Project Description

The corporation *Wielander Inc.* is an internationally successful manufacturer of load and person elevators. The corporation has been founded in 1907 by George Wielander an engineer for mechanical winching systems. Today Wielander Inc. is a high technology corporation with 300 employees worldwide. The impact of software as future technology has been realized and is reflected in the corporation's product portfolio.

The most recent generation of elevators was introduced roughly 20 years ago. The forthcoming elevator control software should integrate cutting-edge technology, all the while guaranteeing compatibility with the existing elevators that rely on proprietary protocols. As a result, a crucial undertaking is to deliver software updates for the current elevators to align them with the new software system architecture.

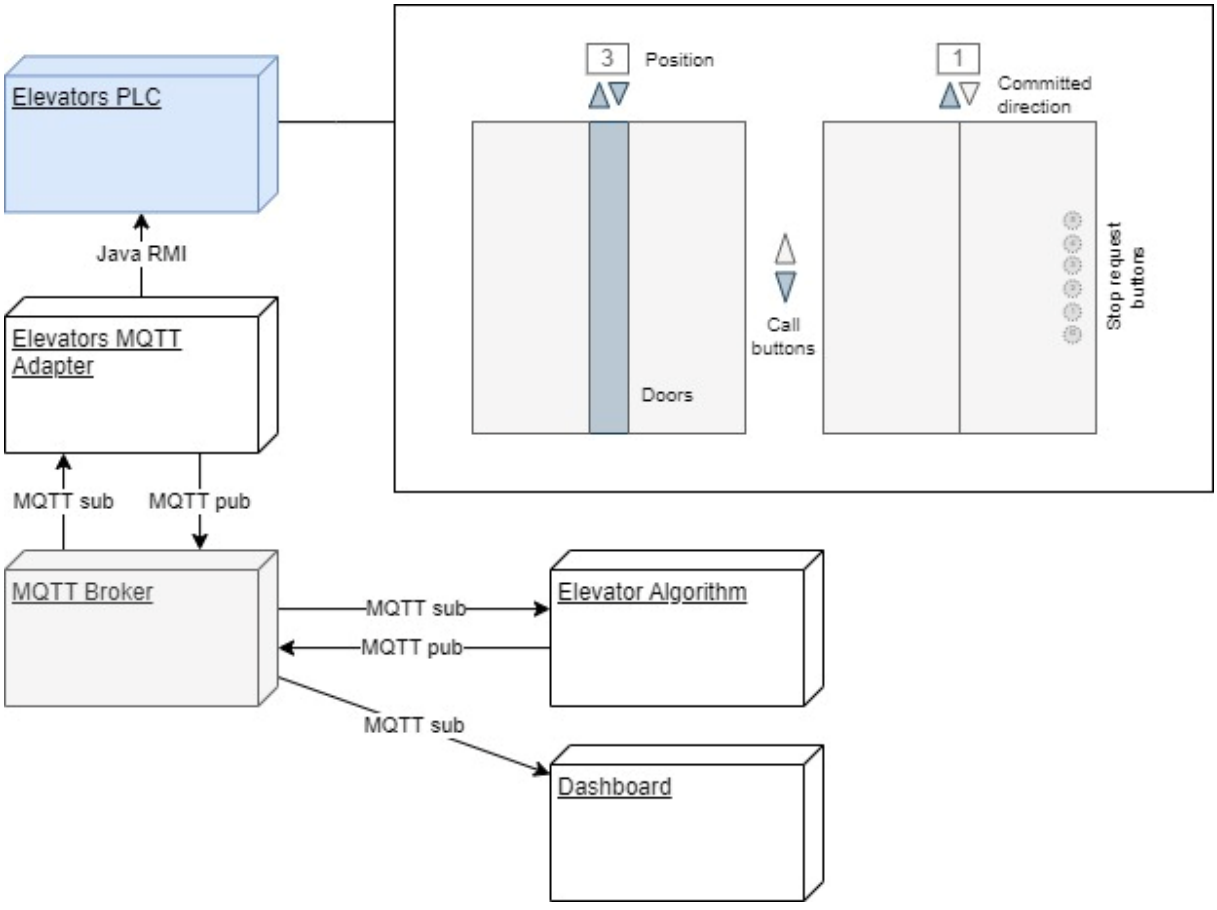

Figure 1: Targeted architecture. The Elevators MQTT adapter connects to the Elevators PLC via Java RMI and translates the API to MQTT messages. The Elevator Algorithm is an MQTT client that controls the elevators according to request buttons. The Dashboard is an MQTT client that displays important information to elevator operators and service personell.

