OSP STAGE 2040 for Project FluxVator

T3 Qui Shibo 201013760 Taeho Kang 201013275 Ingoo Lee 200913215 T3

Real Use Cases

UI Design

System Architecture

Interaction Diagram(Sequence)

Class Diagram

Ending Remarks

Real Use Cases

1. The Big Picture Remains the Same. (for now.)

1. The Big Picture Remains the Same. (for now.)

18 actual Use cases in total

6 evident, 12 hidden

Unchanged from essential use cases

Use Case Change Notes

2. Details, Details and More Details.

2. Details, Details and More Details.

Step by step normal flows

Reference variables & operations for relevance

>Gritty details are subject to changes.

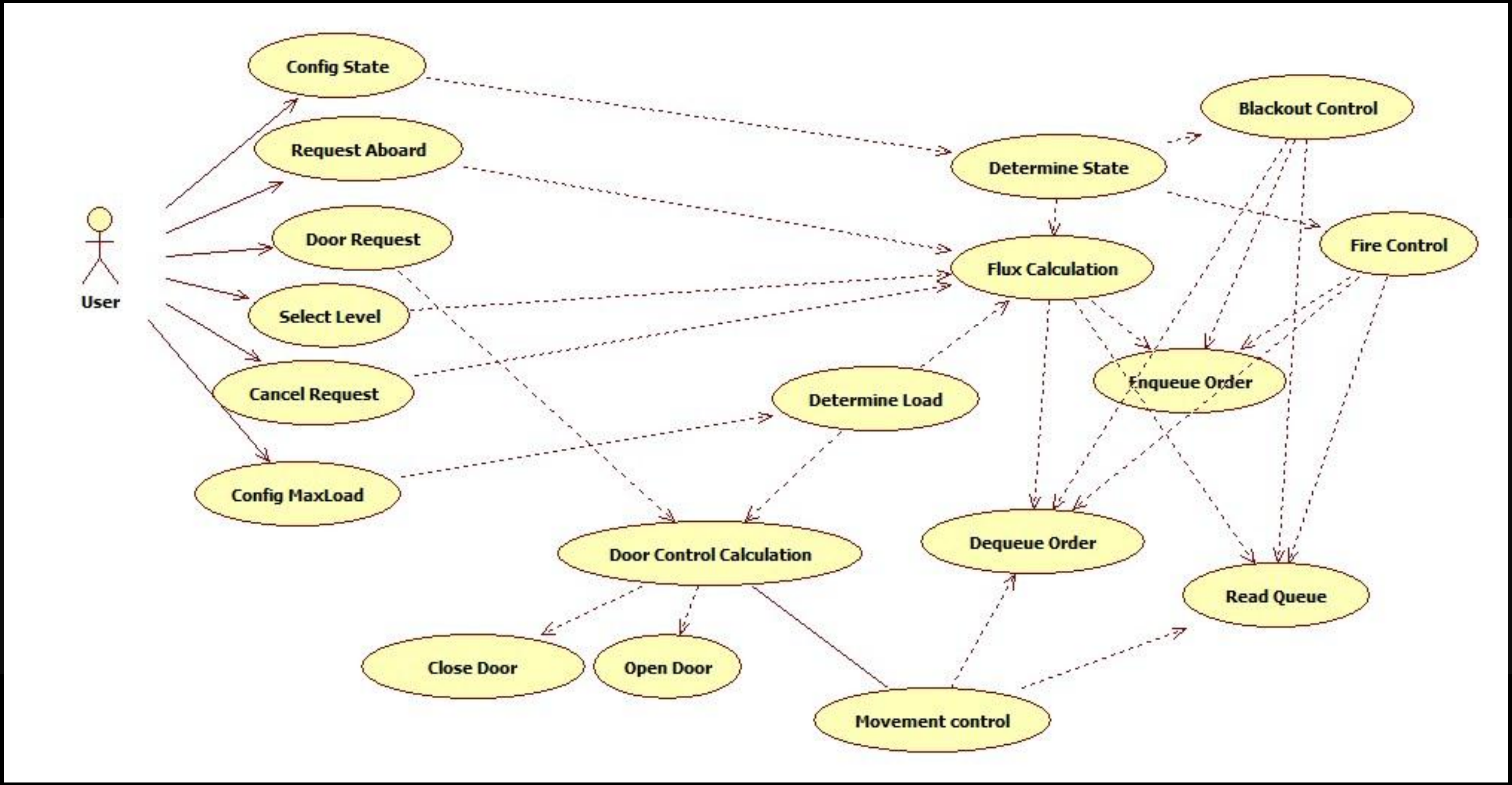
3. Actual Flow Elaboration

3. Actual Flow Elaboration

Will be done in sequence diagrams.

