TTK4145 Prelimitary design description - Group 13, Monday 8.15-12.00, Desk 14

Léo Bernard (ljbernar@ntnu.no), Mona V. Vahedi (monavv@ntnu.no),
Sara Østnor (saraao@ntnu.no)

1. Operation and Fault Tolerance Strategy

The controller distributes hall orders to the elevators, which use a cost function to determine the order of hall and cab requests. Each elevator module manages its own cab button lights, while the controller manages all hall lights, receiving acknowledgments of completed requests from the elevators. Elevators maintain local queues for cab requests to ensure resilience. The elevators periodically send their current queue and position to the controller, ensuring that if a node with active hall requests disconnects, the controller reassigns the request to another elevator. Upon recovery, the elevator resumes cab requests based on the last known state and receives an updated queue from the controller without reassigned hall requests. To handle spontaneous crashes, elevator queues are preserved until the target floor is reached. This ensures that if a cab node crashes with active orders, it will proceed with those orders upon recovery.

TCP is chosen as the primary communication protocol for its reliability features, including acknowledgment (ACK) and ordered delivery, minimizing issues with packet loss. UDP packet loss during initialization can be tolerated due to periodic retransmission.

2. Module Design

The elevator project consists of the following modules:

- **Controller**: Handles the program logic and coordinates information flow between elevators and hall buttons. Equipped with a queue of incoming hall requests, that are distributed between the elevators.
- **Backup Controller**: Maintains a copy of the queue and takes over if the primary controller fails. In such case, the backup controller will become the new master, making the previous master the new backup when it is reconnected.
- **Queue**: Manages requests with queue elements containing details like button type (cab or hall) and source elevator.
- **Elevator**: Operates each elevator, handling requests, movement, and door control.
- **Door** and **Time**: *Door* manages door operations, starting a *Timer* thread to keep doors open for a fixed period when the elevator reaches a target floor, and during obstructions.
- **Button** Module that can represent different instances of buttons (hall/cab), also allowing enabling and disabling a button.

3. Network Topology and Protocols

The network follows a star topology, with the controller acting as the central hub (master). The elevators and the hall button panels communicate exclusively through the controller. This ensures a single point for fault management, such as reassigning hall requests during faults.

UDP is used for initialization and failover detection due to its low overhead, while TCP is used for the main communication to ensure reliable delivery.
Their use can be summarized as follows.

- Master broadcasts UDP packets to identify itself and initialize connections with the elevators.
- Backup controller monitors the master's UDP broadcasts. If the broadcasts stops, the backup assumes the master role and emits its own broadcasts until the master recovers.
- Elevators listen for broadcasts and setup themselves up over TCP.
- All communication after initialization occurs over TCP for reliability, using acknowledgements to do so.

4. Communication and Serialization

The elevator system project will be implemented using Rust due to its robust concurrency model and safety features. Rust's ownership model prevents data races by ensuring that mutable references have a single owner at a given time. In addition to mechanisms like `Send` and `Sync`, provide further safeguards against race conditions.

Struct serialization is used for communication between modules. Unlike JSON, struct serialization avoids unnecessary overhead and dependency on external libraries. Rust's `std::net` module is utilized to manage TCP and UDP transmission.

5. State Diagram

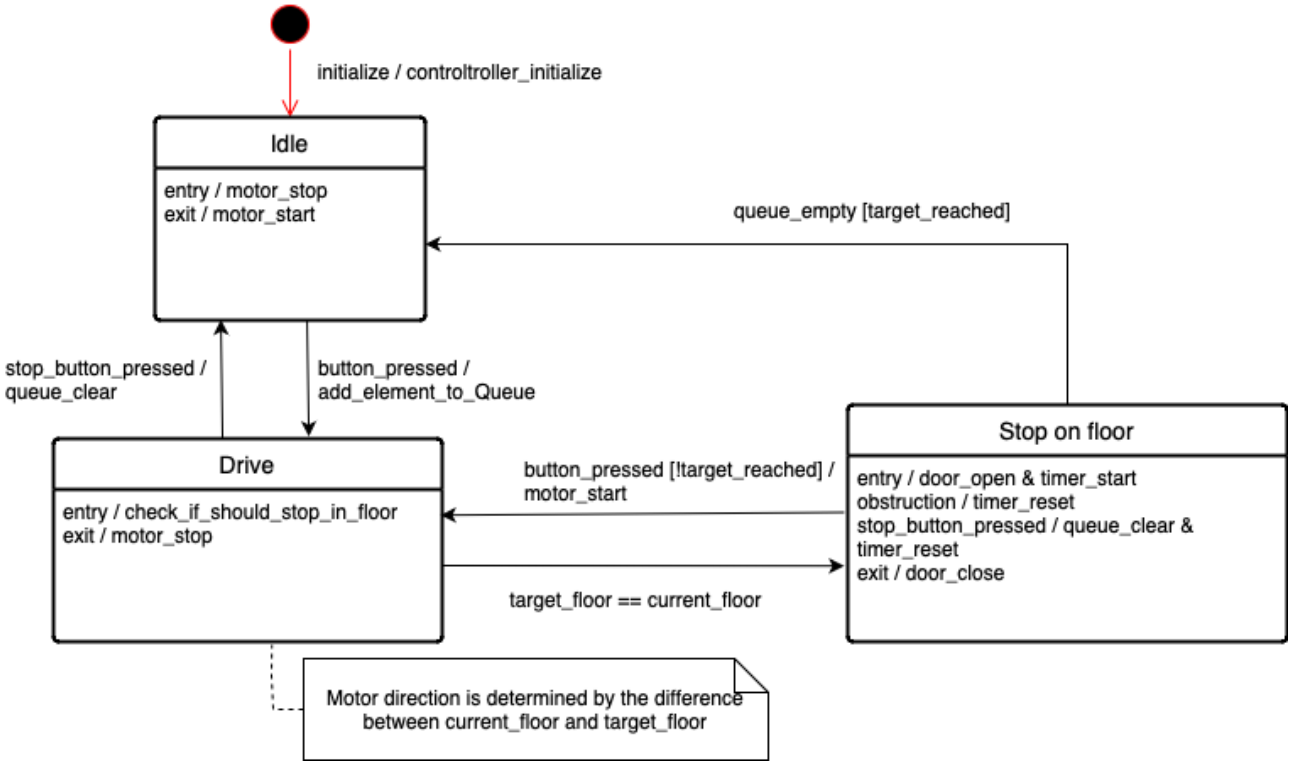

Figure 1: Single elevator state diagram.