Assignment

Your job is to implement a first version of the new architecture, which is a central part of the new software system. This version will be used to evaluate technical concepts and to define the interface to the low level control system.

Following requirements have to be demonstrated with the prototype:

- The proprietary API is fully available via MQTT. Elevator state is published as MQTT messages to carefully defined MQTT topics. This includes, but is not limited to, elevator position, direction, current payload, speed, status of the doors and the set targets of the elevator. For each floor, if there is a call or a stop request. Consider “MQTT Retained Messages” for one-time static information (e.g., number of elevators).
- The *Elevator Dashboard* presents pertinent information about the elevators in a user-friendly manner. The specifics regarding the information to be displayed are open for further discussion.
- The *Elevator Algorithm* autonomously manages the elevator's operation in response to calls from various floors and the stop requests made by passengers within the car. This implementation is founded on an algorithm that efficiently guides the elevator to different floors, considering call signals, passenger requests, and the maximum weight capacity.
- The low-level system of the elevator will be exposed as RMI interface. Details about this interface are shown in the table below. The accessible values will be updated in cycles of 0.1 seconds.

Implementation and Commissioning

Use the information provided in this document as basis for the project. Further information will be provided by the client upon request (i.e., questions will answered in the course or via the discussion forum). Conduct the project in teams. Apply the techniques and tools introduced in the course in order to develop a high-quality software system. The commissioning will take place in a final presentation and discussion at the end of the course.

Interface Definition (Draft Version)

int	**getCommittedDirection**(int elevatorNumber) Retrieves the committed direction of the specified elevator (up / down / uncommitted)
int	**getElevatorAccel**(int elevatorNumber) Provides the current acceleration of the specified elevator in feet per sec^2
boolean	**getElevatorButton**(int elevatorNumber, int floor) Provides the status of a floor request button on a specified elevator (on/off)
int	**getElevatorDoorStatus**(int elevatorNumber) Provides the current status of the doors of the specified elevator (open/closed)
int	**getElevatorFloor**(int elevatorNumber) Provides the current location of the specified elevator to the nearest floor
int	**getElevatorNum**() Retrieves the number of elevators in the building
int	**getElevatorPosition**(int elevatorNumber) Provides the current location of the specified elevator in feet from the bottom of the building
int	**getElevatorSpeed**(int elevatorNumber) Provides the current speed of the specified elevator in feet per sec
int	**getElevatorWeight**(int elevatorNumber) Retrieves the weight of passengers on the specified elevator

boolean	**getFloorButtonDown**(int floor) Provides the status of the Down button on specified floor (on/off)
boolean	**getFloorButtonUp**(int floor) Provides the status of the Up button on specified floor (on/off)
int	**getFloorHeight**() Retrieves the height of the floors in the building
int	**getFloorNum**() Retrieves the number of floors in the building
int	**getLogging**() Retrieves the current logging level
boolean	**getServicesFloors**(int elevatorNumber, int floor) Retrieves whether or not the specified elevator will service the specified floor (yes/no)
int	**getTarget**(int elevatorNumber) Retrieves the floor target of the specified elevator
void	**setCommittedDirection**(int elevatorNumber, int direction) Sets the committed direction of the specified elevator (up / down / uncommitted)
void	**setServicesFloors**(int elevatorNumber, int floor, boolean service) Sets whether or not the specified elevator will service the specified floor (yes/no)
void	**setTarget**(int elevatorNumber, int target) Sets the floor target of the specified elevator