For more info on use cases,

refer to our 2040 document.



Use Case Change Notes

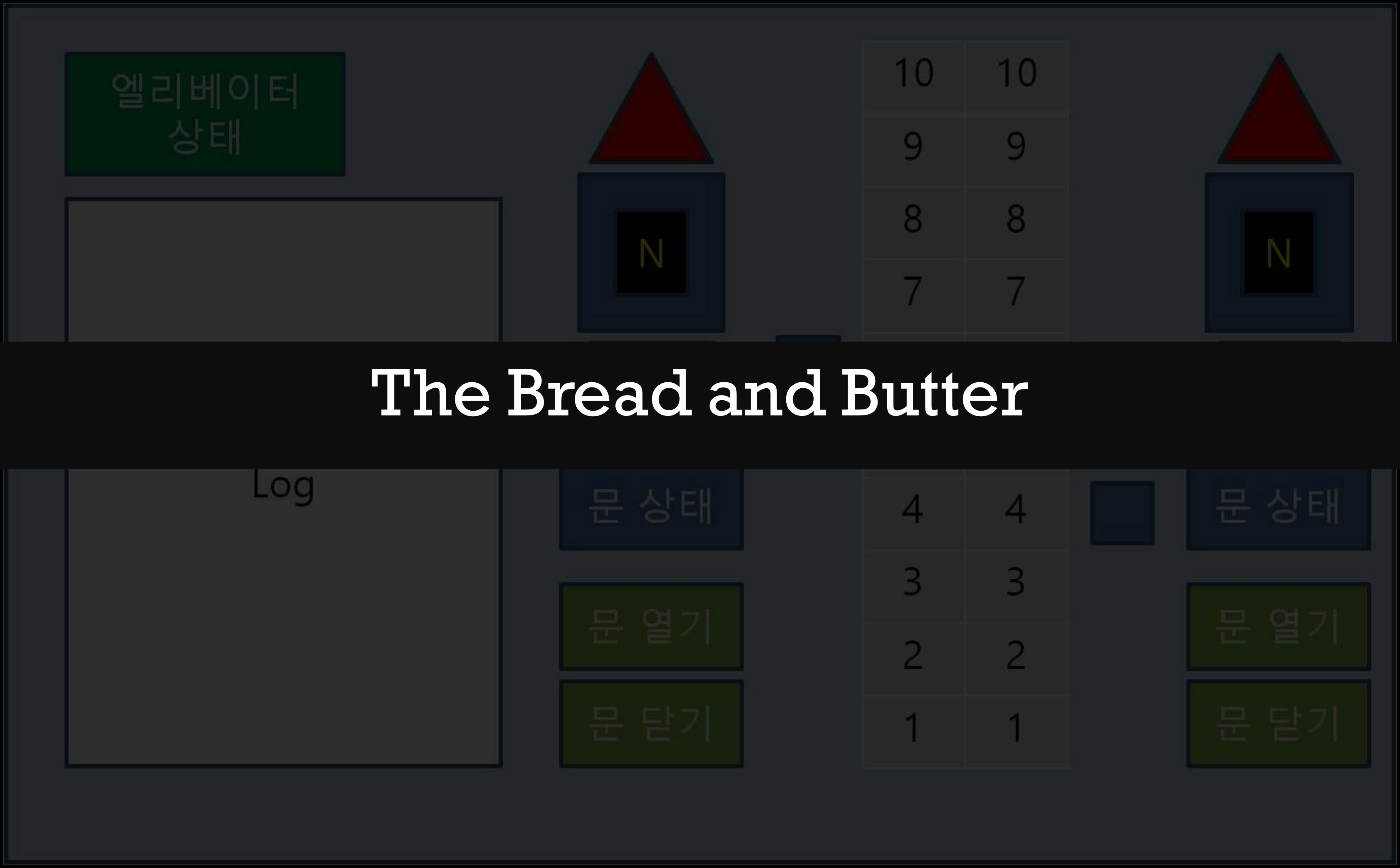
1.The Big Picture Remains the Same.

