Smart Elevators

ID1 - 206770414 , ID2 - 322782376

In this project we are dealing with how smart elevators call system works in its optimistic way.

We are presenting three project that is relative to our project “Smart Elevators”, each of the three projects that we have found is similar; deals with large buildings, crowding at peak times, and the idea behind each project is suitable to our project.

1. **In the first link**[1], a person presses his desired floor on the keypad, the system assigns a lift to him. So, they suggested a **three-steps solution**:
   I. The system checks if any lift is on that floor, if yes, it assigns the lift.
   II. If no, it checks whether the use wants to go to an upper floor or lower floor from his current floor.
   III. It then assigns the lift which is going up and nearer to that floor.

   They designed it that each elevator has a maximum of 4 stops, in addition, each elevator is connected to an Arduino (Master), and each elevator is connected to its own Master.

   Each elevator maintains a queue, the first element in the queue is its current floor, the subsequent elements are its destination in order, and its length is 4 (the maximum number of stops).

   Pseudo Code: (only steps)

   I. Declare arrays for the slaves.
   II. Begin SPI protocol.
   III. Create as many functions as there are keypads in the building, inside each function:
   take keypad input.
   check all the filters and assign a lift.
   goes to selected slave and added to selected slave queue.

   Slave code:

   I. Declare array for the slave.
   II. Turn on slave mode + turn on interrupt.
   III. Receive destination from master, add to the queue.
   IV. Always looks at the first element of the queue and goes to that floor.
   When the destination is reached, it is deleted from queue.
   Inform master about deletion so that it can update the queue for this slave.

1. https://www.geeksforgeeks.org/smart-elevator-pro-geek-cup

2. **In the second link**[2] , they are using ThyssenKrupp Elevator that offers Destination Dispatch, a revolutionary system that focuses on passenger destinations.
   This optimization technique is used for multi-elevator installations, in which groups of passengers heading to the same destinations use the same elevators, overall trip-times can be reduced by 25% with capacity up by 30%.
   I. passengers use the keypad to register what floor they are traveling to.
   II. The keypad will graphically direct each passenger to the appropriate car for their destination.
   III. Once the car arrives, lobby position indicators allow the passengers to see the designated stops for the elevator.
   IV. By dispersing passengers evenly among various cars, they get to their destination faster.

3. **In the last link**[3] , they adopted a control system called destination dispatch which helped them reduce the crowd in a building that contains 45 floors with 4000-8000 visitors.
   When passengers enter their floor on the keypad before they board the system uses a totally different algorithm to decide which car should answer the call the control system takes the information from each of the waiting passengers and tries to put people with who are going to a common destination on the same car so that will minimize the number of stops
   The number of stops is cut in half making travel time shorter and boosting the capacity of the whole system by a whopping 30%.

Online/Offline Algorithm: In general, the main difference between Online and Offline algorithms is that in Online optimization, we have to make decisions before all data are known (based on a real-time algorithms), on the contrary, in Offline algorithm the whole problem data is given from the beginning.

Online: for our project, we are going to use three methods:

I. The priority is to find the available elevator which is located at the "LEVEL = 0" level of the building.
II. The second priority for the closest elevator which is in the same direction as the passenger's direction.
III. The third and last is picking a random elevator if the two choices above aren't available.

If the building has more than one floor, let's suppose that the building has $\boldsymbol{n}$ floors, in this situation, if the elevator still didn't reach the top (last floor in the building = $\boldsymbol{n}$) then go up one floor above, same as if the elevator didn't reach the bottom (first floor in the building) then go down one floor beneath, if so and these the elevator reach the top/bottom, then we calculate a partition for the building which works this way: Number of floor divided by 2, in this way the elevator stops in the middle of the building waiting for a new call.

