Smart Home System - Akka

Bruno Morelli Cristian Lo Muto Vincenzo Martelli

Contents

1 Introduction

This document is the documentation of the Akka Smart Home application, which purpose is to provide a comprehensive overview of the project. 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.
Every appliance may come and go depending on its operational times. The processes in charge of managing every appliance may also crash unpredictably.
The goal of this project was to implement the software layer for the smart home, this has been achieved using the actor model.

2 Approach and Assumptions

The actor model fits perfectly the need of this project because every device, appliance and sensor can be viewed as an actor. Furthermore, Apache Akka implementation of the actor model provides an easy way to implement communication between the various actors, handle actors faults and test the system in both a local and a distributed environment. The system was projected in such a way that it can work in parallel in the various rooms of the home, this was done with ease by implementing the rooms as actors. A relevant assumption is the fact that only the panel to coordinate the smart home and the server that receive the command are configured to work in a distributed setting. All the appliances and the sensors are simulated on the same machine of the SmartHome, this assumption holds quite well since in a real scenario they are usually not connected to the internet directly, but they use a bridge. In our system the SmartHome class represents the bridge. There are two implemented appliance and sensor which purpose is to demonstrate that the system actually work, but it is easy to add new type of sensor and appliance. It was assumed that every time the user adds an appliance the relative sensor is also added. Another assumption made was about the operative time of the appliance, it was asked for the systems to allow appliances to come and go according to their operative times. This was implemented by having the appliance going off when it uses too many watts, simulating an overheat, and coming back again after ten messages arrived from the sensor. This seamed a good way to simulate the operative times of the appliances without having to implement timers.

3 Design

3.1 FrontEnd

3.1.1 PanelMain

The PanelMain class is simply a command line interface that can be used from the user in order to create the message with the parameters of the command he want to set and to send them to the Panel.

3.1.2 Panel

The Panel class on the client side behaves as an actor to receive commands from the user and forward them to the SmartHome server. It includes methods such as addRoom, addDevice, setPreference, and simulateCrash. These methods handle commands related to adding rooms, devices, setting preferences, and simulating crashes for testing purposes. All request are forwarded to the SmartHome using the "ask pattern", so the communication is synchronous.
Naturally, to add a device, you first need to create a room where to place it. Similarly, to configure a preference in a specific room, you need to have that particular type of device already present in that room.

3.2 BackEnd

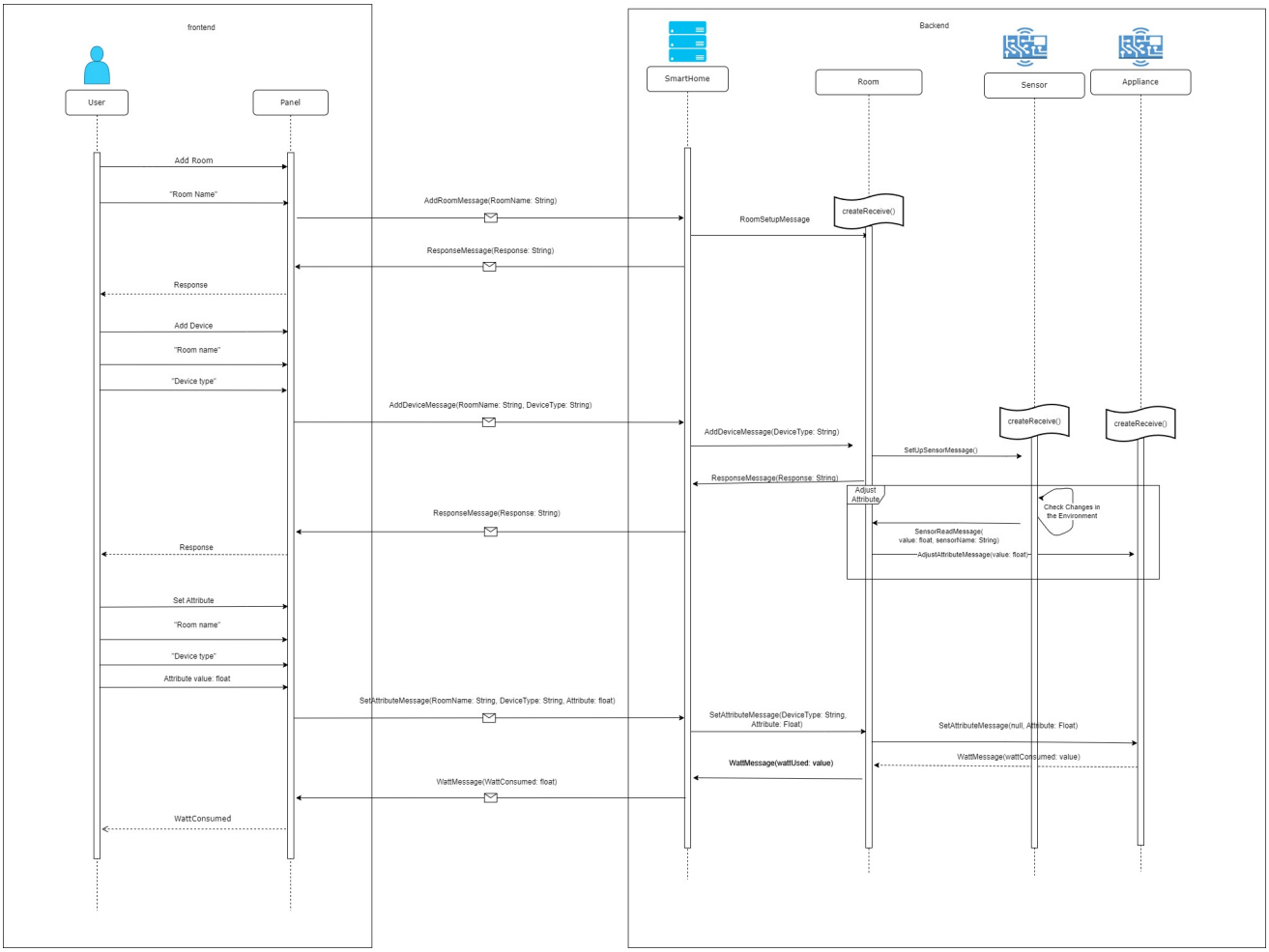
The backEnd of this system is composed of four types of actors:

