Smart Cities and Internet of Things

Group 6 : A Smart Environment Management System for University Lecture Rooms

July 20, 2018

1 Introduction

Naive Energy Management in Lecture Rooms of Universities can lead to huge unnecessary expenses in energy costs while at the same time impacting the lives of university staff and students in a negative way.

In this project we built a model for a smart lecture room management system which regulates room temperature and saves energy, when the room is not used. In order to make the project comprehensible the complete documentation can be found online[1].

For the model we used the following hardware components:

- 1x Aeontec `Z-Stick`
- 1x `Multisensor 6`
- 1x Plugwise Set (`Plugwise Stick`, `Circle` and `Circle+`)
- 3x `Raspberry Pi 3B+`

2 Lecture Room Model & Assumptions

In order to model the lecture room, we used a cardboard box, the box can be heated up using a heat blow tool, it can be cooled down using a laptop cooling pad, both of these actuators are turned on or off depending on the current state of the System. Inside the box we have two sensors, each of them can collect various measurements. In order to perform reasoning over the systems state, we are interested in the rooms temperature, the luminescence and the state of our actuators.

The heating system can blow hot air through a hole on the side of the box, the cooling system sits on top of the box and can blow the hot air out, which is then replaced by cold air from the hole on the side.

We also made the following closed world assumptions:

- It can either be day or night (depending on the level of light)
- At day the room ls always in use
- At night the room is never used
- When the room is in use energy does not have to be saved
- When the room is not in use, energy saving is active
- The temperature of the room can be too hot

[1] https://github.com/sciot/group6

- The temperature of the can be too cold
- The heat blow tool can be turned on or turned off (and also be kept on or off)
- The cooling pad can be turned on or turned off (and also be kept on or off)
- The systems current state can be used to compute the actions to be taken in the next timestep
- The fact, that our timesteps are not equidistant can be neglected