2. https://web.archive.org/web/20120718231009/http:/thyssenkruppelevator.com/destdist.asp
3. https://www.youtube.com/watch?v=T6gzm_ifzg8&t=212s&ab_channel=cplai

Offline: in case of the whole data is given from the beginning, then we program an algorithm to work in this way:

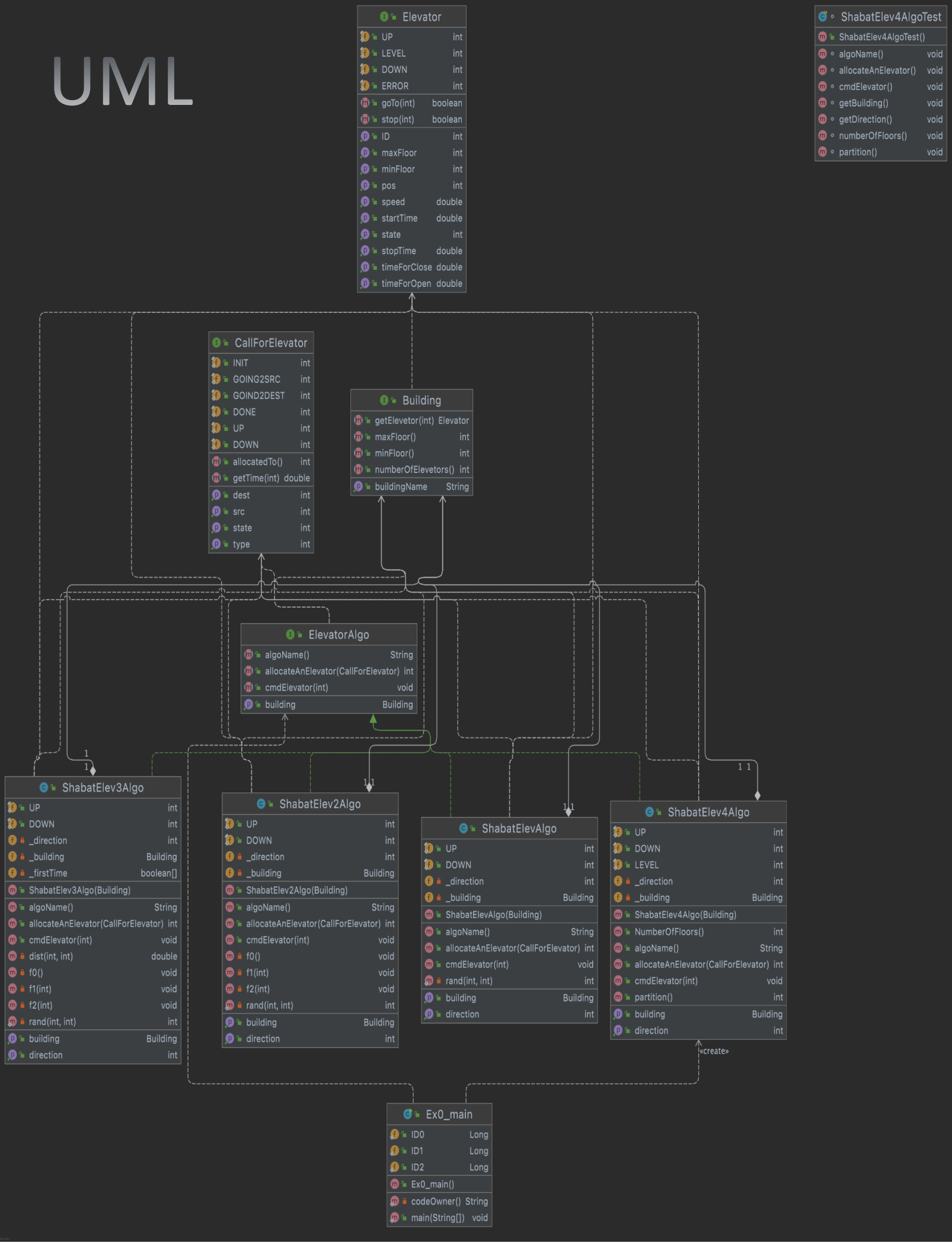
I. Firstly, we save the data of how much the building has floors, elevators, visitors, and gusts/workers (in case this information is given), in addition and most importantly we check in which floors in current time does it get crowded by the people, to which floors are they going to, and in this way, we optimize our code to work well with this situation.
Mostly we can separate the elevators to the floors in this way.

