SYSC3010 Computer Systems Development Project
FANS Detailed Design Report

Group L1-G8

[1]

Grant Achuzia, 101222695
Javeria Sohail, 101197163
Matteo Golin, 101220709
Saja Fawagreh, 101217326
TA: Sean Kirkby

Created: March 11th, 2024
Modified: March 11, 2024

Contents

1 Problem Statement

1.1 Functional Requirements

1. TODO

2 Design Overview

2.1 System Overview Design

2.2 Communication Protocols

2.2.1 I2C Communication

Sender	Receiver	Message	Data Format	Protocol

Table 1: Messages for I2C communication in FANS.

2.2.2 Local Area Network Communication

Sender	Receiver	Message	Data Format	Protocol

Table 2: Messages for local area network communication in FANS.

2.2.3 Cloud Database Communication

Sender	Receiver	Message	Data Format	Protocol

Table 3: Messages for cloud database communication in FANS.

2.2.4 User Notification Communication

Sender	Receiver	Message	Data Format	Protocol

Table 4: Messages for user notification communication in FANS.

2.3 Message Sequence Diagrams

2.3.1 Trigger Emergency Use Case

2.3.2 Add New Contact Information Use Case

2.3.3 Change Emergency Threshold Use Case

2.3.4 Emergency Response Use Case

2.4 Database Table Schema

3 Software Design

Each node in the FANS system has its software design outlined below. Simple system nodes with an algorithmic design have their control flow described using state machine diagrams as an aid. Class diagrams are included for nodes that use an object-oriented approach for their design.

3.1 Sensor Data Collection System

The sensor collection system will be programmed using the Python programming language for simplicity and its large collection of libraries.

The structure of the program will be that of two continuous polling loops, running as separate processes to utilize the CPU fully and get around the Global Interpreter Lock (GIL) of Python that prevents maximizing traditional thread performance.

The first polling loop will be responsible for collecting sensor data continuously and placing it on a queue for the other loop. It will collect data from all sensors, and that is its sole responsibility.

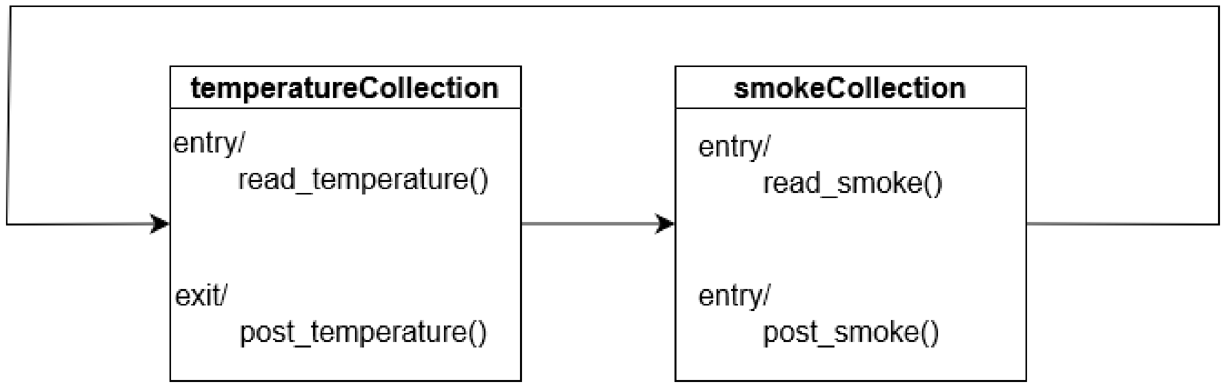


Figure 1: The polling loop for sensor data collection.

The second polling loop will be responsible for reading the sensor data from the shared queue and writing it to the Firebase database. It will also be responsible for performing logical checks on this data to determine whether or not there is an active fire emergency. It will update the emergency flag in the database whenever a sensor data measurement is above the specified threshold, and also communicate the emergency over UDP to the notification system and alarm system. Once all sensor data has been posted to the database, this loop will check for configuration updates in the sensor data thresholds from the Firebase database and apply them to the next iteration.