3 Architecture

3.1 High level Overview

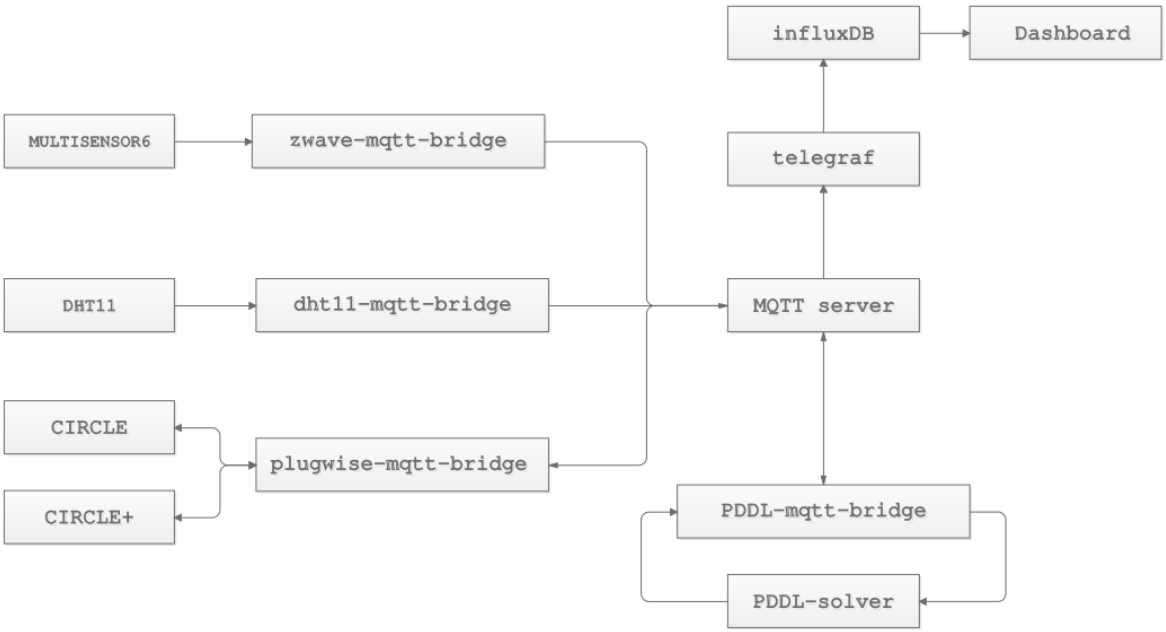

Figure 1: Overview of the projects software components

At the center of our architecture we use a `MQTT` server as messaging middleware for communication between the running components. Messages are sent - depending on their data format - to the topics `sciot/json` or `sciot/influx`. In order to connect the components to `MQTT`, we use the following software components:

- `zwave-to-mqtt-bridge`
- `dht11-to-mqtt-bridge`
- `plugwise-to-mqtt-bridge`
- `PDDL-to-mqtt-bridge`

All of these bridges are written by us in python, with the exception of `zwave-to-mqtt-bridge`, which is a modified version of the eponymous python package available via `pip`. In order to visualize the metrics we collect, we use the well known TICK[2] stack to store and display the measurements.

The reasoning and decision making in the system is taken care of, by the `PDDL solver`: We use the `PDDL-mqtt-bridge` to gather information on the system and to create a `PDDL` Problem Instance describing the

[2] `telegraf`, `influxDB`, `chronograf`, `kapacitor`

current state of the system. The `PDDL solver` then executes a fast forward search, which results in a sequence of steps to be taken in the next timestep. The solution to the `PDDL` problem instance is then applied to the system by the `PDDL-mqtt-bridge`, resulting in new measurements in the future that can be reasoned about.

3.2 Architectural Concerns and Network Overview

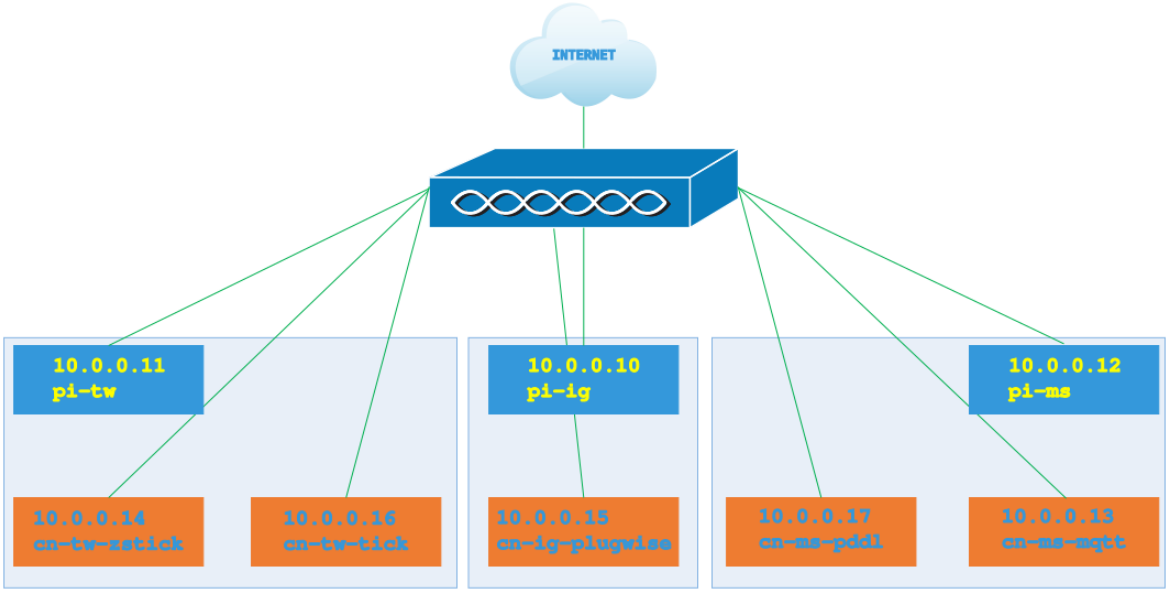

Figure 2: Network Overview of the System: We used a flat network topology, the dark blue boxes are the raspberry pis, the orange boxes are containers.

