Lift Algorithm

by Khalid Olowe-Makorie

Table of Contents

Main Task

Important things to note:

- In this solution the 10 floors of MedicineChest are treated as **1 Ground floor and 9 upper floors.** Hence the floor order goes ground floor, floor 1, floor 2 and so on till the top and final floor 9.
- The workers are evenly distributed among the floors therefore no floor has priority over the others
- People start work at 8AM and everyone has left by 6PM. This means that around 8AM there will be more people entering the lift from the ground floor than any other floor. This also means that towards 6PM there will likely be more people entering from the above 9 floors to go down to the ground floor.
- The lift only has a single call button, no up or down button. Therefore, we cannot determine whether a person wants to go up or down when they call the lift to arrive at their floor. This will only be known once the lift has arrived at said floor and the person chooses a desired destination floor inside the lift.

My general approach

The lift algorithm will function as a **Finite State Machine (FSM)** where the lift constantly transitions between **6 states**:

1. When the lift is empty and there are no external floor calls.
2. When the lift is empty and there are one or more external floor calls.
3. When the lift is occupied but not at max capacity and there are no external floor calls.
4. When the lift is occupied but not at max capacity and there are external floor calls.
5. When the lift is at max capacity (8 people).
6. When the lift has been put into an emergency state i.e. the alarm button has been pressed.

There are also some important definitions that relate to these states:

External floor call - Any call that has been made by a person outside of the lift by pressing the external call button.

Desired destination floor – A floor that has been set from within the lift using the internal buttons to indicate where a person would like to get off.

The state of the lift algorithm will only be recalculated in the following instances:

- The external call button on any floor has been pressed
- An internal floor number button has been pressed to indicate a desired destination floor
- When the internal weight of the lift changes i.e., someone gets on/off
- When the emergency button is pressed

These 6 states were chosen because they are very distinct and simple. Once the status of the emergency button, the number of people in the lift and the number of external calls is known, we can easily determine which state the lift is in and trigger the desired lift behaviour. Moreover, the lift does not need to constantly reassess its state. It is only when a key factor of the lift changes i.e., the number of external floor calls, that the lift's state might be affected. By reducing the number of cases in which the state needs to be redetermined, we can minimise the amount of processing time and power required by the algorithm.

Furthermore, whenever the lift starts moving in a given direction, it always completes all work in that direction before it considers going back. For example, if the lift is on the ground floor and two external calls are made, one on floor 1 and one on floor 5. The lift will move first to floor 1, then to floor 5 before even considering going back down towards the ground floor, regardless of whether the people on floor 1 selected to go down to the ground floor or not. This is to ensure that people at the far ends of the building i.e., ground floor or floor 9, will always be reached. Alternatively, if the lift were to always move according to the closest desired destination floor regardless of direction, those people at the far ends could get stuck on their floors for large amounts of time. For instance, if the lift on the ground floor is called to floor 1 and floor 5 again but the people on floor 1 choose to go down to the ground floor. The lift would then go back down to the ground floor as its closer than floor 5. Now on the ground floor the lift would then move back up to respond to the call on floor 5, but if during transit it is called to floor 2 and those people also choose to go to the ground floor, the people on floor 5 are stuck once again and forced to wait. This could repeat and result in the lift never arriving on floor 5. Consequently, the current direction of the lift is prioritised over the absolute nearest desired destination floor.

I acknowledge that prioritising direction is flawed in the sense that people on lower floors might be taken all the way up to floor 9 before being able to go down to the ground floor. However, I have deemed this more acceptable than the potential infinite wait loop that occurs if the absolute nearest desired destination floor is prioritised.

I will now discuss the behaviour of the lift in each individual state.

State 1 – Empty lift, no external floor calls

In this state the lift has nowhere it absolutely needs to be, however, it can be placed well in anticipation of future calls. Around the 8AM we can predict with reasonable accuracy where most people will enter, thus the current time most determines the lift's behaviour in this state.

In this state, from the early morning till to 8AM the lift will default to move back to the ground floor. This is to anticipate all the workers arriving to work in the morning and moving up to their offices in the above floors. "Early morning" is a flexible boundary and will be defined by the earliest time people tend to come to work, say 6AM or 7AM.

For the remainder of the day the empty lift will default to moving to floor 5. As no floor has priority over the others, due to workers being evenly distributed, it is optimal to put the lift in the middle of the building for faster access to most floors. Granted this does benefit the middle floors the most, it is more equal than placing the lift closer towards the top or bottom floors. Although we are aware that most people will be returning to the ground floor to leave the building at around 6PM, there is still an equal number of people coming from each floor. Thus, placing the lift in the middle floor is still optimal.

