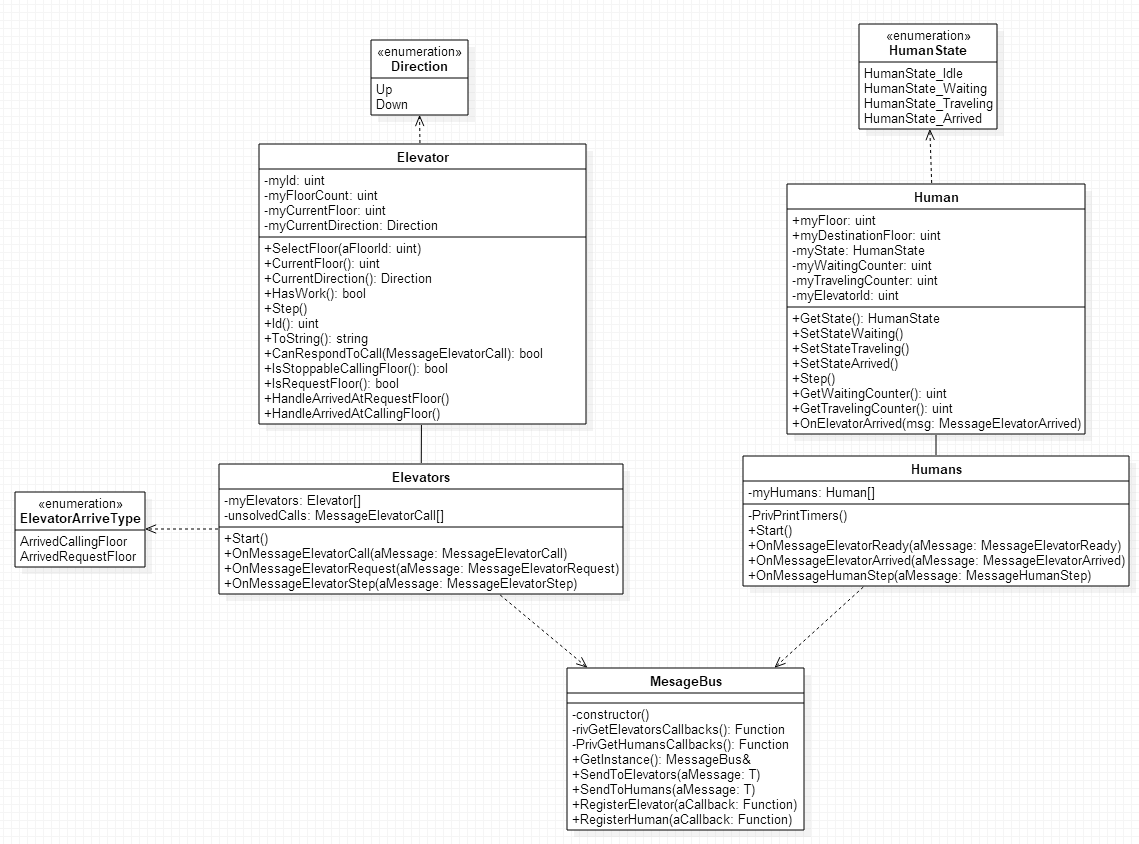
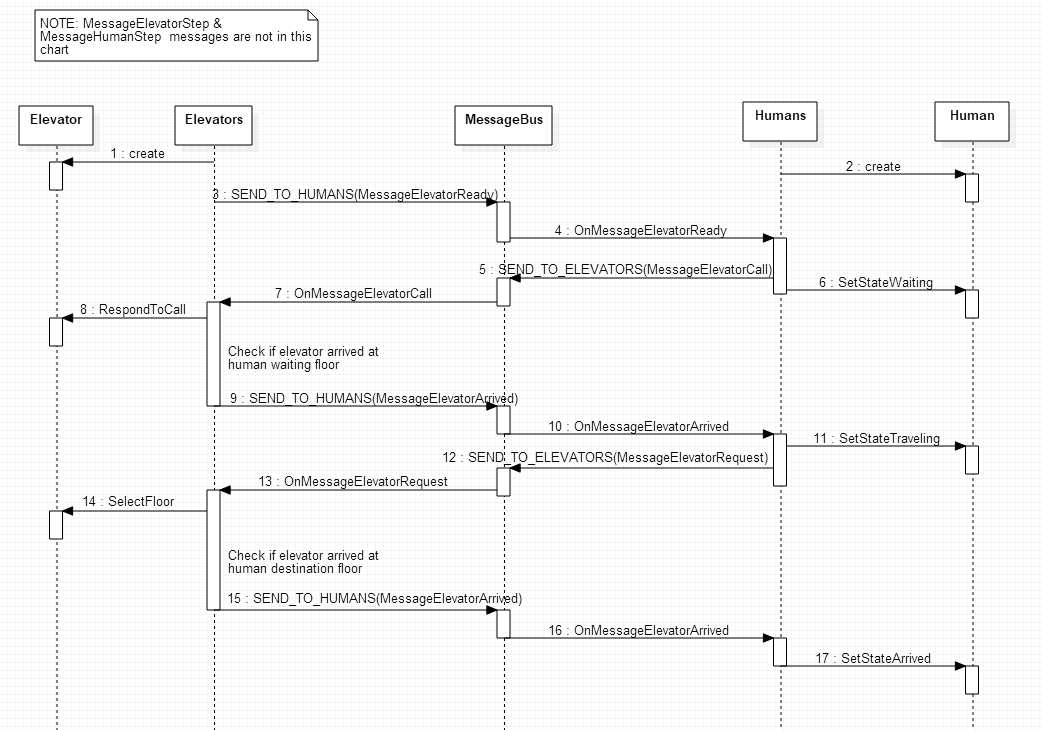
**Elevator System**

