Coffee Maker Machine

Improvements:

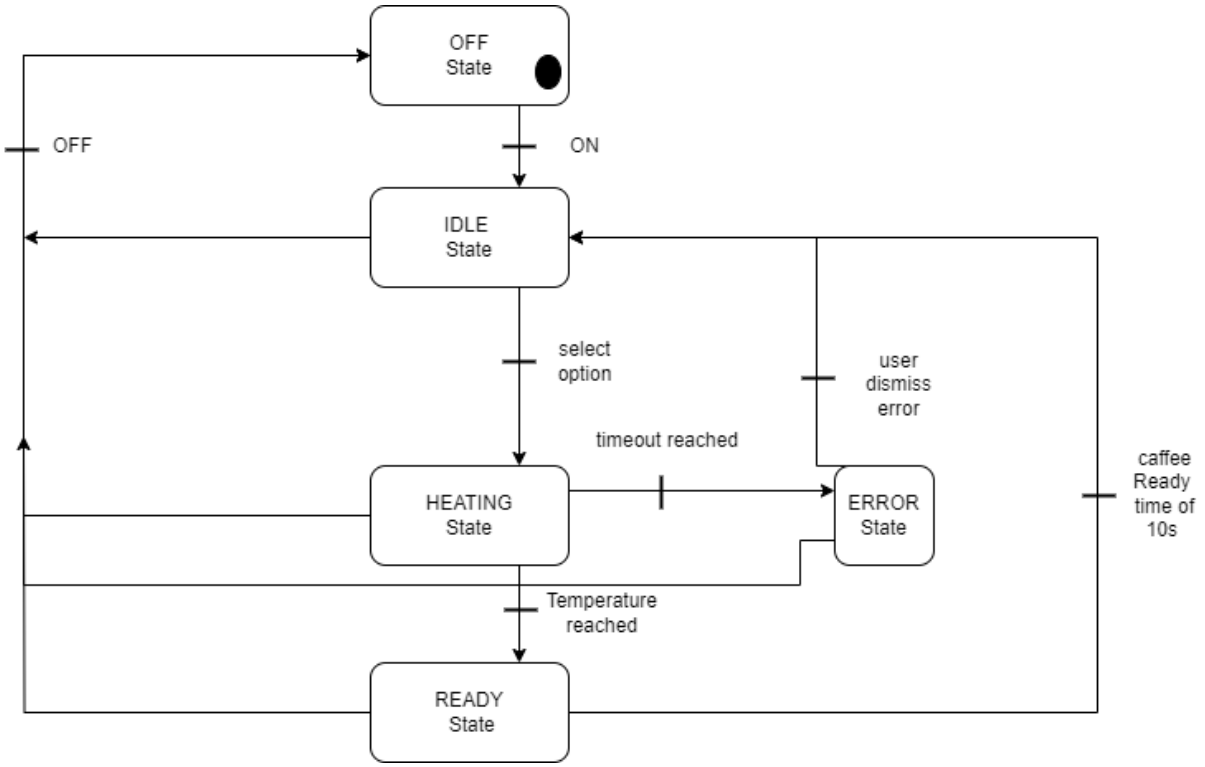
- **IDLE state:** Upon turning on the machine, it will enter an idle state, where it awaits user interaction. (Implemented and tested)
- **Error state:** An error state has been incorporated to manage various issues that may arise with the machine. (Implemented only in heating state and tested)
- timeout mechanism: To ensure efficient performance, a timeout feature has been implemented. This feature monitors the hardware components to prevent prolonged heating processes. Should a hardware malfunction occur, the system will gracefully transition into the error state. (Implemented and tested)

- ingredient verification: Prior to initiating any selected option, the machine will confirm the presence of essential ingredients—such as coffee, water, and milk. Should any of these components be absent, the machine will transition into an Error state, prompting user intervention. However, this step has not yet been implemented and requires further simulated hardware reactions and additional time and effort for testing.

New Requirement:

- The machine initially starts off in an OFF state.
- Once the user turns it on, the machine will transition to the Idle state. The user should choose an option.
- Once the option is selected, the machine will enter the Heating state and begin heating the water.
- Once the temperature is reached, the machine will transition to the Ready state.
- After 10 seconds in the Ready state, the machine will return to the Idle state.
- If a 1-minute timeout is reached and the temperature has not been reached, the machine will transition from the Heating state to the Error state.
- Once the error has been handled and the user dismisses the error message, the machine will return from the Error state to the Idle state.
- Once the user turns OFF The machine in any State the machine should be turned off

Diagram state:



Test strategy:

To thoroughly test the new requirements for the Coffee Maker Machine, several testing approaches can be applied, including state transitions coverage, boundary testing, and path testing.

1. State transitions Coverage:

Ensure that all possible state transitions are tested based on different events and conditions

- OFF → Idle (when turned on)
- Idle → Heating (when option selected)
- Heating → Ready (temperature reached)
- Heating → Error (timeout and temperature not reached)
- Ready → Idle (after 10s)
- Error → Idle (after user dismiss error)

2. Boundary Testing:

Test scenarios at the boundaries of valid and invalid inputs or conditions:

- Timeout boundary (1 min)
- Temperature boundary (temperature reached and not reached)

3. Path Testing:

Path testing aims to test different paths or sequences of events to ensure comprehensive coverage.

