Embedded Systems and Internet-of-Things
-
Third Assignment

Kimi Osti

February 19, 2025

Contents

Chapter 1

System Requirements

The system is a smart IoT-based temperature monitor. In particular, it measures a closed environment's temperature at any given time, and controls a window connected to a motor to properly ventilate the room in case of critical temperatures. It also implements a manual mode, which can be activated via a button on the *Window Controller*, or via a web-based *Operator Dashboard*, that allows an operator to physically control the window opening angle thanks to a potentiometer attached to the *Window Controller*.

1.1 Temperature Monitor

It's the main system component. It periodically measures the room's temperature, and communicates it to the *Control Unit*, which is responsible of controlling the other component's behavior according to the valued registered by this subsystem. The frequency of the measurements depends on the state of the system, which is stored in the *Control Unit* and is communicated to this component in real-time. It's connected to the *Control Unit* via the *MQTT* protocol, and must include a LED signaling whether the connection is properly established.

1.2 Window Controller

It's the in-place operator interface. It's responsible of physically triggering the window movement, according to the values communicated by the *Control Unit* if the system is in automatic mode, or according to the value controlled by the potentiometer if the system is in manual mode. It has a button to switch between these two modes, and it also has a screen that tells the

operator the state of the system at any given time. All necessary info is communicated by the *Control Unit* via Serial communication.

1.3 Operator Dashboard

It's a web-based user interface that allows operators to work remotely on the system. It shows a graph representing the current state of the system and a brief history of the last measurements, sided by a statistic showing the average, minimum and maximum values in the last period of time. In addition to this info, it exposes a simple operator interface which allows the user to switch between manual and automatic mode, as well as to restore the system status in case an alarm was triggered. It communicates with the *Control Unit* via the *HTTP* protocol in order to reflect user actions on the actual in-place subsystems.

1.4 Control Unit

It's the core of the entire system, and it serves as a mediator between subsystem interactions. Its main function is to track the system state and all info related to the measurements and to the current operating mode (manual or automatic). It also determines the sampling frequency for the *Temperature Monitor* according to the current measure, and the window opening percentage according to the system state (if in automatic mode). It's also responsible of storing all system data, including the last measurements that the Dashboard shows in its graph, and the average, minimum and maximum values that are shown to the operator.

Chapter 2

System Architecture

This system can be structured dividing responsibilities according to the MVC architectural pattern.
In particular, the Controller behavior is implemented by the **Control Unit**, while the main View component interfacing towards the user would be the *Operator Dashboard*. The *Temperature Monitor* is fully a Model component, since it samples the real-world status value and shares it with the *Control Unit*. Finally, the *Window Controller* can be considered to be almost completely Model, with a part of View in the Control Panel, which allows the user to switch mode between manual and automatic, and to control the window angle when in manual mode.

2.1 Temperature Monitor

This subsystem's main feature is sampling the room's temperature with a given frequency. It's also responsible for sending that data to the *Control Unit*, so it must include a component handling the subsystem connectivity to the back-end.
Its detailed behavior can be described analyzing each sub-component with its responsibilities. Each sub-component can be modeled as a Task, and all Tasks run in parallel in the subsystem on which this part of the application is deployed.

2.1.1 Temperature Measuring Task

The central Task is the one responsible of actually measuring the temperature. It can be modeled as a Synchronous Finite State Machine.