State 2 – Empty lift, one or more external floor calls

In this state the lift merely moves to the nearest floor that it has been called to. This will also set the lift direction which plays a significant role in states 3 and 4.

If the lift is called to an above floor, the lift direction will be set to UP. If the lift is called to a floor below it's current position, then the lift direction will be set to DOWN.

State 3 – Occupied but not full lift, with no external floor calls

In this state, the lift will go to the nearest desired destination floor in the current lift direction.

There are only 2 instances in this state in which the lift direction is set:

If the lift arrives at the ground floor its direction will be set to UP, and if the lift arrives at floor 9 its direction will be set to DOWN. This is because the lift cannot move beneath the ground floor or above floor 9.

Moreover, the lift will always move in the current lift direction until either:

1. Floor 9 or the ground floor have been reached
2. The highest desired destination floor is reached with lift direction=UP
3. The lowest desired destination floor is reached with lift direction=DOWN

   This is to ensure all desired destination floors are reached in the current direction before those in the opposite direction are dealt with. There are no external floor calls hence they are not factored into this decision.

State 4 – Occupied but not full lift, with one or more external floor calls

The lift behaviour in state 4 is almost identical to that in state 3, other than the fact that external floor calls are factored into the lift movement. As in state 3, the lift will go to the nearest desired destination floor in the current lift direction. However, the lift will also stop at any floors on the way in which external floor calls have been made.

The lift direction will also be set in the same way as in state 3 if the lift arrives at the ground floor or floor 9.

The lift will always move in the set lift direction until either:

1. Floor 9 or the ground floor have been reached
2. The highest external floor call or desired destination floor is reached (whichever is higher) with lift direction=UP
3. The lowest external floor call/desired destination floor is reached (whichever is lower) with lift direction=DOWN.

In these cases, the lift direction would be inverted and the lift would move in the opposite direction.

State 5 – Lift is at max capacity

In this state the lift is at max capacity and therefore cannot attend to external calls due to a lack of space. As a result, the lift ignores any external calls and just goes to the nearest desired destination floor in the current direction.

As with states 3 and 4, the lift direction will be inverted if the lift arrives at the ground floor or floor 9.

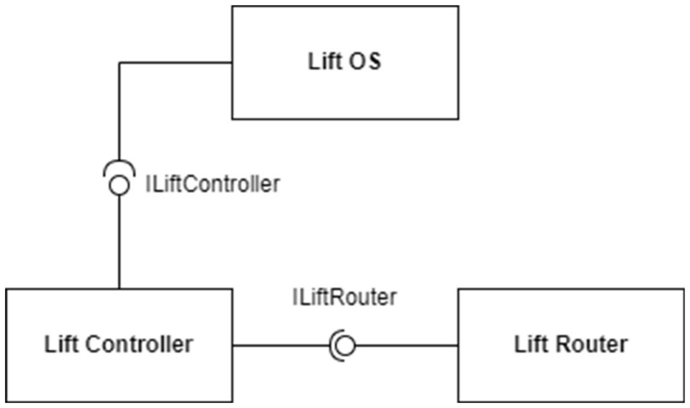
State 6 – Emergency state

The action of the lift in this state depends on the kind of emergency and the emergency procedures at MedicineChest. However, in most cases if the internal alarm button is pressed or there is a hardware fault in the lift, it is best for the lift to either stop moving entirely until the emergency state has been resolved or to move to the nearest immediate floor and wait.

Technical Details of System Architecture

I will now detail the system architecture. This approach is made under the assumption that there is a lift Operating System (OS) which connects the lower-level lift functions to the higher-level abstraction where the lift routing algorithm resides.

There are two major classes that control the lift in this higher-level abstraction. There is the lift router which is the algorithm discussed so far and determines where the lift will go next, and the lift controller which the lift OS interacts with to manage and run the lift router. Both of these classes are accessed through their respective interfaces to enable the modularity and extensibility of the system.



Lift Controller

To ease communication between the lift OS, the lift controller is accessed through an interface called ILiftController (see below). The ‘list of desired floor destinations’ and ‘list of external floor calls’ are merely dynamic data structures used to store the desired floor destinations and external floor calls. These could be implemented as linked lists or dictionaries/hashmaps. The only major requirement is that each node stores a single integer value to represent the floor number and can be quickly added and removed from the data structure.