We use decision table outlining the test cases based on the provided requirements:

Current State	Event	Conditions	Next State
OFF	Turn On		Idle
OFF	Turn Off		OFF
Idle	Choose Option		Heating
Idle	Turn Off		OFF
Heating	Temperature Reached	Temperature reached	Ready
Heating	Timeout	Temperature not reached after 1 minute timeout	Error
Heating	Turn Off		OFF
Ready	10 Seconds Elapsed		Idle
Ready	Turn Off		OFF
Error	Dismiss Error		Idle
Error	Turn Off		OFF

I have just tried to cover four possible scenarios, which are:

Test Case 1: Normal Operation - Successful Heating

- Start in OFF state.
- Turn the machine ON.
- Select an option.
- temperature reaching.
- Operation finished
- back to Idle state.

Test Case 2: Timeout During Heating

- Start in OFF state.
- Turn the machine ON.
- Select an option.
- timeout reached
- user dismiss the error.
- Machine transitions back to Idle state.

Test Case 3: Immediate Turn Off

- Start in OFF state.
- Turn the machine ON.

- User select option
- Turn the machine OFF.
- Machine transitions back to OFF state.

Test Case4: Transition from Error After Dismissing

- Start in OFF state.
- Turn the machine ON.
- Select an option.
- temperature reaching.
- Operation finished
- Select another option.
- Temperature not reached
- timeout reached
- user Handle the error.
- Machine transitions back to Idle state.

Temperature Control Unit:

The Temperature Control Unit was developed to facilitate communication with a simulated hardware unit through the MQTT protocol. This communication process encompasses both subscribing to and publishing the target and current temperature values, thereby enabling seamless interaction between the control unit and the hardware unit.

In order to ensure the proper functionality of the developed control unit script, I created a hardware unit script. Additionally, I utilized a Docker image for the MQTT broker. This setup guarantees that the control unit script performs as anticipated, effectively managing temperature control and coordination with the hardware unit.

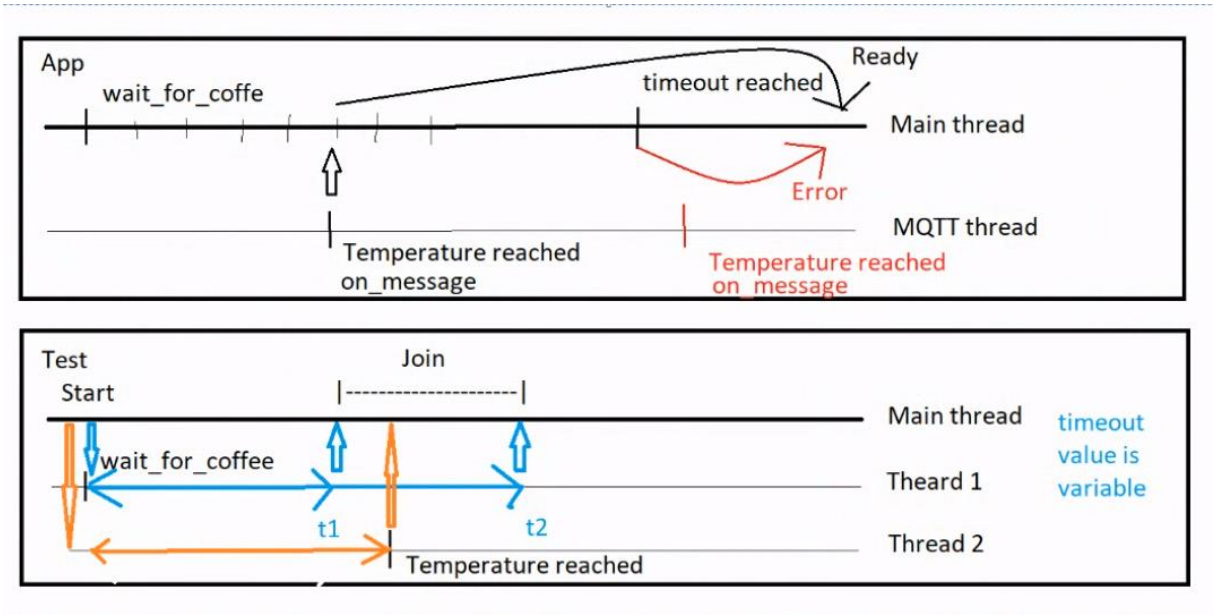
Functionality:

- The control unit initiates the process by connecting to the MQTT broker and subscribing to the 'current temp' topic.
- The hardware unit, on the other hand, also connects to the broker and subscribes to relevant topics.
- Once the heating state is started, the temperature control unit publishes a target temperature and sends a command to the hardware unit to activate the heater. A timer is then initiated.
- The hardware unit responds by heating up and continuously sending its current temperature to the control unit.
- The control unit continuously compares the current temperature with the target temperature. If the target temperature is reached before the timeout, the control unit instructs the hardware unit to stop the heater, transitioning the system to a 'ready' state.
- If the timeout is reached and the target temperature has not been reached, an error condition is detected and handled appropriately.

By following this sequence of actions, the Temperature Control Unit effectively controls the heating process and ensures that the desired temperature is attained within the specified timeframe.

Testing:

To test the temperature control within a specified timeout, I implemented threading logic. This approach was necessary because we have two functions that need to work simultaneously. Threading allows these functions to run concurrently and ensures proper coordination for effective testing of the temperature control system.



This logic was implemented in test_temperature_control.py. To avoid waiting for 60 seconds for each test, I have set the timeout to 10 seconds and applied boundary testing for the timeout duration.