NTNU
Kunnskap for en bedre verden

Department of Engineering Cybernetics

TTK4145 - Real-time Programming

Preliminary design description

Author:
Hermann Baarset (*hermaba@stud.ntnu.no*)
Igor Pekarskiy (*ihorp@stud.ntnu.no*)
Jonathan Kretschmer (*jonathhk@stud.ntnu.no*)
Group 35 (Wednesdays 09:00-12:00, workstation 06)

Friday 2nd February, 2024

1 Module Overview

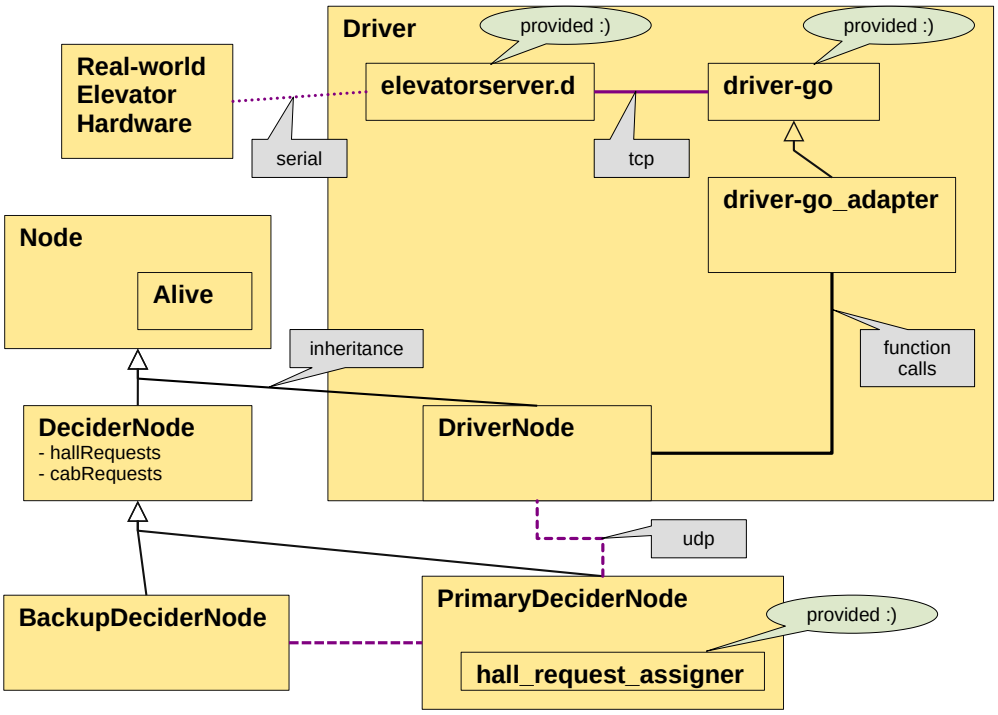

Figure 1: Module Overview

Real-world Elevator HW – elevator (or simulator) with cab, floors, buttons, light, motor, sensors and serial interface
Node – shared functionality, between all node types
Alive – continuously listening on sent "I-am-alive" messages over broadcast and maintaining an alive-list of DeciderNodes and DriverNodes; functionality is shared with all node types
Driver (*container module*) – controlling the HW and providing a network interface with reduced (network-fail-safe) functionality to the primary
driver-go_adapter – extending the provided driver with certain functionalities, i.e. *move_to_floor(int)*
DriverNode – forwarding user interaction to primary; receiving task from primary, like *go_to_floor*, and taking care, that it is executed
PrimaryDeciderNode – user requests are received, the requests are sent to the BackupDeciderNodes and it decides which elevator handles which request
BackupDeciderNode – used by the PrimaryDeciderNode as remote backup of all hall- and cab-requests

2 Topology and Protocol Choice

Programming language – *go*: take advantage of the message passing paradigm between concurrently running processes with the language inherent *channels*

UDP is the main communication protocol between remotely running processes. This allows us to send custom messages, include acknowledgement timeouts and properly react on detected timeouts.

The alive modules require broadcasting, so everybody knows who is online or unavailable. Other modules will communicate directly with each other (peer-to-peer). For the request assignment we use a star topology, where the primary acts as the common "hub" and makes the decisions. We also separate *decision making* and actual *hardware controlling* part into different independent modules. This way there could be even more backups than physical elevators running.