As previously mentioned, the state of the lift router should only need to be updated in 4 major cases. However, the lift OS will only need to know where it should send the lift next at certain times i.e., when people have moved inside the lift and the doors have closed. Therefore, it is a waste to constantly recalculate the lift router state whenever one of those 4 specified cases occurs. Instead of this, the lift controller compiles all the information needed to calculate the new state through the use of the first 5 methods in its interface. Then, when the lift OS is ready it calls the GetNextDestination function to calculate the new state and obtain the new target floor (where the lift should move to next).

Pseudocode:

```
Interface ILiftController {

void FloorArrival(current floor) //this function would be called every time the lift arrives at a floor to determine what floor the lift is currently on, it also removes the current floor from the list of desired destination floors and the list of external floor calls if it is in any of them
void WeightChange(current number of people in lift) //this function would be called every time the weight within the lift changes and calculates the number of people currently in the lift

void InternalFloorButtonPress(internal floor number button pressed) //this function is called every time an internal floor number button is pressed and adds this floor to the list of desired destination floors

void ExternalFloorButtonPress(external floor number button pressed) //this function is called every time an external floor button is pressed and adds this floor to the list of external floor calls

void EmergencyButtonPress(emergency status) //sets the emergency condition to true or false, will be factored in when the new current state is calculated

Integer GetNextDestination() //updates the state of the lift router with values set from the above methods and returns the new target floor

}
```

The Lift controller class will implement this interface and have class variables to store the emergency state, list of external floor calls, list of desired destination floors, the current floor, the current number of people in the lift, as well as an instance of the lift router interface.

Lift Router Interface

The lift router interface has one function called UpdateState() which takes as parameters the time of day, the current floor the lift is on, a list of the desired destination floors, a list of the external floor calls, the number of people in the lift, and a Boolean value indicating if an emergency has been triggered or not. This function will return the floor number that the lift should move to. The lift controller will call this function whenever the lift OS requests the new target floor.

Pseudocode:

```
Interface ILiftRouter{

Integer UpdateState(time, current floor, list of desired destination floors, list of external floor calls, number of people in list, is there an emergency)

}
```

Lift Router class

The lift router class will then implement the ILiftRouter interface. It should have a class variable for the current lift direction, this could represented as an enum with values {UP, DOWN} or a simple Boolean like goingUp. As default this variable should be in the up direction as the lift will start on the ground floor.

Each state in the lift FSM has its own method which when run returns the new target floor. The UpdateState() function merely determines which state method it needs to call based on its given parameters and returns the output of that method call.

Extension 1

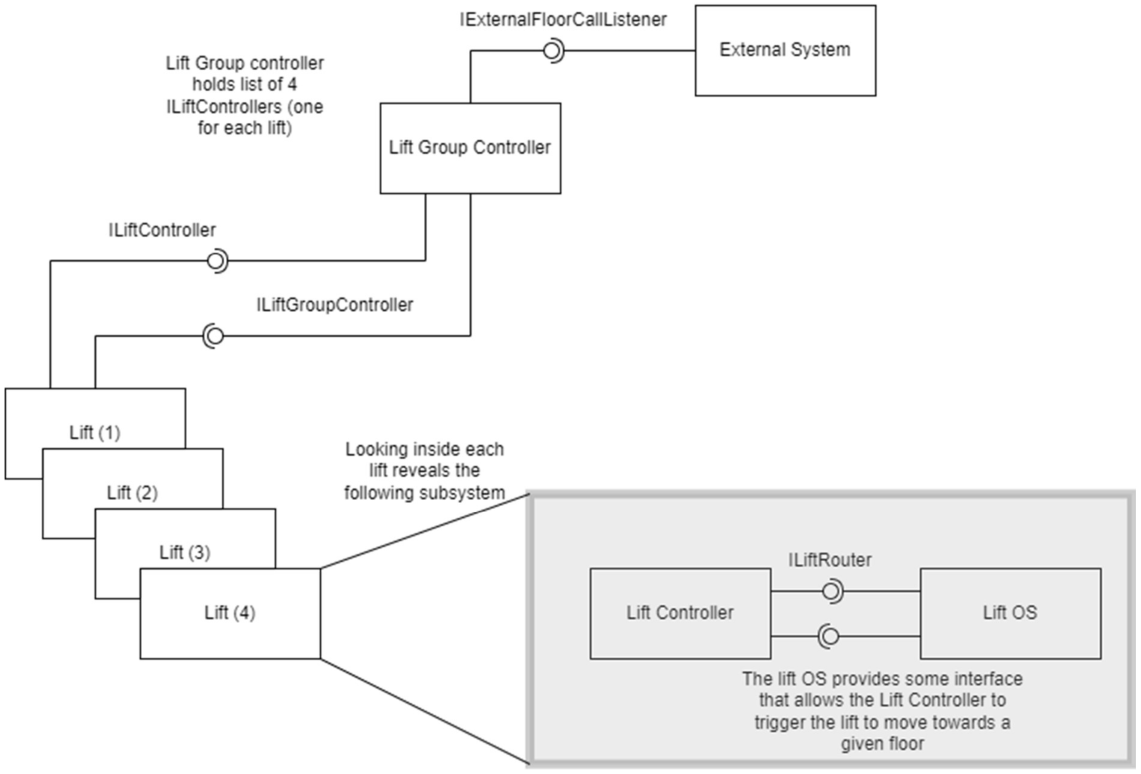
For this new scenario there would be several changes made to the previous system. Firstly, as there are 4 lifts there is a need for a higher lift group controller which manages the operation of the 4 lifts. Each individual lift would also have less control over its routing. External floor calls would now become the responsibility of 4 lifts instead of just one, thus it is up to the lift group controller to decide which of the 4 should handle a given call. People are also not distributed evenly among floors and therefore lifts should favour certain floors over others instead of being equally spaced between them.