2. Details, Details, and More Details.

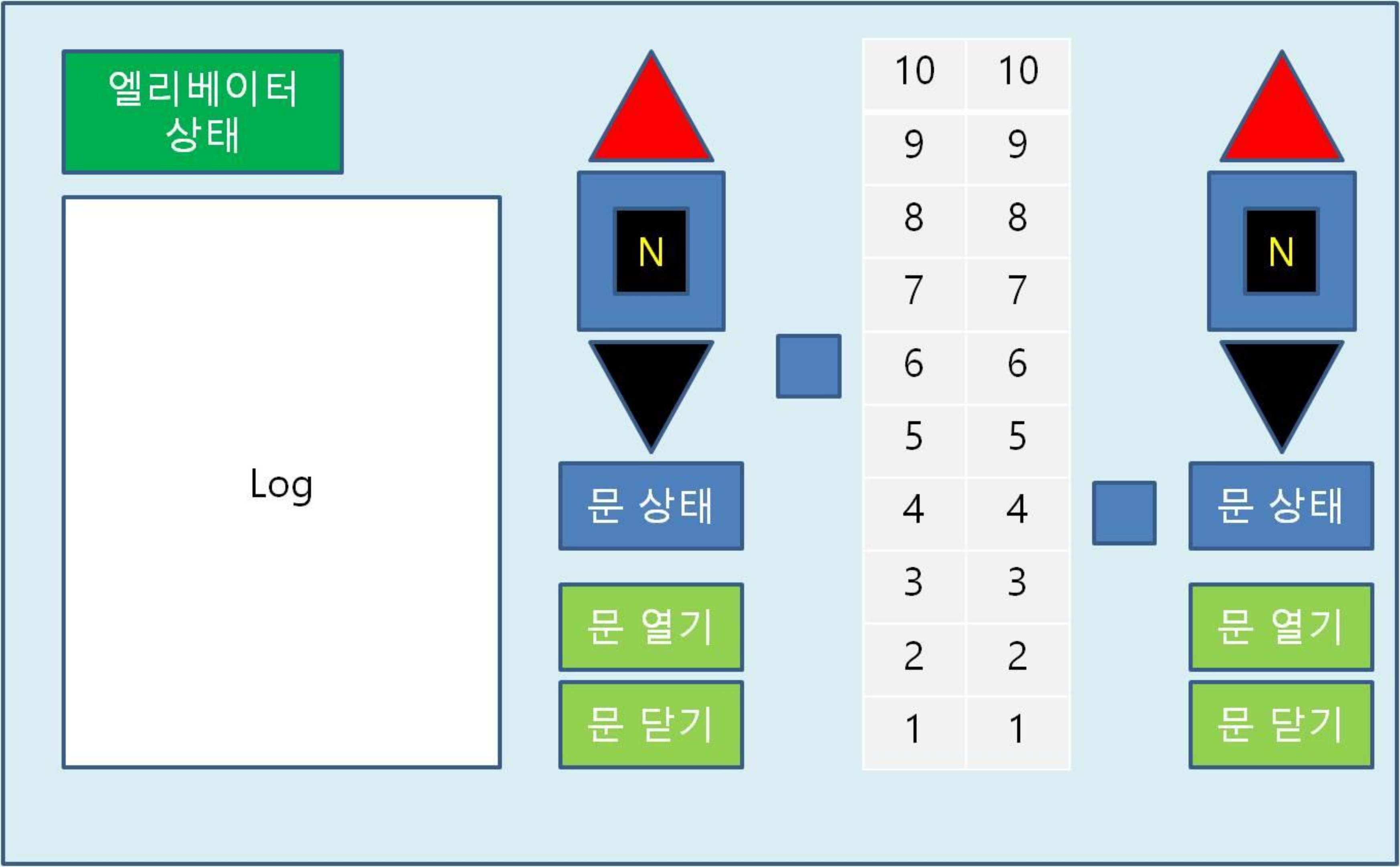
3. Actual Flow Elaboration

UI Design

The Bread and Butter



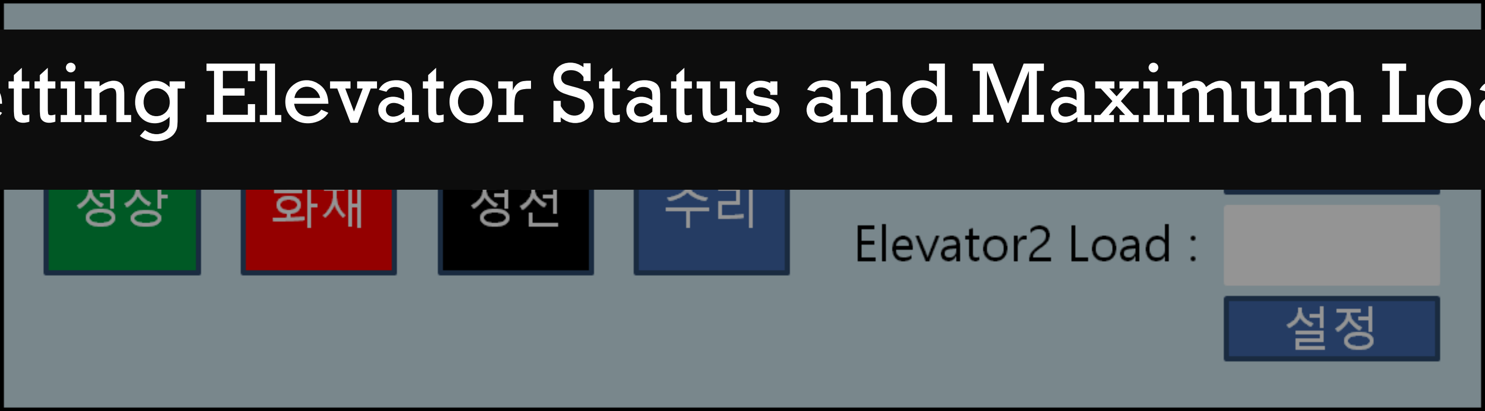