- **SmartHome:** The main actor of the backend, it is the one that has the duty of create the rooms and forward the messages coming from the frontend to them. It also stores information on the quantity of watts used from the entire system.
- **Room:** Every room has the reference of its appliances and devices, it sends the requests coming from the smart home to the appliances and sends back the energy reading. Furthermore it has to listen for changes in sensors readings and report these values to the appliances.
- **Sensor:** Every type of sensor has its own class, it just need to have a loop that starts when receives the first setUp message and send back the reading when it is needed.
- **Appliance:** An actor that perform a job every time it receives a request with a preference or a change in the environment.

The communication between these actors happens mostly using the 'ask pattern', e.g. the smart home asks the room to set a preference on an attribute, the smart home asks the appliance to do the work, the device answers with the watt used to do this and the the room answers the smart home with the total watts used and the home then reports the cumulative watt usage to the frontend. An exception is when the appliance has to perform a job in response to a change in the environment, that is reported from a sensor to the room. The room has to tell the watt usage to the smart home without this being an answer to a request, these are the only time a message goes bottom up on its own.

3.3 Interaction

Here there is a sequence diagram that shows how the two sides interacts. In this diagrams the user creates a room, adds an appliance and sets a preference.



The remark the difference between the two systems they are inserted in two boxes and the messages sent back and forth have a little 'mail' image to indicate that they travel on the network. Consistently with the assumptions written above the sensors and appliances have been placed on the same system of the SmartHome. The 'adjust attribute' box represents the continues cycle of the sensor getting reads communicate them to the Room and then the appliance doing the work to stay on the target, this happens continuously and asynchronously. The appliance and its related sensor are created at the same time by the room but only the sensor needs a setup message while the appliance will be ready to do its job when it receives the first message with a preference.
In order to simulate operational time of appliances a threshold value for the used watt is inserted. When they go over this threshold they "overheat" and go in an *off* status. When the appliances

are off they discards the messages that arrive to them, at the tenth message they become *on* again. This was implemented using the *become()* method offered by actor to change their behavior, to increase fidelity it was decided not to stash messages.

3.4 Fault tolerance

While running, both the Rooms (that is the process that manage every appliances) and the Appliances can fail. If a Room fail its supervisor (SmartHome Actor) apply a resume Supervisor's strategy, meaning that the room will be restarted recovering its state. For the Appliances it works in the same way, but in this case the supervisor of the appliances is the Room in which it is located. The user can test this behavior by sending a crash message from the panel. The following picture show the hierarchical relation of supervision between actors.