1. **Introduction**

**1.1 main classes diagram of the system**

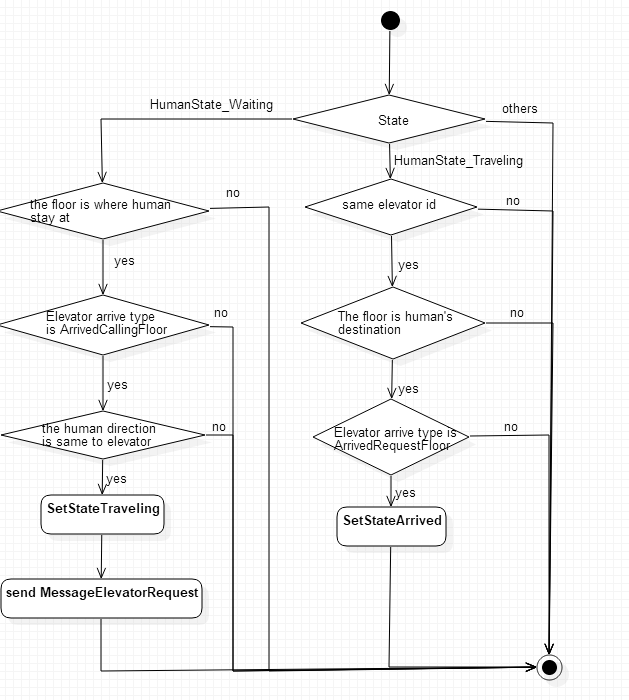


**1.2 Basic message and call flow between an elevator and a human**



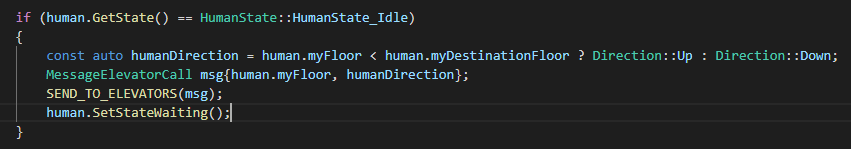
1. **Humans module design and implementation**
   1. **Handle MessageElevatorArrived**

For class Human, I added a private member myElevatorId(unsigned int) to indicate which elevator the human is in when human in travel. And added a public method “void OnElevatorArrived(const MessageElevatorArrived&)”, This method is to handle message MessageElevatorArrived. The general procedure for Humans handle MessageElevatorArrived is as below:



