Elevator Simulator

Design Principles & Algorithm

Aaron Cohn

# Overview

The application is a basic simulation of a single elevator in a building with N floors, where N is a number between 2 and 16. The simulation demonstrates that the elevator will visit each floor in a sensible order even if multiple requests are received during transit.

Conceptually, the only interface available during the simulation is the panel on the inside of the elevator.[[1]](#footnote-1) The user presses buttons to select or deselect floors, and the elevator travels to those floors based on the elevator algorithm.

# General Design

The application architecture consists of a few major parts:

1. Main application / simulator
2. Elevator
3. ElevatorDispatcher

## Main application

The main application is responsible for collecting configuration information and running the simulation. In the first phase, it prompts for the number of floors and the travel-time between floors.

Once the configuration is collected, phase two is to run the simulation. The simulation is a loop that waits for user input in the form of a floor-number, and then it feeds that floor-number to the dispatcher. It kicks out of the loop and kills the simulation when the user enters ‘q’ instead of a floor-number.

## Elevator

The elevator is responsible for simulating the behavior of traveling from one floor to another. It keeps track of its current floor, destination floor, and direction of travel. It also keeps track of which floors are selected using an array of Booleans.

To simulate traveling from one floor to the next, Elevator uses TimerTask. When the goToFloor method is called, it calculates the amount of time it would take to travel from the current floor to the destination floor, and then it creates and schedules a GoToFloorTask. The task runs immediately and calls Thread.sleep on itself for the calculated duration. When GoToFloorTask’s thread wakes up, it sends a notification to the dispatcher that it arrived at its destination floor.

## ElevatorDispatcher

The dispatcher’s responsibility is to take requests and decide where to send the elevator next. If the elevator is idle when a request comes in, the dispatcher simply calls the elevator’s goToFloor method. Otherwise, it toggles the appropriate switch on the floor selection panel using the elevator’s toggleFloorSelection method.

Elevator is loosely coupled to ElevatorDispatcher via the Event / EventListener classes in the event package. This is essentially the Observer pattern, aka Publish-Subscribe. When goToFloor finishes, it calls the notifyListeners method of the elevator’s elevatorArrived event. When the dispatcher’s ArrivalListener receives that notification, it calls the dispatcher’s sendNextRequest method, which is where the elevator algorithm resides.

## Elevator Algorithm

The simulation uses the basic elevator algorithm described on Wikipedia:

* Continue traveling in the same direction while there are remaining requests in that same direction.
* If there are no further requests in that direction, then stop and become idle, or change direction if there are requests in the opposite direction.

sendNextRequest contains two loops, one that goes up, and one that goes down. Which one goes up and which one down depends on the initial value of the elevator’s direction. Its value will be -1, 0, or +1. A value of -1 means down, while 0 is idle, and +1 is up. The loop index starts at the elevator’s current floor, and then it checks floors in the direction the elevator is going by adding the value of direction to the loop index. It continues until it either finds a selected floor or runs out of floors. If it finds a selected floor, it calls goToFloor and then quits. Otherwise, it flips the sign of direction and does the loop again to check for floors in the opposite direction. The behavior is exactly the same as before, except if it doesn’t find any floors in that direction either, then it will call the elevator’s setIdle method, which just changes the elevator’s direction to 0.

1. The initial design included a hallway call button on each floor, but during the implementation, I realized that the differences between the two interfaces were negligible in terms of their effect on the elevator’s behavior, so I simplified it to only include the inner panel interface. The design makes it easy enough to bring back the call-buttons in case the application needs to be extended for multiple elevators. [↑](#footnote-ref-1)