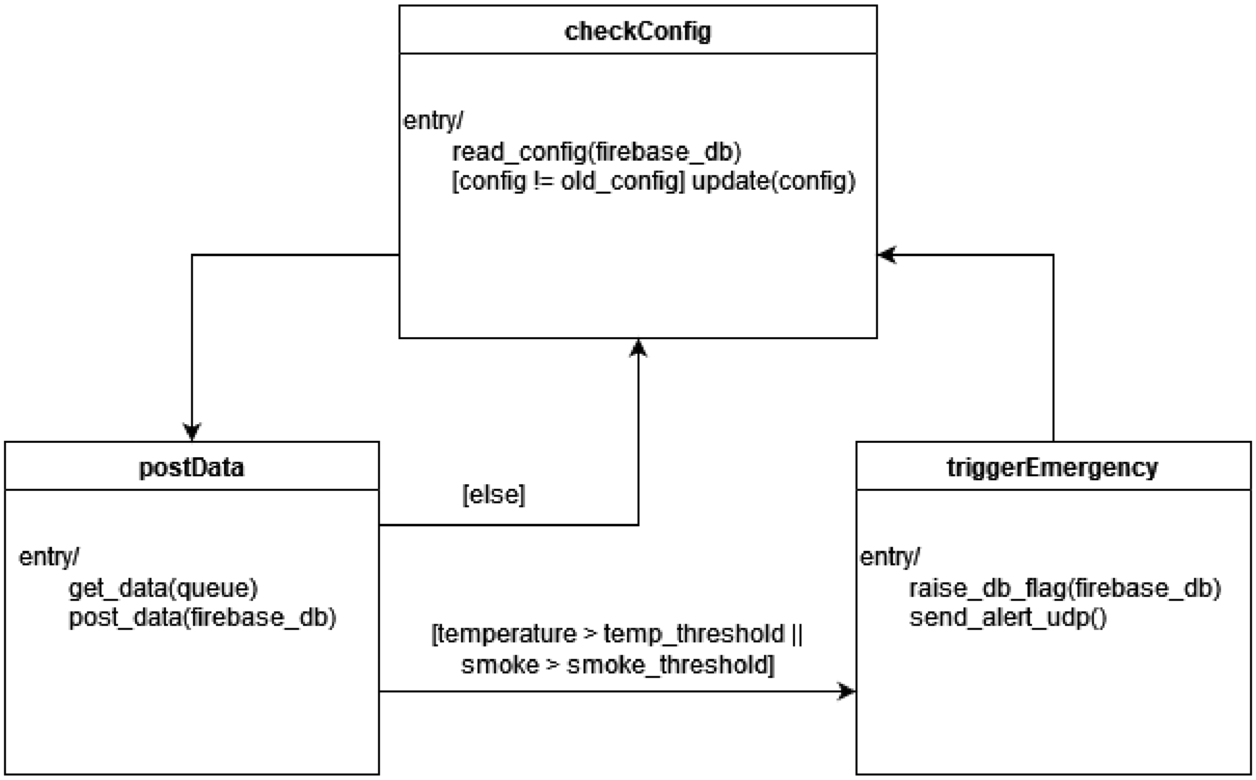


Figure 2: State machine diagram for the second process of the sensor data collection system.

3.2 Alarm System

The alarm system will be programmed using the Python programming language, again for its simplicity. The alarm system will be composed of one continuous loop, which waits to receive incoming UDP messages. When a UDP message signifying an emergency is received, the alarm system will trigger both an alarm buzzer and flashing LED lights for an infinite duration. The alarm response will be interrupted only when the system receives another UDP message signifying that the emergency has ended. This behaviour is similar to a state machine, so the software will be created using the state design pattern.

