Centrally Controlled Multi-Elevator System: UML Analysis

Nafis Ahmed
Department of Embedded Systems Engineering
Fachhochschule Dortmund
Dortmund, Germany
nafis.ahmed002@stud.fh-dortmund.de

***Abstract*—Background: This paper will analytically derive a system for dual elevator system on a three storied building where a central controller is responsible for distributing jobs to the locally deployed controllers for each elevator.**

Index Terms: UML, Multi Elevator

I. Introduction

The elevator system is made up of two elevators that can manage three different levels. Each floor has a button that may be used to summon an elevator. The input from the button is received by a central controller, which examines whether elevators are free for moving to the floor and then activates the relevant one. The elevators can be identified as busy or waiting for a new job by the central controller. This is accomplished by receiving proper elevator events from respective local elevator controllers. Each elevator has its own local elevator controller, which receives and processes events from the central controller. When an elevator is moving or waiting, it sends an event to the central controller. The elevator controller is in a holding pattern until the central controller sends it a fresh destination event. Moving from one floor to the next is done in steps (up or down). When the user arrives at their location, the door opens, allowing them to enter the elevator and select the desired floor. The event for closing the door and starting to move to the destination is pressing the selected floor. The elevator's central controller will be notified that the user's input has been received. When the user arrives at their destination, the door is opened, and the elevator is ready for the next duty. Extra access to the system is granted to maintenance personnel. This enables the position of the elevators (current floor) to be read out and the elevators to be controlled to a specified floor to be controlled. There is also an emergency signal button on the elevators. This is used to transmit a signal to the fire department immediately.

II. Actor Analysis

A. Passenger

A passenger has the ability to control elevators either from the inside or outside. When outside, the passenger can press a button to call the elevator or when inside the passenger has the options to choose where should the elevator go.

B. Central Controller

The central controller is responsible for processing button calls from the outside and check which elevator is free to serve the passenger on the floor the call was requested from. In some cases, both elevators can be free and therefore the central controller will collect information from the local controllers about their individual locations, and then dispatch the job to the closest elevator.

C. Local Controller

This actor is responsible for all the mechanical activities like closing or opening doors and responding to calls from the inside buttons of the elevator by a passenger. The local controller has a port available to be accessed by the central controller to check its current status whether it is busy or free for a job. It also has another port which can be accessed by the central controller to get the elevator's current location.

Use case diagram can be seen in figure 1 of Appendix

III. Class Structure

To derive any class diagram for a system, six design steps can be followed:

- Determine the problem domain's relevant classes: All substantives are potential class candidates.
- Eliminate the irrelevant classes
- Determine operations and attributes
- Find out how objects are related (Association, Aggregation)
- Extract the common properties of multiple classes and define them in a common super class
- Repeat second to second-last until the class structure is stable.

Upon studying the problem domain and use case diagram, three classes seem to be enough for defining the Java application to be developed

A. *Main App Class*

This class is the primary application class which defines event listeners for all the buttons in the system. Based on the events specific activities will be triggered for either the central controller or the local controllers. The main app class also defines the UI and connects the UI components to the event listeners.

B. *Central Controller Class*

This class provides a method to capture the respective button events for floor calls from outside. The event passes the appropriate button number (the floor number of the button) to the method which then executes the routines of Central Controller. Only one object of the central controller is created by the main app class. It is a subclass of the main app class.

C. *Local Controller Class*

Since the system consists of dual elevators, the main app class creates two objects of the local controller. It also provides methods for returning information about its status and current location. This class has attributes to identify the elevator by its ID. It is a subclass of the main app class.

Class diagram is represented in figure 2 of Appendix.

IV. Behaviour of the System

The system is designed with concurrency in mind; hence it implements the Java AWT event handling library. The event dispatching thread (EDT) is a Java background thread that processes events from the graphical user interface event queue of the Abstract Window Toolkit (AWT). It's an example of the event-driven programming approach in general. Therefore, all the buttons in the system are a member of the main app class and event listeners are attached to these buttons. The events utilize methods of the objects created from both the central controller and local controller. The main app class also has members which represent the indicator of elevator location which are modified by the local controller objects to show their current location.

Any function of the objects of local controller that is being fired by the event listeners will cause the main thread to be blocked until finished. Which means separate threads need to be started inside these functions to carry out the tasks so that the main class can update the position indicators being changed by those local controllers. Each elevator has an emergency switch which immediately sends a notification to the fire department. This makes the entire system non-blocking and fully concurrent as it should be. A derivation of the State Machine in figure 4 of Appendix shows this concurrent behavior of the entire system.