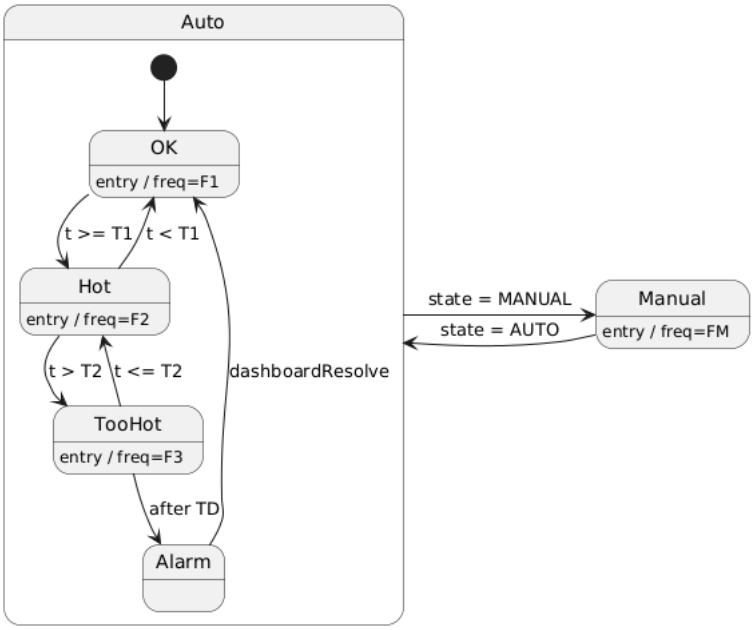

Figure 2.1: FSM modeling the temperature measuring task behavior

It is clearly modeled - for clarity - how the state transitions in automatic mode are determined by the temperature values. But in reality, all state transitions are demanded by the *Control Unit* according to the values that it receives from all components, since it is the central Controller for the application.
Also, it is clearly modeled to be network-agnostic. Indeed, this sub-component is meant to measure at all times, and it will be a responsibility of the network-related tasks to retrieved the stored data when it's supposed to be shared.

2.1.2 Connection Monitoring Task

This task is the one responsible for monitoring the system connection state. It's supposed to check whether the component is still online, and when it's not, to try and reconnect. It works on two layers: the first one checks whether the system is connected to the Internet, and the higher one checks whether the *MQTT* subscription is still on.

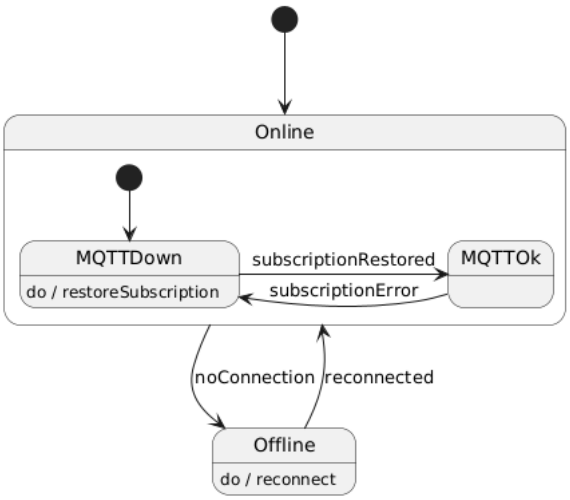

Figure 2.2: FSM modeling the connection monitoring task behavior

2.1.3 Communication Task

This task is responsible of communicating with the *Control Unit* via the *MQTT* connection set up by the *Connection Monitoring Task*. It ensures that data is properly collected, assembled to form a message and sent to the *Control Unit*. On the other hand, it's also responsible of receiving the response messages published by the *Control Unit*, which can dictate the *Temperature Measuring Task* state, and consequently its sampling frequency.

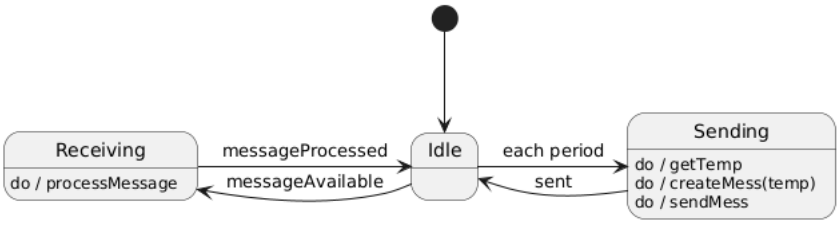

Figure 2.3: FSM modeling the communication task behavior

It's here clear how this task too is network agnostic: simply, when there is no connection or no active *MQTT* subscription no messages are received - and the system therefore never goes to the *Receiving* state - and the ones that are sent might be lost.

2.1.4 LED Task

This task's main responsibility is to represent to the user whether the sub-system is online.

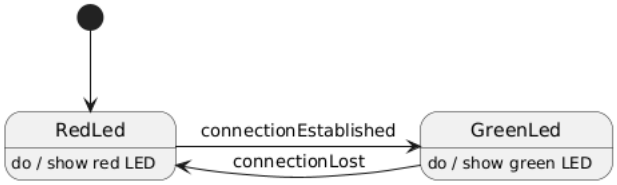

Figure 2.4: FSM modeling the LED task behavior

In this scheme, it's obvious how this Task's state depends directly on the state of the *Connection Monitoring Task*. In particular, the LED is shown as green only when the device is fully connected to the Internet and the *MQTT* subscription is running properly.