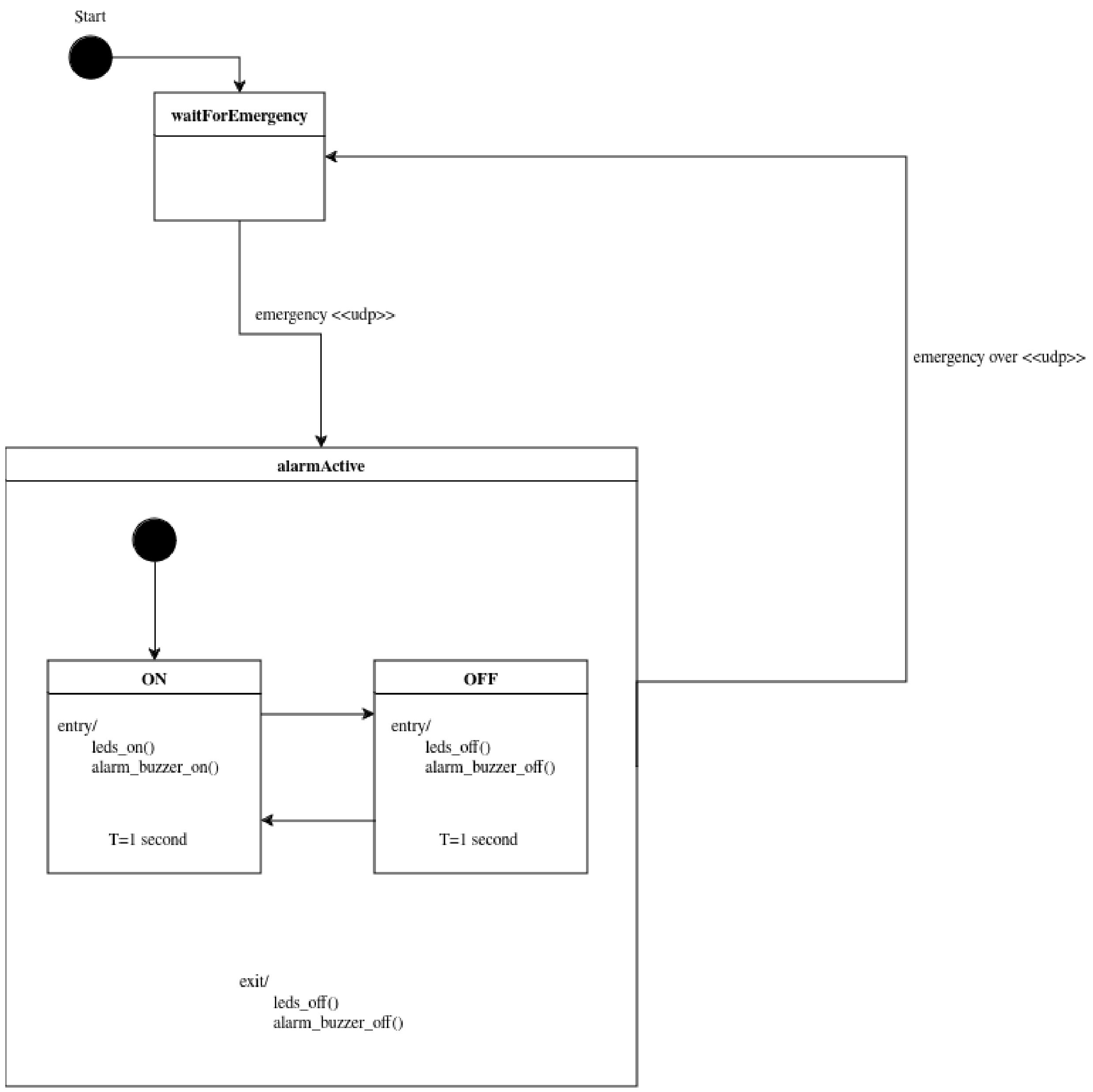


Figure 3: State machine diagram for the alarm system node.

3.3 Notification System

The notification system will be written in the Python programming language, also due to its simplicity. The notification system will listen for a UDP message signifying an emergency, and then send out email and SMS notifications to all users in the Firebase database. Once a UDP message signifying the end of the emergency is received, the system will send a followup email to all users that the emergency has been resolved.

The notification system will keep a local cache of user contact information as a backup for failing internet connectivity. It will periodically update its local cache with any changes in the upstream

Firebase database.