References

[1] Omg.org [Online] Available: https://www.omg.org/spec/UML/2.5/PDF/.

[2] Unified Modeling Language User Guide, Grady Booch, James Rumbaugh, Ivar Jacobson, 1998/2005

[3] Software Engineering, Ian Sommerville, Addision-Wesley, 9th Edition, 2011

[4] "Ideal modeling & diagramming tool for Agile team collaboration," Visual-paradigm.com. [Online]. Available: https://www.visual-paradigm.com/

Appendix

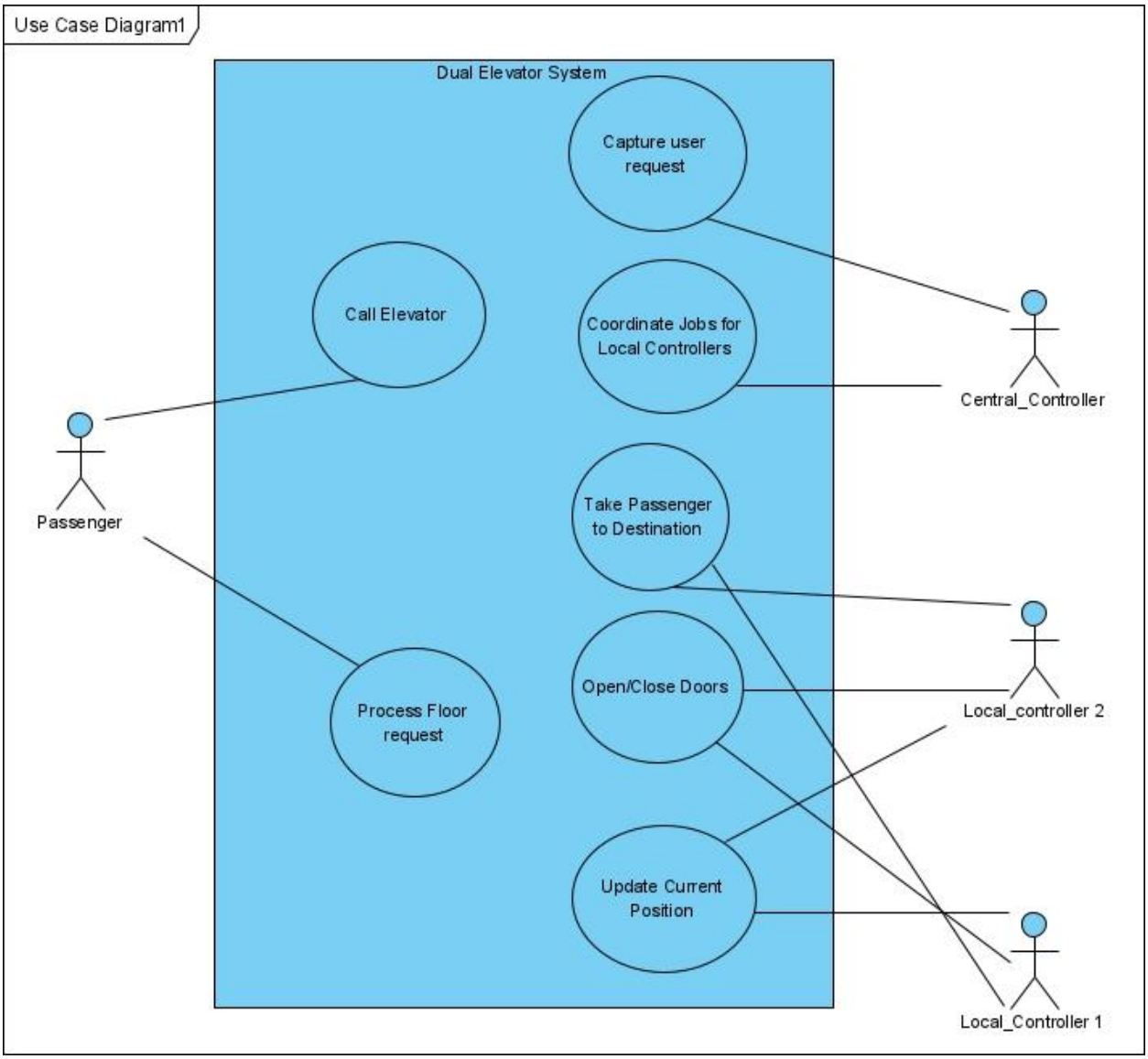