The design simplifies hardware issue handling by allowing a *DriverNode* to become unavailable (stop sending alive-messages) while a *BackupDeciderNode* on the same PC remains available for backups. This is good for redundancy. In this case the *PrimaryDeciderNode* doesn't need to track various error and alive modes of a hypothetical combined "DriverAndBackupDeciderNode".

3 Challanges – Fault Tolerance

The button light contract is enforced by the following procedure. When a button is pressed, the request is sent to the primary. The primary is sending the request to all available backups. Everything gets acknowledged in reversed order and the button is lit up. If any of these steps don't succeed, the button will not light up and the user has press again.

After the request has been sent to the backups, the primary makes sure the request is serviced in the most efficient manner. It will sent more elevators, if necessary, i.e. if a DriverNode crashes.

If every request has been backuped? Which problems remain to solve?

A backup disconnects/crashes/restarts/... – No big issue, as there are other backups, and the primary can deliver a fresh request-list, when it is up again.

A driver disconnects/crashes/restarts/... – No big problem, as the primary still knows its requests. It can instruct it again, when it is up again, respectively reassigning the pending requests to the still available drivers. (See Figure 2.) An elevator in active obstruction mode will set itself as unavailable. Consequently, new requests will be handled by other available nodes.

A primary disconnects/crashes/restarts/... – The challenge. We implement a fail-safe way to assign every node dynamically to one of the following modes: *SINGLE_ELEVATOR*, *BACKUP*, *PRIMARY*. See state machine for primary reconfiguration in Figure 3.

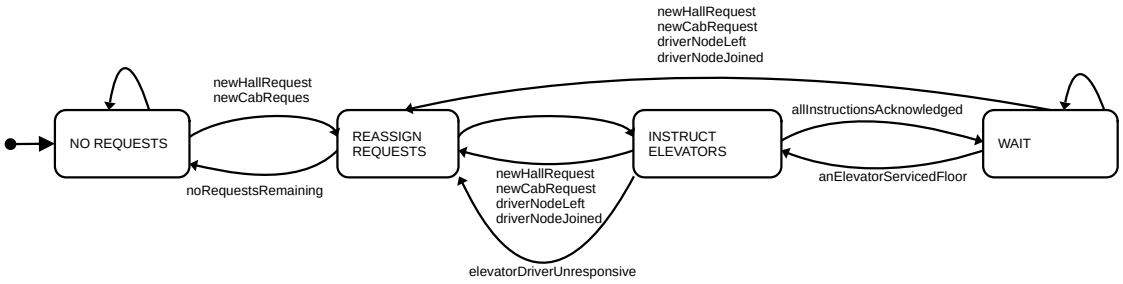

Figure 2: "State Machine" Request (Re-)Assignment

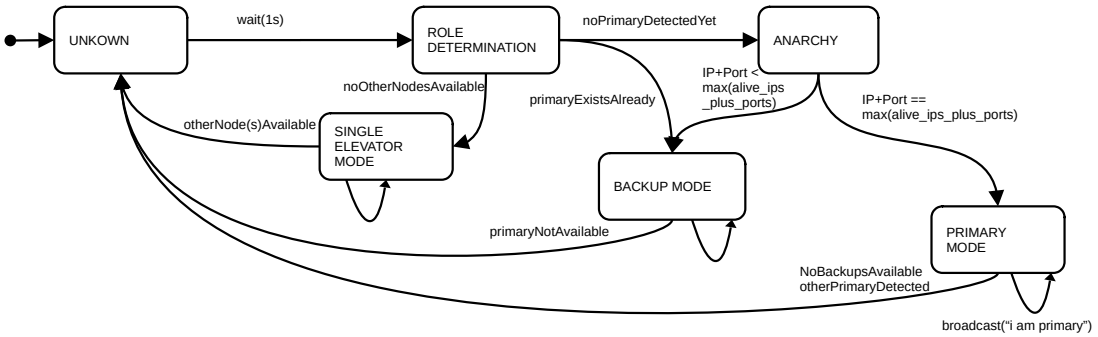

Figure 3: State Machine Primary Reconfiguration