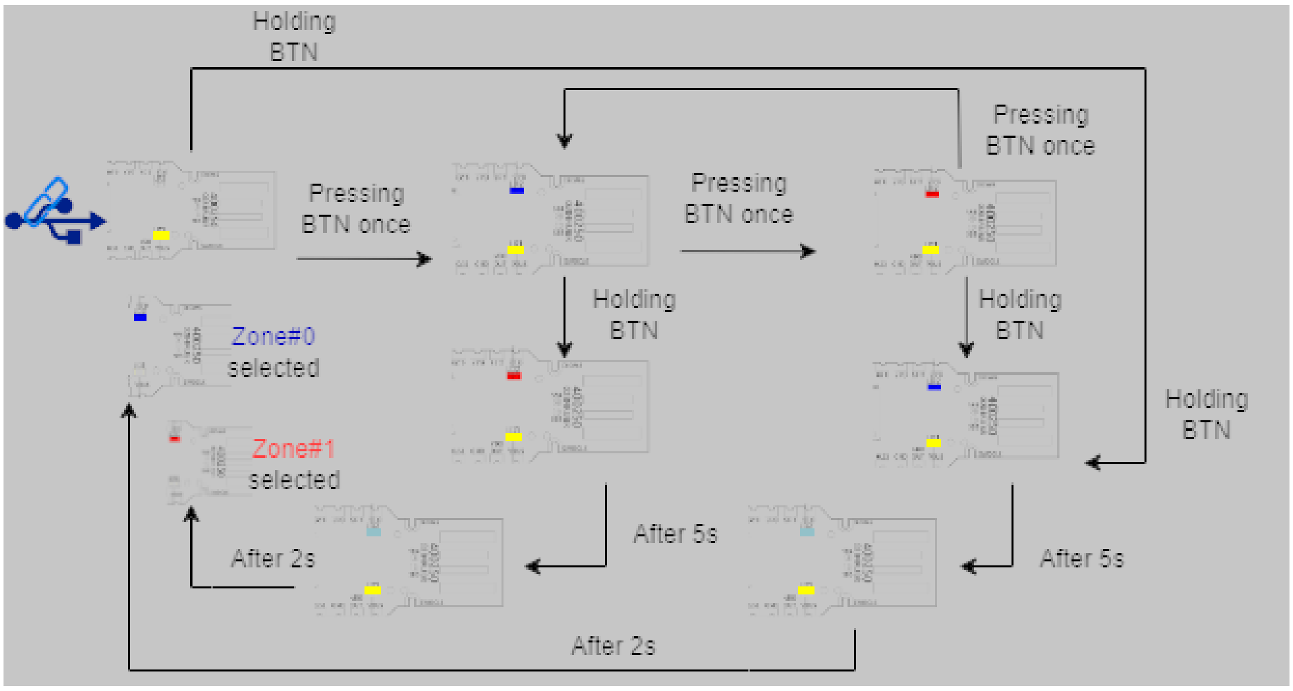


Figure 1

After the zone selection, the embedded temperature/VOC sensors will start to "sense the environment" (they will generate random values emulating their real behavior).
The connection of nrf52840 dongle implementing the actuators will result in:

- the green led turning on for the siren & kitchen fan actuator to indicate that no intrusion is triggered and that CoAP server for the siren and the kitchen fan is now ready.
- the activation of CoAP conditioner and smart bulb (by default are OFF, so no led is colored).

Launching the Java Application:
After launching the SafeDomoticHome Java Application, the list of available commands will be displayed and a connection to the MQTT broker and to the MySQL database will start.

```
***SafeDomoticHome***
-------------------------------------------
Possible commands:
!exit: exits the program
!arm z0 z1: arms one or more alarm zone
Example: !arm Y N -> arms zone0 excluding zone1
!disarm: disarms all the alarm zones
!showzones: gets the current status of all zones
!temperaturecheck: gets the most updated temperature report
!clima ON|OFF [tempDegrees] [fanSpeed]
!pressswitch: invert the light status
-------------------------------------------
```

In the following part, please consider that:
- on the mote where the CoAP siren & fan server is running, the LD2 represents the status of the siren instead the led LD1 represents the status of the fan.
- on the mote where the CoAP conditioner & smart bulb server is running, the LD1 represents the status of the conditioner, instead the LD2 represents the status of the smart bulb.
To distinguish LD1 from LD2 it's worth to look at the mote.

By typing `!exit` you will close the program and this will result also in the end of the autonomous handling of the smart fan in the kitchen.
By typing `!showzones` you can see the current status of the door where the sensor are attached to. Example:

```
-------------------------------------------
!showzones
Magnet Sensors Status at 14:13:03
Zone #0: closed
Zone #1: opened
-------------------------------------------
```

