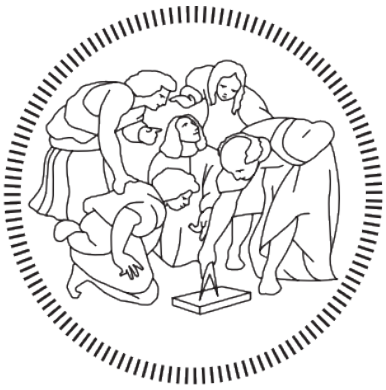

POLITECNICO
MILANO 1863

Networked software for distributed systems

A Simple Smart Home

Authors:
Stefano Carraro
Stefano Fossati
Andrea Restelli

Professors:
Prof. Luca Mottola
Prof. Alessandro Margara

2022-2023

Contents

1 Introduction

1.1 Request

1.1.1 Description

In a smart home, a control panel provides a user interface to coordinate the operation of several home appliances, such as HVAC systems, kitchen machines, and in-house entertainment, based on environmental conditions gathered through sensors. Users input to the control panel their preferences, e.g., the desired room temperature. Based on information returned by every appliance, the control panel offers information on the instantaneous energy consumption over the Internet. You are to implement the software layer for the smart home using the actor model. Every appliance may come and go depending on its operational times. The processes in charge of managing every appliance may also crash unpredictably. You need to demonstrate at least three example executions that include:

1. the user inputting some preferences,
2. sensors producing some (dummy) values, and
3. appliances changing their behavior based on 1. and 2.

The three example executions must be able to operate in parallel in two or more rooms of the same smart home and tolerate process crashes.

1.1.2 Assumption and Guidelines

1. Although processes can crash, you can assume that channels are reliable.
2. You are allowed to use any Akka facility, including the clustering/membership service.
3. Sensors, appliances and the control panel may be emulated via software with dummy data.

1.1.3 Technologies

Akka

1.2 Additional Assumption

If the user interface for inserting commands into the application crashes it is manually restarted.

2 Design

2.1 Application Architecture

The following figure shows the components of the application and their interconnections. Since Akka relies on the Actor Model, each element represented below correspond to an Actor in our application in order to exchange messages.

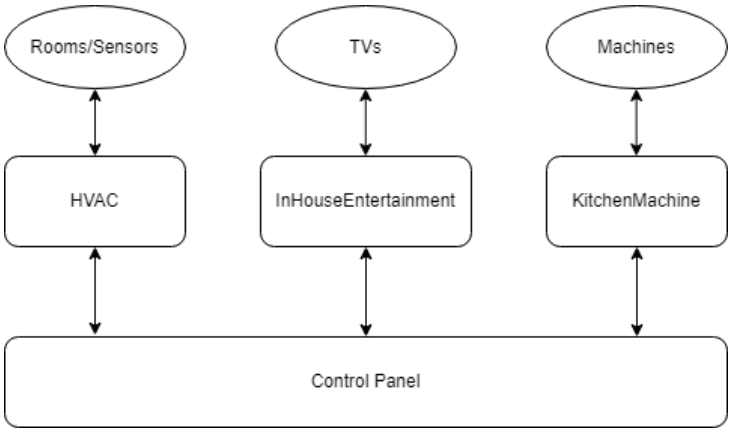

Figure 1: Application Architecture

From the Control Panel the user can perform the following operations:

- Create a new Room, TV, Machine or remove an existing one
- Show Sensors, TVs or Machines connected
- Set the desired temperature in a room
- Turn on/off TVs or Machines
- Request the total energy consumption
- Make a Sensor, TV or Machine crash
- Make HVAC, InHouseEntertainment or KitchenMachine crash

After the user select what to do a message will be sent to the respective HVACActor, InHouse-EntertainmentActor, KitchenMachineActor that will handle the request.
From now on we will refer as ServerActor to HVACActor, InHouseEntertainmentActor or KitchenMachineActor. ClientActor will refer aswell to ClientActorHVAC, ClientActorIHE or ClientActorKM.

2.2 Control Panel Structure

The Command Interface serves as the user interface for inserting commands into the application. The structure of the Control Panel is the following:

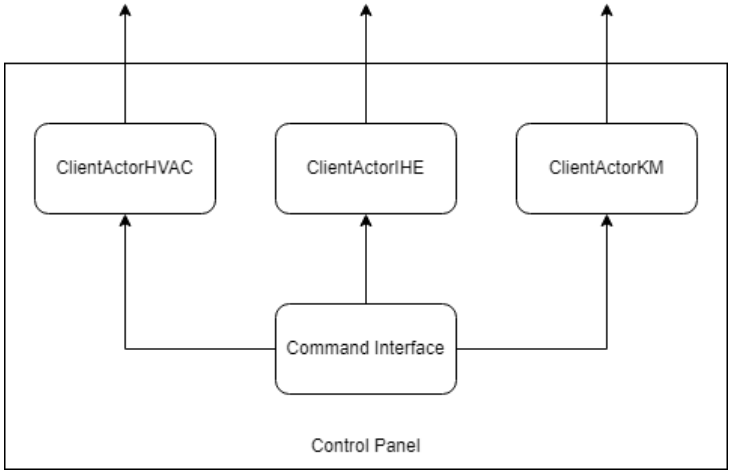

Figure 2: Control Panel Structure

The inserted command is then delivered to its respective ClientActor, which initiates the operation by communicating with the ServerActor.
It would've worked aswell with a single client actor, but we've preferred to split them by areas of interest and threat them separately: this allows an easier scalability if a new feature with new devices needs to be added.