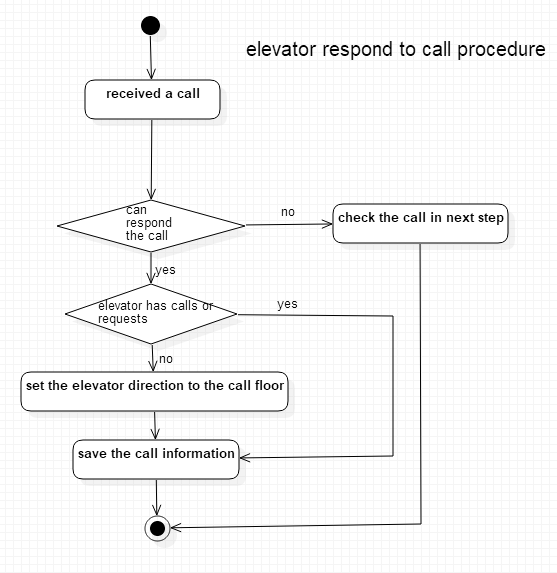
* 1. Handle MessageHumanStep

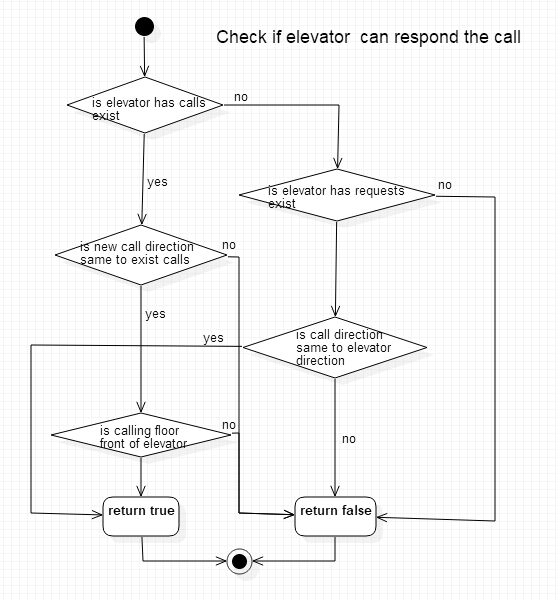
Human only care if the elevator is arrived and if elevator reached the destination floor. So in Humans::OnMessageHumanStep, what I added is only to change Human Idle state to waiting state, and send the message MessageElevatorCall. The related code as below:



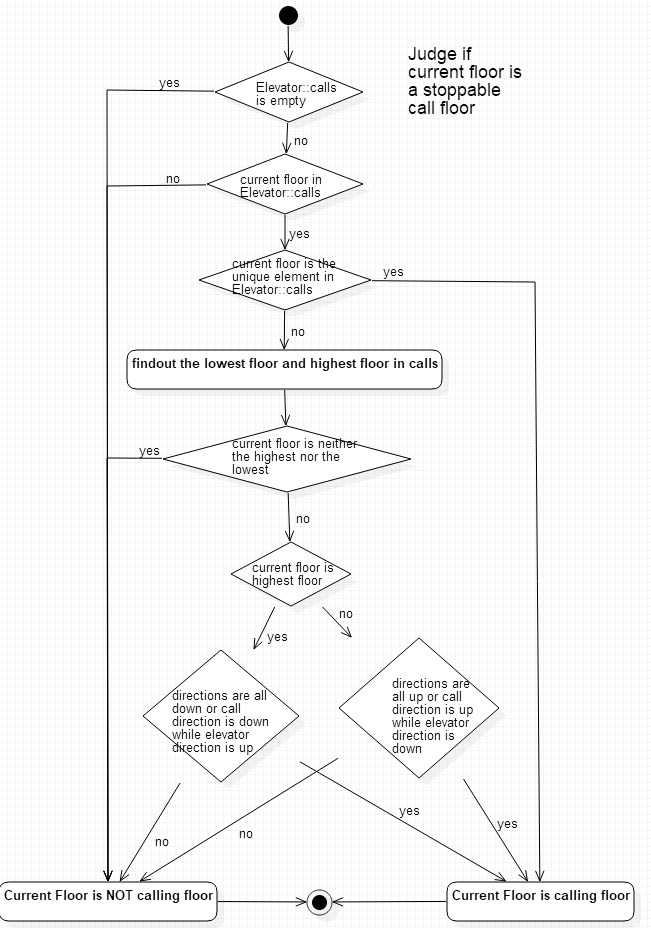
1. **Elevators module implementation**
   1. **Handle MessageElevatorCall**