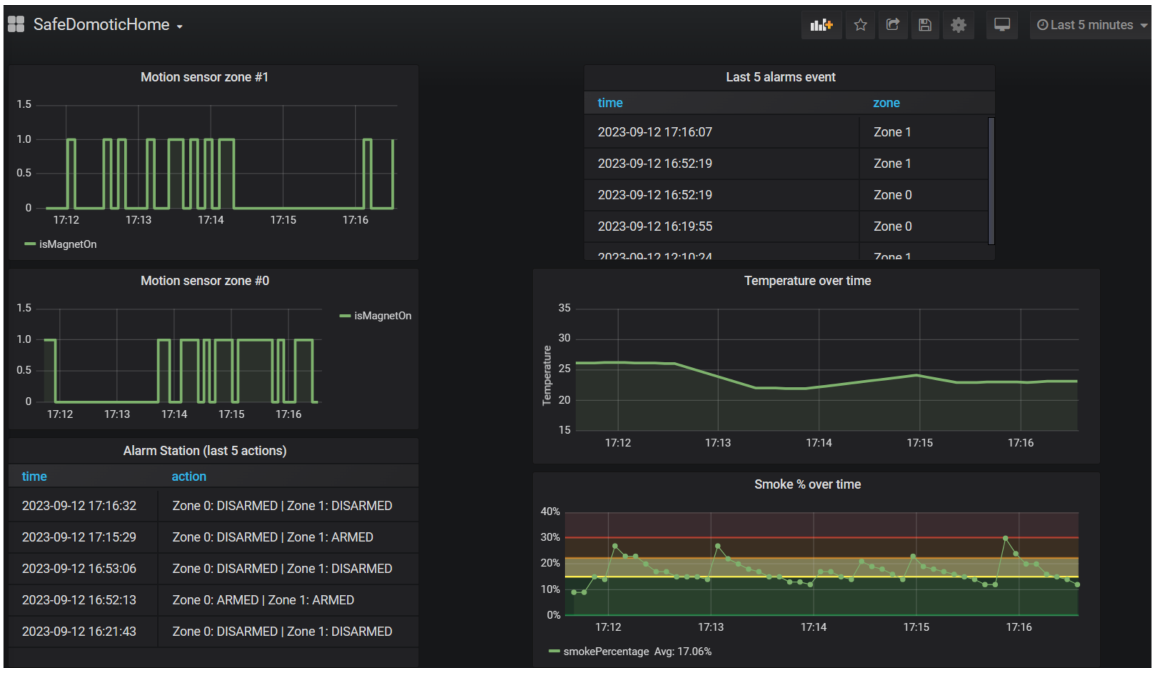
By arming the alarm like is described in the CLI output, sooner or later the siren will be triggered. If you select a higher probability by pressing the button on the alarm sensor 3 times (by default the probability is 20%, so if you press the button 3 times the probability will become 80%) the CoAP siren led corresponding to the zone where the intrusion was detected will turn on. If both red and blue led on the CoAP siren are turned on, this means that the alarm was triggered in both the zones. To disarm the alarm, type the command `!disarm` and the alarm will be disarmed: the siren mote will display the green led again. To exclude a zone from arming, make sure that the alarm is disarmed first. Example:

```
-------------------------------------------
!arm N Y
First disarm ALL the zones!
-------------------------------------------
```

The other commands are auto explicative. The `!clima ON` command only accepts values for the temperature in the interval [16.0, 34.0] and the fan speed must be in [1,5].

While the Java Application is running, a modification to the kitchen fan actuator is made basing on the data collected from the VOC sensor. By default, the probability that a smoking fume is detected is very low. To rapidly increase this probability, keep press BTN on the VOC sensor for few seconds. The change of this probability is displayed by the yellow led turning on.

The actions made by the smart fan in the kitchen could be observed in the Grafana Dashboard together with some other summary information [fig.2]:



Last 5 alarms event

time	zone
2023-09-12 17:16:07	Zone 1
2023-09-12 16:52:19	Zone 1
2023-09-12 16:52:19	Zone 0
2023-09-12 16:19:55	Zone 0
2023-09-12 12:10:24	Zone 1

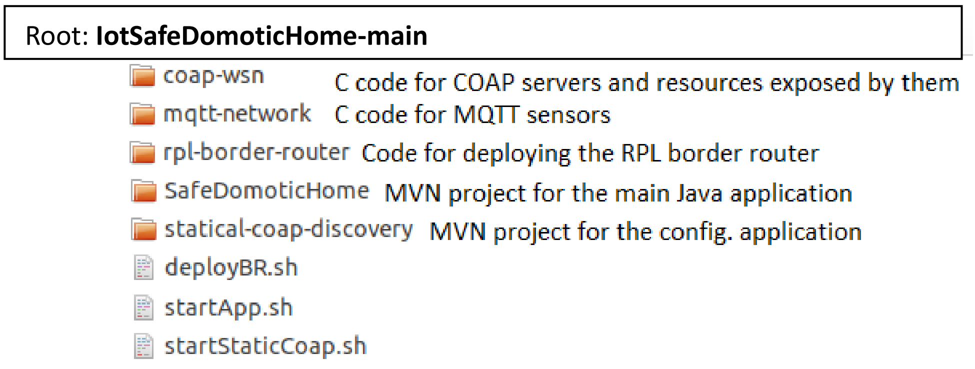
Alarm Station (last 5 actions)

time	action
2023-09-12 17:16:32	Zone 0: DISARMED | Zone 1: DISARMED
2023-09-12 17:15:29	Zone 0: DISARMED | Zone 1: ARMED
2023-09-12 16:53:06	Zone 0: DISARMED | Zone 1: DISARMED
2023-09-12 16:52:13	Zone 0: ARMED | Zone 1: ARMED
2023-09-12 16:21:43	Zone 0: DISARMED | Zone 1: DISARMED

Figure 2

For the smoke percentage, you can see that 3 threshold are defined:
<15%: good. The smart kitchen fan does not intervene. The yellow led is off.
between 15 % and 22%: The smart kitchen fan intervenes with speed 1, the yellow led on the fan mote will blink slowly (every 3 seconds).
between 22% and 30%: The smart fan will change its speed to 2, the yellow led on the fan mote will blink faster (every 2 seconds).
>30%: critical level. The smart fan will change its speed to 3 (its maximum speed). The yellow led on the fan mote is on without blinking.

Code organization:



Root: **IotSafeDomoticHome-main**

coap-wsn	C code for COAP servers and resources exposed by them
mqtt-network	C code for MQTT sensors
rpl-border-router	Code for deploying the RPL border router
SafeDomoticHome	MVN project for the main Java application
statical-coap-discovery	MVN project for the config. application
deployBR.sh	
startApp.sh	
startStaticCoap.sh	