II. If there is only one elevator, then it will be programmed by the closest and the same direction call.

III. If the building has more than one elevator, let's suppose that the building has $\boldsymbol{n}$ elevators, in this situation, we separate it into two parts:
   a. The building has an **odd** number of elevators: in this situation, we divide the building into $\left\lfloor \frac{\boldsymbol{n}}{\mathbf{2}} \right\rfloor$ sections, in each section, there are two elevators, one of these elevators answers only down calls and the other one answers up calls (they only answer calls in their sector), except one section that has only one elevator that answers all calls in its section (goes up and down), in this situation there is $\left\lfloor \frac{\boldsymbol{n}}{\mathbf{2}} \right\rfloor$ elevators which goes up, $\left\lfloor \frac{\boldsymbol{n}}{\mathbf{2}} \right\rfloor$ elevators which goes down, and one elevator that goes up and down.
   b. The building has an **even** number of elevators: in this situation, we divide the building into $\frac{\boldsymbol{n}}{\mathbf{2}}$ sections, in each section, there are two elevators, one of these elevators answers only down calls and the other one answers up calls (they only answer calls in their sector), in this situation there is $\frac{\boldsymbol{n}}{\mathbf{2}}$ elevators which goes up, $\frac{\boldsymbol{n}}{\mathbf{2}}$ elevators which goes down.

In case of some floors got crowded as we can know from the statistics then we call the elevators in the other floors to help minimize the crowd in the waiting floors.

UML


Elevator
UP int
LEVEL int
DOWN int
ERROR int
goTo(int) boolean
stop(int) boolean
ID int
maxFloor int
minFloor int
pos int
speed double
startTime double
state int
stopTime double
timeForClose double
timeForOpen double
ShabatElev4AlgoTest
ShabatElev4AlgoTest()
algoName() void
allocateAnElevator() void
cmdElevator() void
getBuilding() void
getDirection() void
numberOfFloors() void
partition() void
CallForElevator
INIT int
GOING2SRC int
GOIND2DEST int
DONE int
UP int
DOWN int
allocatedTo() int
getTime(int) double
dest int
src int
state int
type int
Building
getElevetor(int) Elevator
maxFloor() int
minFloor() int
numberOfElevetors() int
buildingName String
ElevatorAlgo
algoName() String
allocateAnElevator(CallForElevator) int
cmdElevator(int) void
building Building
1
1
1 1
1 1
1 1
ShabatElev3Algo
UP int
DOWN int
_direction int
_building Building
_firstTime boolean[]
ShabatElev3Algo(Building)
algoName() String
allocateAnElevator(CallForElevator) int
cmdElevator(int) void
dist(int, int) double
f0() void
f1(int) void
f2(int) void
rand(int, int) int
building Building
direction int
ShabatElev2Algo
UP int
DOWN int
_direction int
_building Building
ShabatElev2Algo(Building)
algoName() String
allocateAnElevator(CallForElevator) int
cmdElevator(int) void
f0() void
f1(int) void
f2(int) void
rand(int, int) int
building Building
direction int
ShabatElevAlgo
UP int
DOWN int
_direction int
_building Building
ShabatElevAlgo(Building)
algoName() String
allocateAnElevator(CallForElevator) int
cmdElevator(int) void
rand(int, int) int
building Building
direction int
ShabatElev4Algo
UP int
DOWN int
LEVEL int
_direction int
_building Building
ShabatElev4Algo(Building)
NumberOfFloors() int
algoName() String
allocateAnElevator(CallForElevator) int
cmdElevator(int) void
partition() int
building Building
direction int
«create»
Ex0_main
ID0 Long
ID1 Long
ID2 Long
Ex0_main()
codeOwner() String
main(String[]) void