There are some constants though, for example, the fact that we still cannot determine whether a person wants to go up or down when they call a lift to a floor.

The New System

I will now briefly explain how the different parts of the system interact and then will talk about them each individually to provide more detail on how they function.

Each individual lift has its own lift controller script that dictates its individual movement. These lifts are no longer explicitly FSMs but do transition between 2 major states: Moving to a destination floor or Idle. They handle their own desired destination floors themselves and choose which to go to. However, the lift group controller is notified whenever an external floor call is made and can interfere with lifts and override their next destination to stop at an external floor call first before their original destination. Each lift controller also notifies the group lift controller if they're available to handle an external floor call or if they're idle, so that it can either send them to different floors to handle external floors calls or in anticipation of future calls.


IExternalFloorCallListener
External System
Lift Group controller holds list of 4 ILiftControllers (one for each lift)
Lift Group Controller
ILiftController
ILiftGroupController
Lift (1)
Lift (2)
Lift (3)
Lift (4)
Looking inside each lift reveals the following subsystem
ILiftRouter
Lift Controller
Lift OS
The lift OS provides some interface that allows the Lift Controller to trigger the lift to move towards a given floor

Lift Controllers

There will be two types of lift controllers. The first type, DirBasedLiftController, will handle desired destination floors by choosing an initial direction then visiting all desired destination floors in that direction before handling the ones in the opposite direction. This is similar to the lift in the first lift algorithm. The second type, DistBasedLiftController, will handle desired destination floors by visiting the closest desired destination floor. The purpose of having these two types is to accommodate the larger amount of people taking the lift. The first lift type ensures that all floors, such as those on the far ends, are reached within a reasonable amount of time. Whereas the second type aims to minimise the time people spend in the lift and reduce queuing by prioritising delivery speed. This is significant as there will likely be much more queueing in the hotel due to its large size. The number of each type of lift controller depends on the distribution of the popular floors. If there is a small region of the hotel that is most popular, then deploying one or two DistBasedListControllers in that region would be beneficial. However, if the popular floors are spread out over large distances, then having more DirBasedLiftControllers would be better.

These lift classes share a lot of the same functions. The only difference between these lift types is their calculation of the next desired destination floor to go to. As a result, there will be a common LiftController abstract class which both lift types extend. This common class will then handle everything except that target floor calculation, to reduce repeated code.

The LiftController abstract class will implement the ILiftController and ILiftRouter interfaces. The ILiftController interface is used to handle interactions between the lift group controller and the lift controllers. The ILiftRouter interface is used by the lift OS to notify the lift controller of important events that impact its routing.

The pseudocode for the ILiftController and ILiftRouter interfaces as well as the LiftController abstract class can be seen below.

Pseudocode:

```
Interface ILiftController {

Void SetExternalFloorCall(the floor on which the call was made) //Used by the lift group controller to override the current destination and set it to handle an external floor call along the way.

Boolean isBusy() //Used by lift group controller to determine if a given lift is available to handle external floor calls. A lift is deemed busy when it is already handling an external floor call or is at max capacity.

Integer GetCurrentFloor() //Returns the current floor that the lift is on. Used by lift group controller to determine which external floor call it should send a given lift to.

GetDirection() //Returns whichever direction the lift is currently moving in, this could be an enum of UP or DOWN or an integer that's set to 1 for up and 0 for down, etc. The lift group controller calls this function when it needs to determine which external floor call it should send a given lift to.

GetCurrentDestination() //Returns the desired destination floor that the lift is currently moving towards

}
```

Interface ILiftRouter {

Void SetDesiredDestinationFloors(list of desired destination floors) //Will be called by the lift OS to provide the lift controller with the current list of desired destination floors

Void ArrivedAtFloor(the current floor) //Called by the lift OS to indicate to the lift controller that is has arrived at a given floor. This is used by the lift controller to trigger the calculation for the next destination floor based on the desired destination floor list.