Figure 1 Use Case Diagram

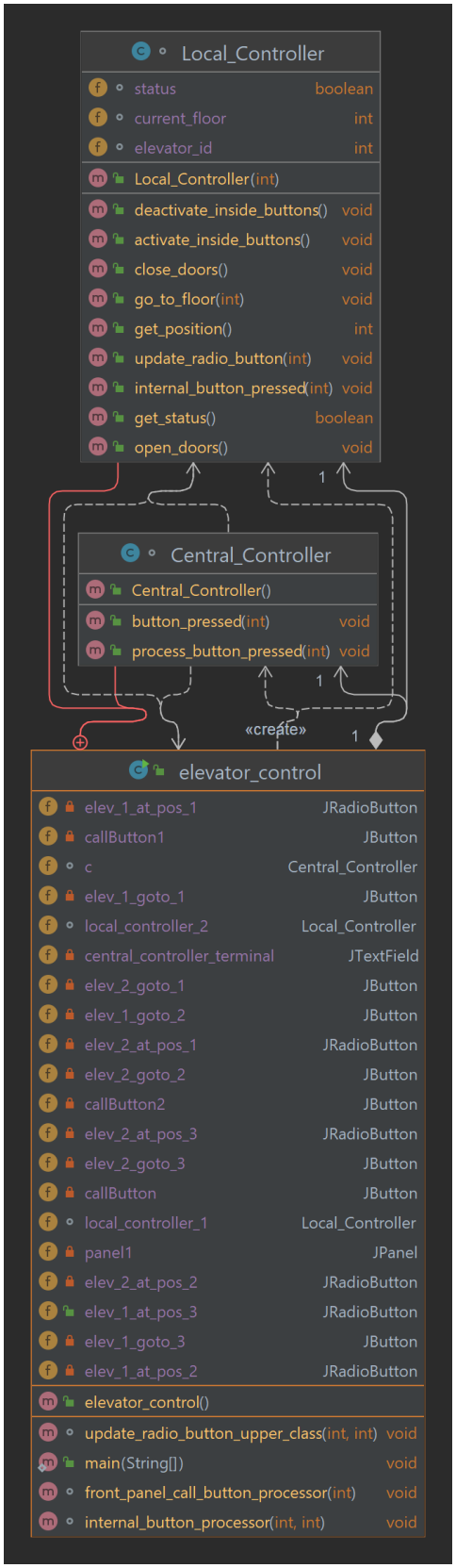

Figure 2 Class Diagram

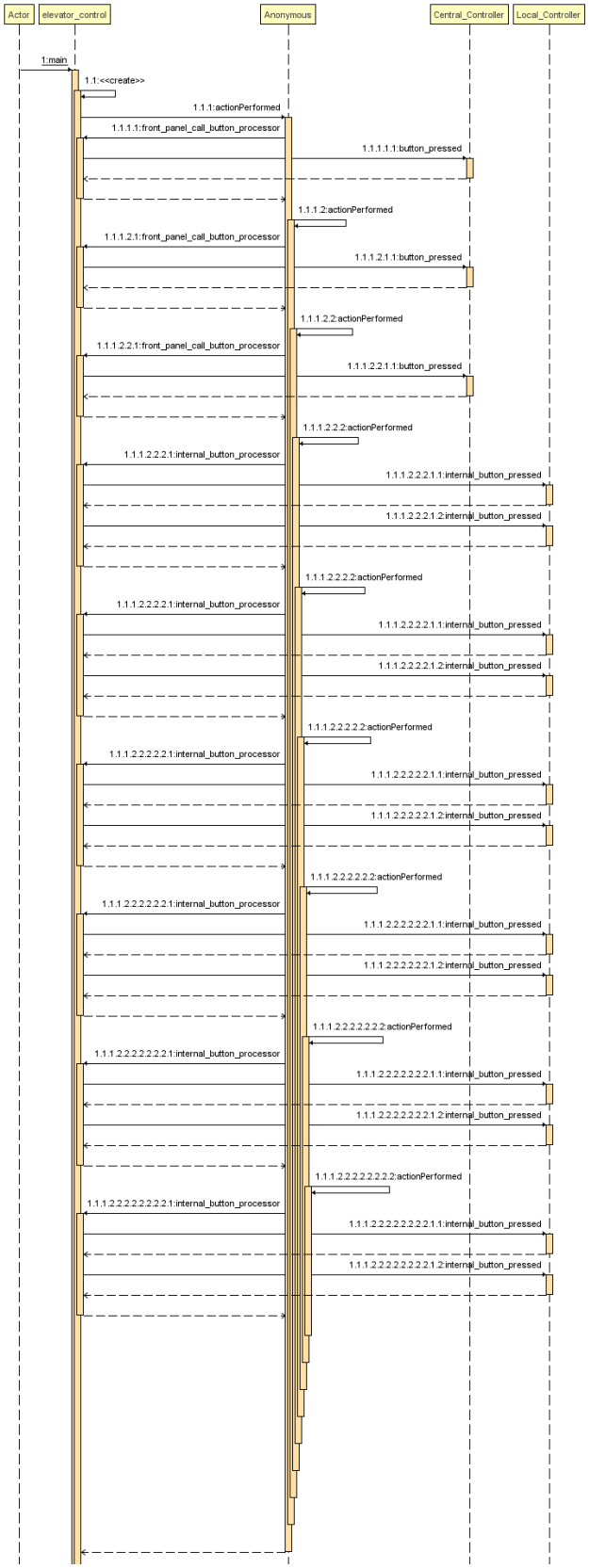

Figure 3 Sequence Diagram

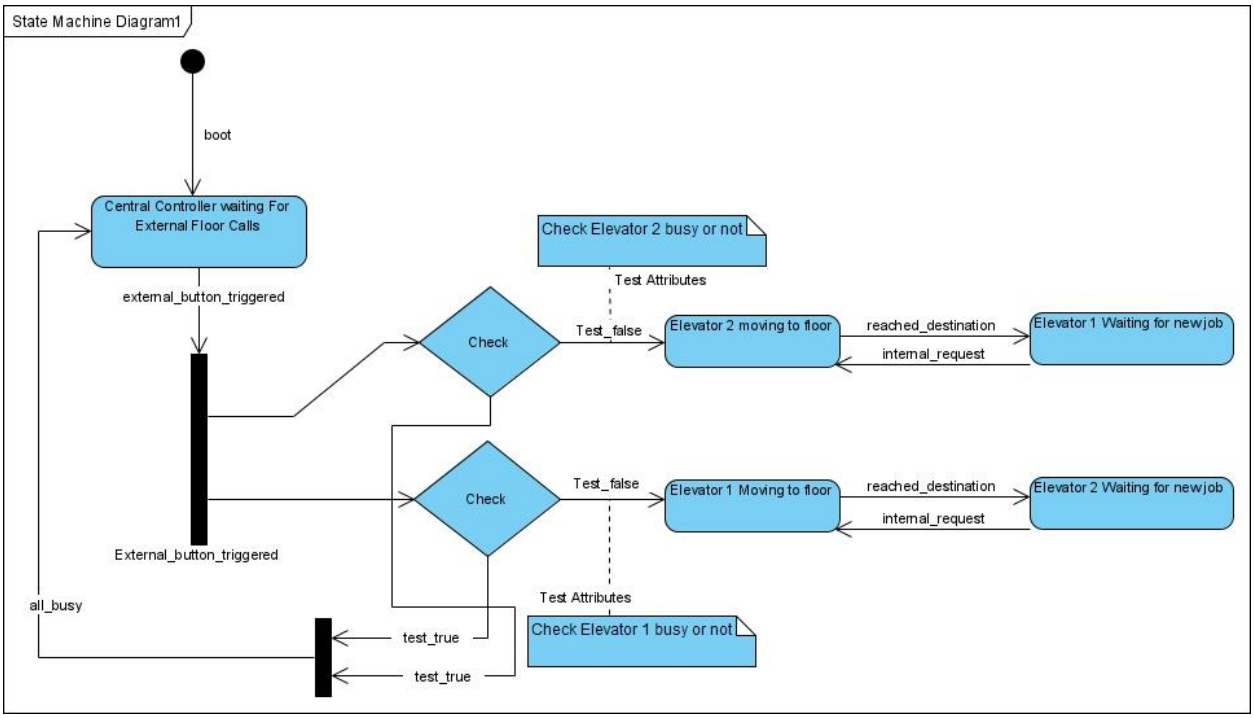

Figure 4 State Machine Diagram