Note the sub-system's connection state is kept internally, and is not communicated to the *Control Unit*. This because, architecturally speaking, the rest of the system is not concerned about this specific sub-system's connection state, since it would be simply cut out of the communication network. So, connection state is here only tracked to help the LED Task show the user the correct information, but the rest of the system - if well decoupled - is required to keep running even if the communication went down (or theoretically speaking, also if there were multiple devices tracking the temperature and connected to the same *MQTT* topic and broker).

2.2 Window Controller

Also the Window Controller can be modeled as a Finite State Machine. Its behavior depends on whether the system is in automatic or manual mode, independently from the Temperature Monitor Measurements. Since it exposes a control panel, as a Model component, it's not properly adhering to the MVC architectural pattern. On the other hand, though, it can be considered Model since all the actions performed by the control panel can be handled internally, communicating to the Controller only state transitions. So, it can be thought as a Model component to simplify and clarify the architectural software structure.

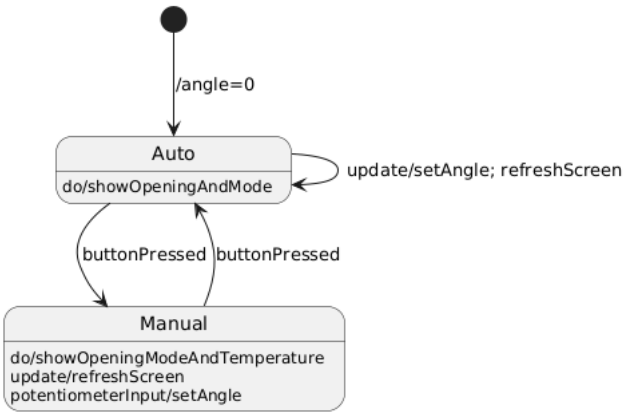

Figure 2.5: FSM modeling the Window Controller behavior

2.3 Operator Dashboard

The Operator Dashboard serves here as a remote user interface to control the whole system. It offers a graphical interface in order to present to the user relevant data with a certain clarity and processing, in order to simplify their operations. It's based on a web interface, and shows various components, which can represent plain output or be associated to input commands. In particular, it offers the equivalent of the Window Controller in-place panel, with a button to switch between manual and automatic mode, and it adds to that a button to register that an alarm situation has been restored. It communicates via HTTP to the Control Unit, which is then responsible of reflecting user actions on the Model.

2.4 Control Unit

This subsystem serves as the application Controller. It exchanges data with all subsystems, storing internally what's relevant for the user in order to represent it properly, and ensures the user input is effective on the real-world domain. It's the core of the application, and therefore tracks the state of the other subsystems, and bridges among the various communication protocols (in this case Serial, HTTP and MQTT) to allow effective data exchange between subsystems. It will be made of various components itself, and its main goal is to decouple the other subsystems, trying to minimize inner coupling between its own components.

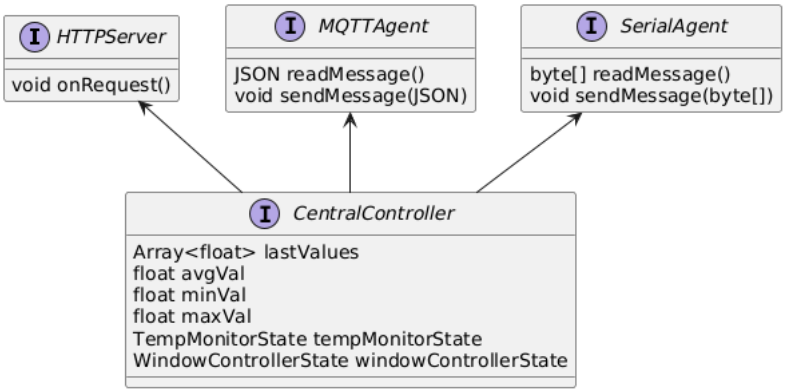

Figure 2.6: UML lass diagram showing the architecture scheme for the Control Unit sub-components

Chapter 3

Implementing Choices

3.1 Temperature Monitor

3.2 Window Controller

3.3 Operator Dashboard

3.4 Control Unit