We used `LXD` as underlaying container technology to run all but one of the services[3]. The containers as well as the raspberry pis run in a flat network topology in the `10.0.0.0/24` subnet and use fixed `IP` addresses. Initially we started out by making everything run on one raspberry pi and later when all components were working, we scaled out to three pis by simply plugging the sensors into the new pis and then moving the containers from the initial host to a new one and starting them, in total we moved 2 sensors and 4 containers to 2 new hosts: All in all we had less than 2 minutes downtime and after we moved, everything just continued working exactly as it did before.

In fact using containers to bundle our software components in logical units has (only to name a few) the following advantages over installing directly on the host system:

- **Scalability**: You can scale out vertically as well as horizontally easily by just adding more hosts or copying containers.
- **Provisioning**: After provisioning containers they can be exported to images, which then can be easily installed everywhere in the network
- **Updates & Rollbacks**: Updating a software component becomes easy, just replace the container and if anything goes wrong roll back to the previous version
- **Security and Privacy**: using `LXD`, containers run in unprivileged mode and it can be controlled exactly which devices the containers can access. Also if an attacker breaks out of the container, he will become a process with non-existing `UID/GID` and as such has no way of interacting with the host system

[3]the DHT11 uses gpio pins and currently runs on bare metal

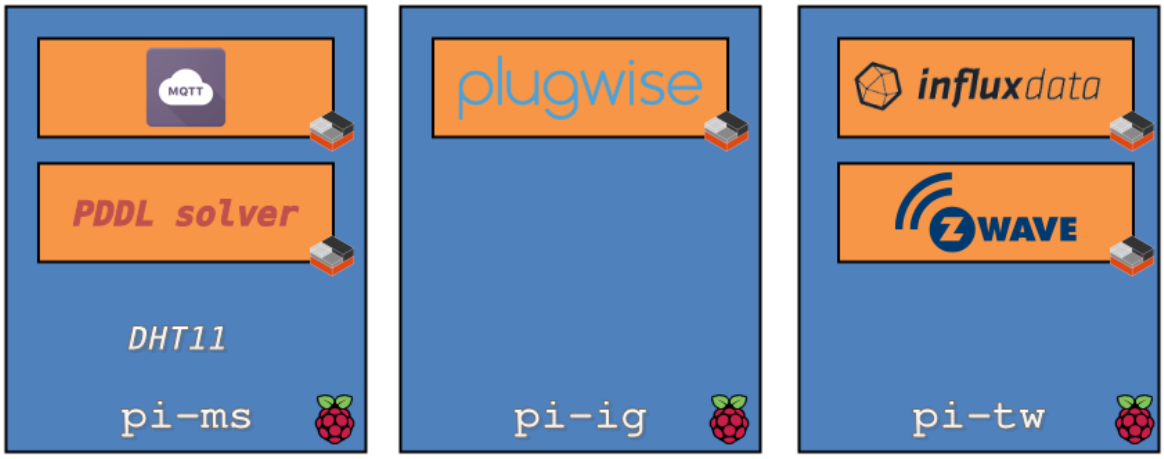

Figure 3: Logical Overview: We use 3 raspberry pis and have bundled all but one software components into `LXD` containers.

4 PDDL - Reasoning about what to do next

We modeled the domain we used the close world assumptions from the beginning of the document.

As already stated, we create Problem Instances on the fly and applied their solutions back to the system.

4.1 Introducing Considerations

In order to make sure that `PDDL` always considers everything important to our system, we also introduce 'Considerations' into our domain: Considerations are values that are initially false and only set to true if the consideration about what to do in the current state has been made for the considerations context.

We use one consideration each in order to enforce checks on the systems energy saving behavior, the heating system as well as the cooling system.

In our problem instances we describe the current state of our system and then simply state that we want all considerations to be met in the goal. PDDL then finds a solution to what is to be done with the system, regarding energy saving, heating and cooling.