}

Abstract class LiftController implements ILiftRouter and ILiftController{

//should have variables to store: the current floor, the current lift direction, the target desired destination floor that the lift aims to move to, the floor of an external floor call if its handling one, and a reference to the running instance of the lift group controller interface

//should also have a dynamic data structure to store the desired destination floors

Void SetDestination() //here the lift should check if it has an external floor call to handle before its next desired destination floor. If the lift has an external floor call it should tell the lift OS to move there. If there is no external floor call it should tell the lift OS to move to the target desired destination floor.

Void SetExternalFloorCall(the floor on which the call was made) //part of ILiftController, this function sets the external floor call variable to the parameter. It then calls the SetDestination function to update the lift's next destination so that it handles this external call before its current destination.

Void ArrivedAtFloor(the current floor) //If the current floor was in the desired destination floor data structure, then it should be removed from that structure. After this it should call CalculateNextDestination function to calculate the next destination floor. However, If the current floor was an external floor call, then the external floor call variable should first be reset to nothing. Then as the next desired destination floor has already been calculated, it should call SetDestination to trigger the lift to move towards that floor. At the end of this function the lift should also notify the LiftGroupController if it is available to handle another external floor call.

unimplemented Void CalculateNextDestination() //this is implemented by the DirBasedLiftController and DistBasedLiftController subclasses to calculate the next desired destination floor that the lift should go to based off their routing priorities. When implemented, this function should also call the SetDestination function when its finished in order to start moving the lift towards the calculated destination floor

//All other interface functions should be implemented

}

Lift Group Controller

The lift group controller holds 2 lists, one for the 4 lift controllers (which are accessed through the ILiftController interface) and another to store the external floor calls.

Every time an external floor call is made, the lift group controller is notified of that floor and adds it to the list of external floor calls. The group controller then iterates through the external floor calls and finds which lift in its lift controller list can handle each external floor call. Empty lifts are always chosen over non-empty lifts to handle these calls. A non-empty lift is only sent to handle an external floor call if it is the closest of all the other lifts that are moving in the direction of that call. This is to ensure that non-empty lists only handle external floor calls on the way to their current destination as not to hinder their current operation. A lift that is at max capacity is never sent to handle an external floor call. If all the lifts are full then the lift group controller does send any lifts to handle any of the calls.

The lift group controller is also notified by a given lift controller if that lift is suddenly available to handle an external floor call i.e. it has just arrived at a floor and is not full. The lift group controller can then decide if there is an external floor call that that lift can be sent to handle. If so, it'll call the SetExternalFloorCall function of that lift controller.

The final case in which the lift group controller is notified, is when a given lift is idle i.e., is empty and has no desired destination floors nor external floor calls to handle. In this case the lift will send these lifts to popular floors in anticipation of future external floor calls. Each floor should have a weight to signify how popular it is compared to the others. This way the proportionate number of empty lifts can be sent to each popular floor.

The lift group controller will implement 2 interfaces: the ILiftGroupController for interactions between the LiftGroupController and LiftControllers, and the IExternalFloorCallListener to listen out for external floor calls.

The pseudocode for these interfaces can be seen below.

Pseudocode:

```
Interface ILiftGroupController {

Void NotifyWhenIdle(the lift that has now become idle) //this is called by the lift controllers to indicate to the group controller that they are idle and do not have a current destination. The group controller will decide here which floor to send them each idle lift to.

Void NotifyWhenAvailable(the lift that is now available) //this is called by a lift controller to indicate that although not be empty, they are available to handle an external floor call if it is on the way to their current destination.

}


Interface IExternalFloorCallListener {

Void NewExternalFloorCall(floor on which the call was made) //Used to notify the lift group controller of a new external floor call.
}
```

Extension 2

When the new lift buttons are added, it will finally be possible to know beforehand if a given person waiting for a lift wants to go up or down. The lift group controller would then be changed to only send lifts to handle external floor calls in which the call direction and lift direction coincide. All other aspects of the lift system from the first extension should remain the same.