2.3 Achieve Fault Tollerance

It's necessary to define a supervisor and a strategy in order to handle process crashes. In this case the supervisor of the devices is the ServerActor itself:

- the HVACActor is the supervisor of all the sensors.
- the InHouseEntertainmentActor is the supervisor of all the TVs.
- the KitchenMachineActor is the supervisor of all the machines.

Each ServerActor is then supervised by another supervisor. Resulting in the following hierarchy:

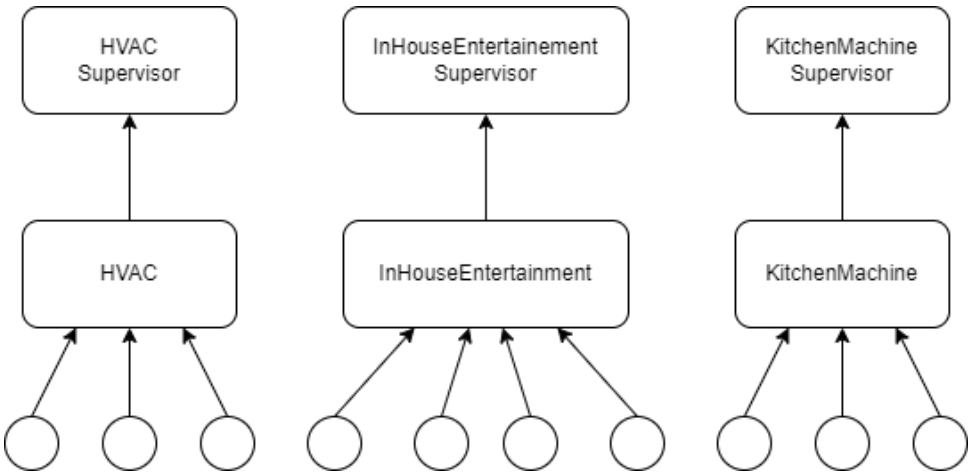

Figure 3: Supervisors Hierarchy

The strategy adopted is the same by all of the three supervisors. When a device crashes it throws a CustomException and its supervisor apply the following strategy:

```
private static SupervisorStrategy strategy =
        new OneForOneStrategy(
                10,
                Duration.ofMinutes(1),
                DeciderBuilder.match(CustomException.class, e ->
                    SupervisorStrategy.resume())
                        .build());

@Override
public SupervisorStrategy supervisorStrategy(){
    return strategy;
}
```

The strategy is applied only on the crashing device and will try to resume it 10 times within a minute.

3 Implementation

3.1 Starting Servers

When a server is started it creates its own system, reading its corresponding configuration file. To achieve the hierarchy described above it is necessary to create the supervisor first and then delegate it to create the ServerActor, this because of when an Actor create a new one.

3.2 Device Creation and Removal

When a new device is inserted in the application it is necessary to instantiate it as a new Actor. Let's assume we want to add a new television, and its ID is "TV1".
In this case, a message (CreateDeviceMessage) is sent from the Control Panel to the InHouse-EntertainmentActor. The ServerActor checks that the ID is unique and creates a new actor. The ActorRef, the reference to that actor, is saved so that the ServerActor knows where to send messages when required.
When we want to remove a device, the corresponding ServerActor checks for the ID of the requested device, retrieves its ActorRef, and proceeds to stop the corresponding actor. This ensures that there won't be any issues if a device is removed while it is active. The ServerActor then unsave that ActorRef.

3.3 HVAC and Sensors Implementation

When a new room is added, the temperature is randomly set between 10.0°C and 25.0°C. The HVACActor keeps track of all the room temperatures, while sensors only retrieve values.
To emulate this process and generate realistic data, sensors need to remember their state, but that value is only used to produce data. This because to modify the temperature retrieved sensors have to work also as actuator (e.g., heat pump). Once a desired temperature is requested for a specific room, the HVACActor saves the desired temperature value for that room, and the temperature increases or decreases until it matches the requested value.
The following steps outline the process:

1. The HVACActor instructs the sensor to increase or decrease the temperature.
2. The sensor schedule a task that modifies the temperature every 3 seconds based on the previous request and send the calculated value to the HVACActor.
   The temperature is increased or decreased by a $dT = 0.1$°C.
3. The HVACActor compare the current temperature value with the desired one. If they're equal then it notices the sensor to stops, otherwise it lets the sensor continues.

If the desired temperature changes before its value is reached, the HVACActor notifies the sensor only if it is necessary to switch the operation applied (e.g., if the value was increasing, it will start decreasing).

3.4 InHouseEntertainment and KitchenMachine implementation

InHouseEntertainment and KitchenMachine operates under the same logic. TVs and Machines can be turned on and off. When a new device is inserted it is set by default to off. Once a device is turned on, as in HVAC, it schedule a task that send message to its corresponding ServerActor.
When the device is turned off, the corresponding ServerActor instructs the device to change its state and cancels the task. When the device is inactive, it does not send any messages to the ServerActor. This allows for proper functionality even in a scenario where the devices can be turned on using their physical buttons and not just from the control panel.

3.5 Energy Consumption

It is necessary to keep track of the energy consumption. We assume that each device of the same type consumes the same amount of energy in the same amount of time.
The following table shows the energy consumption values assigned to each device:

Device	*dE*
Sensor	*10W*
TV	*1W*
Machine	*5W*

When each of the ServerActor receive a message from an operative device (the one sent during the task that we discussed before) it just increase the energy consumed by the respective *dE* value. For example, if a room is at 23.0°C and we desire a temperature of 24.0°C, it will send a total of 10 messages before the server stops the task, resulting in a total energy consumption of 100W.