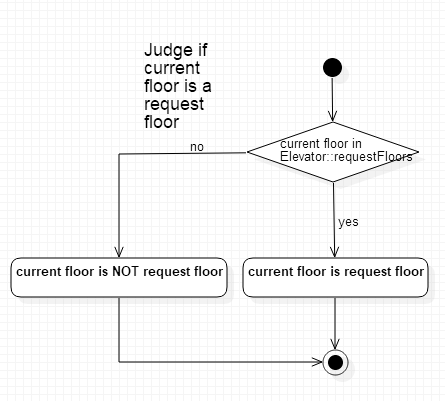
For one elevator scenario, I choose LOOK algorithm to schedule it. The basic strategy is when the elevator received first call, and this call will determine the elevator’s running direction. When received new call, check whether this call can be handled, if not, store the call, and check again in next elevator step, until this call is responded.





The logic to check whether the elevator is stopped to respond a call or a request:



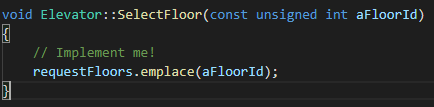


For multiple elevators scenario, I choose the closest elevator from all elevators that can respond the call to respond the call.

* 1. **Handle MessageElevatorRequest**

I added a private member “requestFloors (std::set<unsigned int>)” in class Elevator, it represents this elevator has to go to these floors. When elevator is received the request, the destination floor of the request should be stored in requestFloors. Detail code as below:



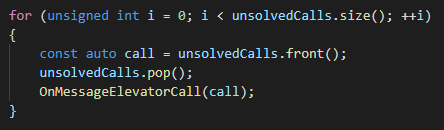


* 1. **Handle MessageElevatorStep**

Elevators mainly do two things here, first one is to check whether any elevator needs to stop at this floor (calling floor or request floor), and do relate operations. Another one is to check if unsolved calls can be responded, and do relate operations.

For first thing, I check the elevators one by one, if current floor is a suitable call floor or request floor, send the MessageElevatorArrived to Humans.

For second thing, I added a queue to save the unsolved calls, if the call is responded, the call will be deleted in the queue, otherwise, the call will be re-added at the end of the queue. Detail code as below (if call cannot be responded, this call will be added again in OnMessageElevatorCall):



1. **Tests**

For detail test result, please see: tests.xlsx

For this system, the main test point is the average waiting time. Because this system only take passengers in one direction, so the average travelling time has no place to optimize.

* 1. **Calculate the average waiting time**

Since when Humans send the call to elevators, it can only be handled in next step, there is an extra 1 is added for every human waiting counter. And when elevators send the MessageElevatorArrived back, it will be handled by Humans in next step, so there is another extra 1 will be added. So the expected average waiting time formula is:

Above N is the human amount, EF is elevator floor, HF is human waiting floor

From above formula, the minimal average waiting time is 2.

1. **bonus question**
   1. **How would you enhance the Threads class to be able to execute work on a specific thread after a specific delay?**

For current system, just not to notify the condition\_variables can keep work executed after a specific delay. For example, set the locThreadWaitTimeMs to 1000, 2000, and etc. But for different delays, we can consider introduce timer, when timer expired, notify the relate condition\_variables, and execute the work.

* 1. **In the supplied code, the system runs forever until the process is terminated. How would you perform a clean exit?**

I would prefer to use an atomic variable to monitor the system status. when the system received exit signal, change the atomic variable value, then exit the dead loop.

* 1. **How would you extend the system to also consider buildings containing n elevators and n humans, where buildings can be of different sizes, that is to extend the system to act as a centralized controller?**

I would design this system as a distribution system. each building should have its own scheduling system (just configuration different). And the center controller only monitors the distribution units and control the elevators' running status. And to compress the waiting time, we may should consider set the elevators always stop at the floors with more humans in a duration when has no passengers.