TTK4145 - Real-Time Programming

Project design presentation

Group 72

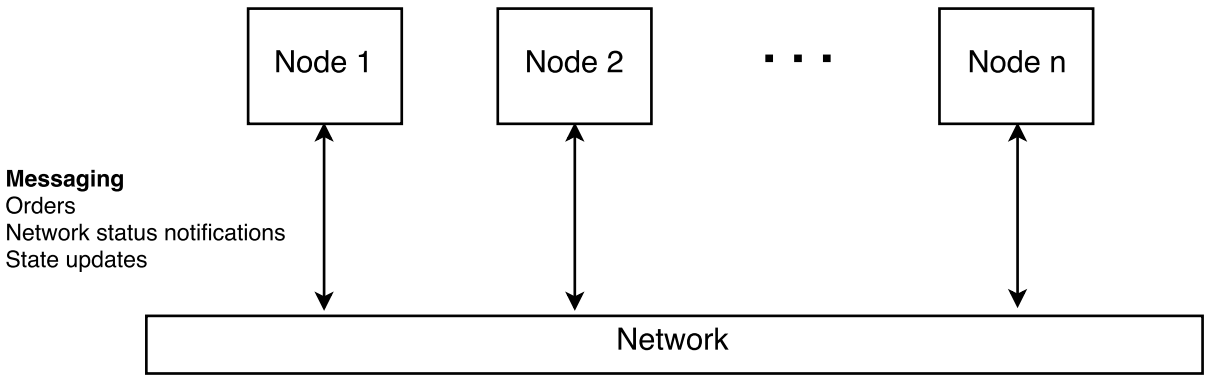
Anders A. Pedersen
Jan Tore Guggedal

Network topology

The nodes communicate on the network by built-in message passing module in Erlang. Every time a node is connected or disconnected, the others are notified.

The message passing system is also used to pass new orders and tell the other nodes that a certain order is handled. This maintains a common order queue distributed in all nodes. When one node disconnects, the others have backup and can redistribute orders. This way it is ensured that no orders are lost.

Each elevator's state is also shared with the other nodes to be used when calculating costs for any new (or recalculated old) order.



Scenarios to be handled and basis for fault tolerance

Listed below are some important scenarios the system must handle and actions we have defined to be taken to avoid losing orders. In general, if a process running on one machine experiences an error, that process (and none other), will be allowed to crash and restart to known state before possible consequences of the error spreads to other parts of the system. The system will then be in a known state. This is a central part of Erlang, and we will exploit it here.

In addition we will use a “flat structure”, as opposed to master/slave configuration, where all nodes are considered equally important and act as each other’s backups. The order queue is backed up in all nodes. Exceptions to order distribution are mentioned below, such as when a node loses network connection before order is distributed.

- **External button (up/down) is pressed**
    - Elevator driver registers signal
    - Order is distributed to all nodes
    - Order is added to all other nodes’ local queue copy
    - The external light is lit at the floor where the order was placed on all nodes
    - Order cost is calculated in each node based on complete list of other live nodes’ states.
    - Action is taken according to result of cost calculation. Only the node with the lowest cost for respective order responds to order.
    - States are updated and distributed to all nodes’ local list
    - Elevator driver receives message to execute the action (if any)

- **Internal button is pressed**
    - Internal order is added to local queue
    - A copy of the local queue is distributed to the other nodes as backup
    - Internal indicator lamp is lit at the requested floor
    - Internal orders are executed before next external order, unless the external order can be handled without changing direction or moving beyond requested floor when handling internal order

- **A node loses network connection**
    - All nodes registers that one node is no longer on the network
    - The node is removed from the live nodes list (locally stored in each node)
    - All the disconnected node’s external orders’ costs are recalculated for the updated live nodes list
    - Based on new order costs, orders are taken by the nodes
    - The node that fell out continues to serve all internal orders (buttons pressed inside the elevator)
    - Restart program when all internal orders are served and attempt reconnection to network
    - If reconnection is not possible, go to idle state
    - Retry to connect to network until successful

- **A node is reconnected to the network**

    - All nodes are notified that a new node is on the network
    - The node is added to the list of nodes and state list
    - Order costs are recalculated based on new live nodes list
    - Nodes establish who takes which orders and service continues
    - If the reconnected node has remaining queue of internal orders, these are handled

- **A node receives an order, but network goes down before distribution**
    - The node detects that the message was never received by the other nodes (ie. detecting that it is no longer on the network)
    - Add order to internal queue and handle as a single elevator
    - After that order is handled, stop handling orders and enter idle state
    - Wait for reconnection to network

- **A node loses power**
    - The same happens as when a node falls out of the network, except that internal orders are not handled (at the time) by the node losing power
    - Internal orders are backed up in the other nodes and stored until the node is back up again

- **A node comes back on the network after power outage**
    - The node signals all other nodes that it's alive again
    - The other nodes respond with a confirmation and potentially the backed up internal order queue, if that existed before the elevator shut down
    - The elevator needs to know if it came back from a network disconnection or a power-up.