5 Reflection

Initially we set out to create a model for a lecture room that considered energy saving, the rooms temperature and the rooms humidity in order to create a healthy environment for students and university staff alike.

After the planning phase however we decided to ditch the humidity part of the project[4]. Other than that we pretty much accomplished everything we set out for, we created a small but scalable system that uses reasoning in order to compute its next steps. So if we could do it all over again, we would build something quite similar.

If we had to continue from were we are right now, then we would focus on two areas: First of all, we would create a bigger system, with more actuators and more states than only day or night in order to be able to model a more complex problem domain and remodel our domain using the temporal extension introduced in `PDDL 2.1`. Secondly, we would take a closer look into how cloud technologies can benefit IoT applications, inter alia how the separation of logical components using containers can benefit the system by working out ways of automatically provisioning and setting up new containers, when sensors are added to any host in the system.

[4] because it would have literally meant spraying water on open computer parts

```
(define (domain sciot_room)

  (:predicates (timeofday ?tod)
               (consideration ?cn)
               (temperature ?t)
               (heater  ?h)
               (cooling ?c)
               (heater_turned_on  ?h)
               [...]
               (energy_considered ?cn)
               (heater_considered ?cn)
               (cooling_considered ?cn)
  )

  (:action turn_on_device
      :parameters (?h ?t ?tod ?cn)
      :precondition (and (heater ?h)
                         (temperature ?t)
                         (timeofday ?tod)
                         (temperature_too_low ?t)
                         (not (heater_turned_on ?h))
                         (is_daytime ?tod)
                    )
      :effect (and (heater_turned_on ?h) (heater_considered ?cn))
  )

  (:action keep_running_device
      :parameters (?h ?t ?tod ?cn)
      :precondition (and (heater ?h)
                         (temperature ?t)
                         (timeofday ?tod)
                         (temperature_too_low ?t)
                         (heater_turned_on ?h)
                         (is_daytime ?tod)
                    )
      :effect (and (heater_turned_on ?h) (heater_considered ?cn))
  )

  [...]
```

```
[...]

(:action turn_off_device
      :parameters (?c ?t ?tod ?cn)
      :precondition (and (cooling ?c)
                         (temperature ?t)
                         (timeofday ?tod)
                         (not (temperature_too_high ?t))
                         (cooling_turned_on ?c)
                         (is_daytime ?tod)
                    )
      :effect (and (not (cooling_turned_on ?c)) (cooling_considered ?cn))
)

(:action keep_device_turned_off
      :parameters (?c ?t ?tod ?cn)
      :precondition (and (cooling ?c)
                         (temperature ?t)
                         (timeofday ?tod)
                         (not (temperature_too_high ?t))
                         (not (cooling_turned_on ?c))
                         (is_daytime ?tod)
                    )
      :effect (and (not (cooling_turned_on ?c)) (cooling_considered ?cn))
)

(:action energy_saving_inactive
      :parameters (?cn ?tod)
      :precondition (and
                         (timeofday ?tod)
                         (is_daytime ?tod)
                         (not (energy_considered ?cn))
                    )
      :effect (energy_considered ?cn)
)


[...]
```

Figure 4: Excerpt of the PDDL domain description: `considerations` are defined in the `predicates` section and later set to `true`, only if the `consideration` has actually been respected.

```
(define (problem sciot_room_too_cold)
   (:domain sciot_room)
   (:objects temp heat_blow_tool cooling_pad now
     cn_energy cn_heater cn_cooling)              ; <- considerations

   (:init
     (heater heat_blow_tool)
     (cooling cooling_pad)
     (timeofday now)
     (is_daytime now)
     (temperature temp)
     (temperature_too_low temp)
     (cooling_turned_on cooling_pad)
     (heater_turned_on heat_blow_tool))

   (:goal (and (heater_considered cn_heater)
               (energy_considered cn_energy)
               (cooling_considered cn_cooling))))
```

Figure 5: A sample PDDL problem instance for our domain: The current state of the system is described and the goal is to fullfill all considerations in the process.