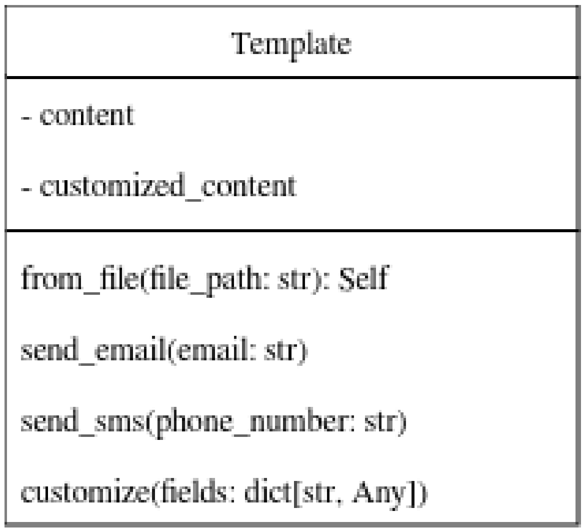
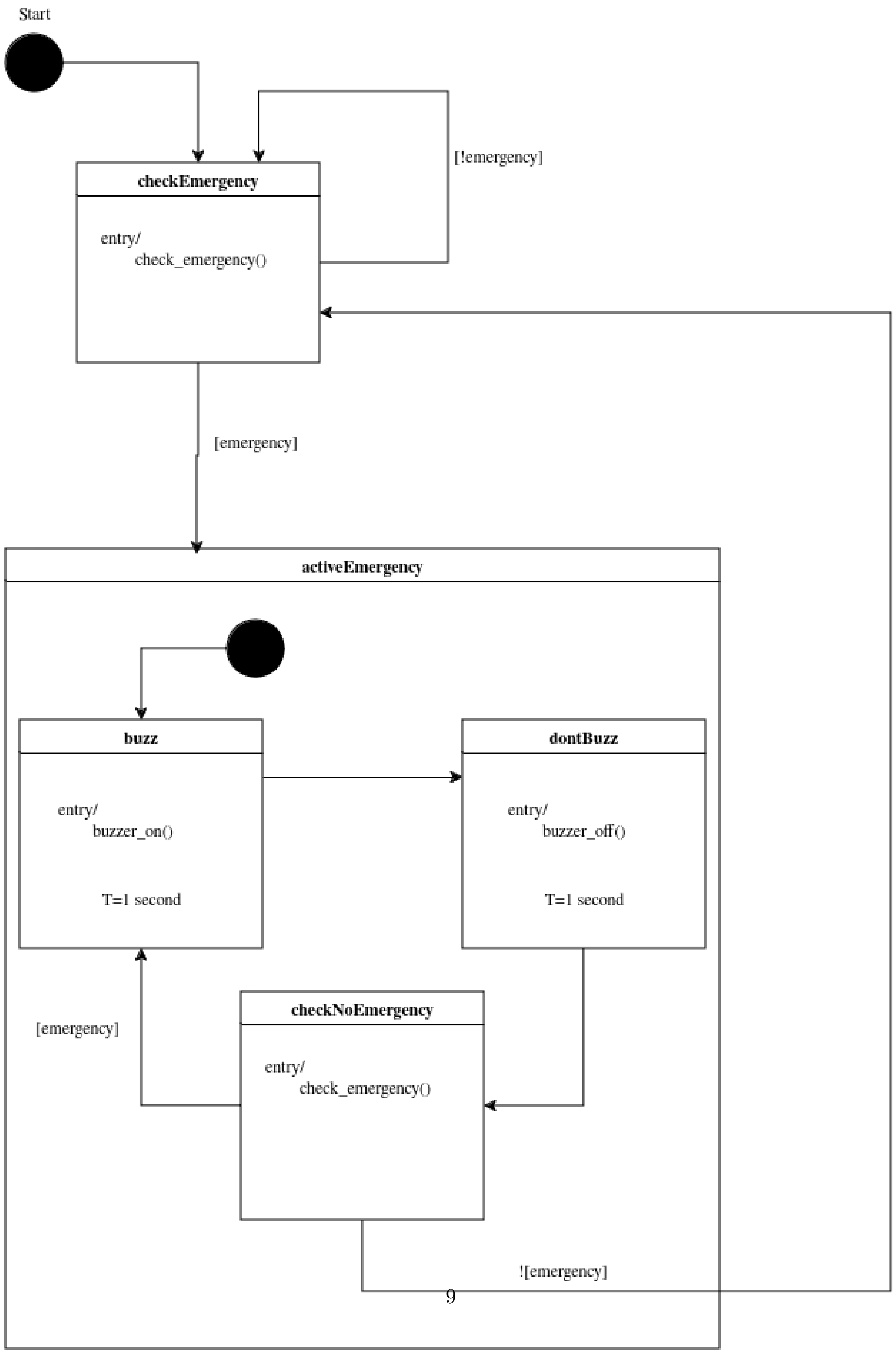

Template
- content
- customized_content
from_file(file_path: str): Self
send_email(email: str)
send_sms(phone_number: str)
customize(fields: dict[str, Any])

Figure 4: Class diagram for the notification system node.

The notification system will also have locally stored email and SMS templates for notifications. These will be loaded as “Templates”, which provide an interface for easy customization and sending of notifications.

3.4 Haptic Alarm System

The haptic alarm wearable also has a very simple software design, and will be written in the Python programming language. The program will continuously poll the Firebase database for changes in the emergency flag. Once the emergency flag has been raised, the haptic alarm will begin to buzz on and off. During this time, it will continue checking for changes in the emergency flag on the Firebase database. Once the emergency flag is lowered, the haptic alarm will stop buzzing, signifying the end of the emergency.


Start
checkEmergency
entry/ check_emergency()
[!emergency]
[emergency]
activeEmergency
buzz
entry/ buzzer_on()
T=1 second
dontBuzz
entry/ buzzer_off()
T=1 second
checkNoEmergency
entry/ check_emergency()
[emergency]
![emergency]

4 Hardware Design

4.1 MQ2 Smoke Sensor

4.2 LCD Display

4.3 AudioHat Module

4.4 Piezoelectric Buzzer

5 GUI Design

5.1 Table of Users/Roles

6 Test Plans

6.1 End-to-end Communication Demo

6.2 Unit Test Demo

6.3 Final Demo

7 Project Update

7.1 Project Milestones

7.2 Schedule of Activities

References

[1] P. Matoušek, *Thick smoke on black background*. [Online]. Available: `https://www.freeimages.com/photo/thick-smoke-on-black-background-1633270` (visited on 02/11/2024).