- **An elevator reaches a requested floor (both internal and external orders)**
    - Order is now considered handled, and is removed from queue
    - External orders are signalled as handled to all other nodes and removed from their queues as well
    - Copies of updated local queue is distributed to the other nodes
    - The "door open" light as an indicator for open door
    - A timer is started to keep the doors open for a predefined time
    - Timer times out and open door light turns off
    - Elevator handles next order in queue or goes to idle state

Modules

… and what they should do

Main

- Controls the program flow and can interface with all other nodes

Network

- Sets up the connections between the nodes
- Uses Erlang's built-in functionality to connect nodes by using names and cookies
- Enables message passing between nodes
- Enables "notifications" when a new node is connected or disconnected

Elevator driver

- Interfaces with the C driver for the physical elevator system
- Enables control of indicator lamps and detection of button states

Order cost calculation

- Based on each elevator's state and location, the cost for each newly added order is calculated
- Each elevator determines which order to handle next
- By using the same algorithm, all will end up with the same calculations and which elevator to take which order

Order handler

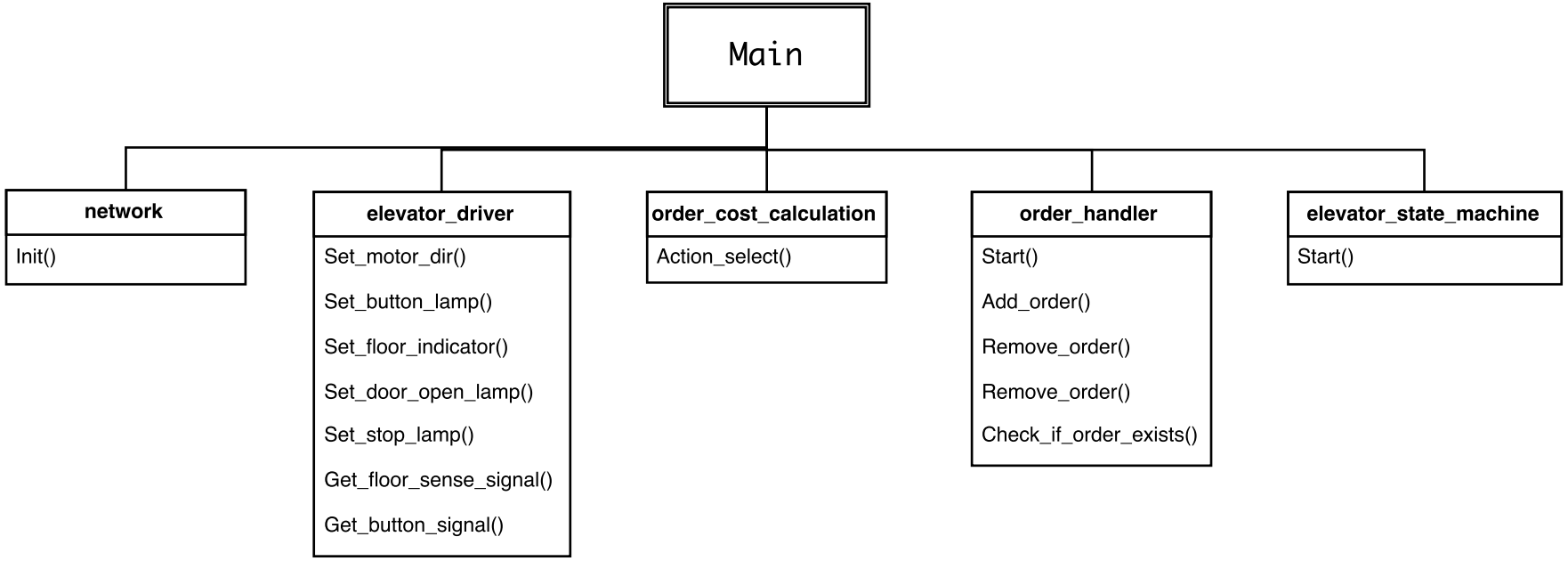
- New orders is to be distributed to all nodes
- External orders is stored in queue, which is equal in all nodes
- Internal orders stored in copied version on all nodes in case of power outage on one
    - (alternatively to be solved by storing internal orders to local file)

Elevator Finite State Machine

- Keep track of each node's state
- All elevators' states are known to each node to be used in cost calculation
- When a new event happens, such as a floor is reached, an order is handled or a new order is received, the state is changed

Modules and interfaces

This design is made to be used in code written in Erlang. Functions within a module should only be accessed from main. Communication between modules will happen by message passing controlled by main or through function calls controlled by main.



Example of interaction between modules

The arrows indicate flow, not function calls across modules. All communication goes through the Main file and its functions that hold process IDs and passes messages between processes. For data not intended to be sent over the network, data is sent as parameters to functions, and no data is shared. The function calls mentioned in the module blocks indicate which functions in the interfaces are used by the processes running in main context.

Pressing an external button