UI Design : Main View



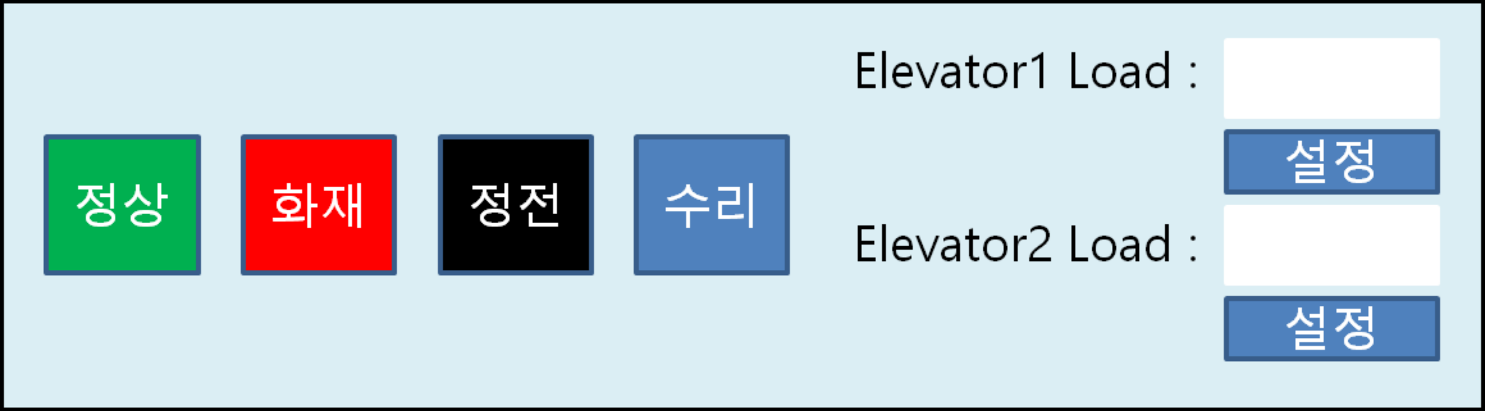
10	10
9	9
8	8
7	7
6	6
5	5
4	4
3	3
2	2
1	1

UI Design : Main View

Setting Elevator Status and Maximum Load

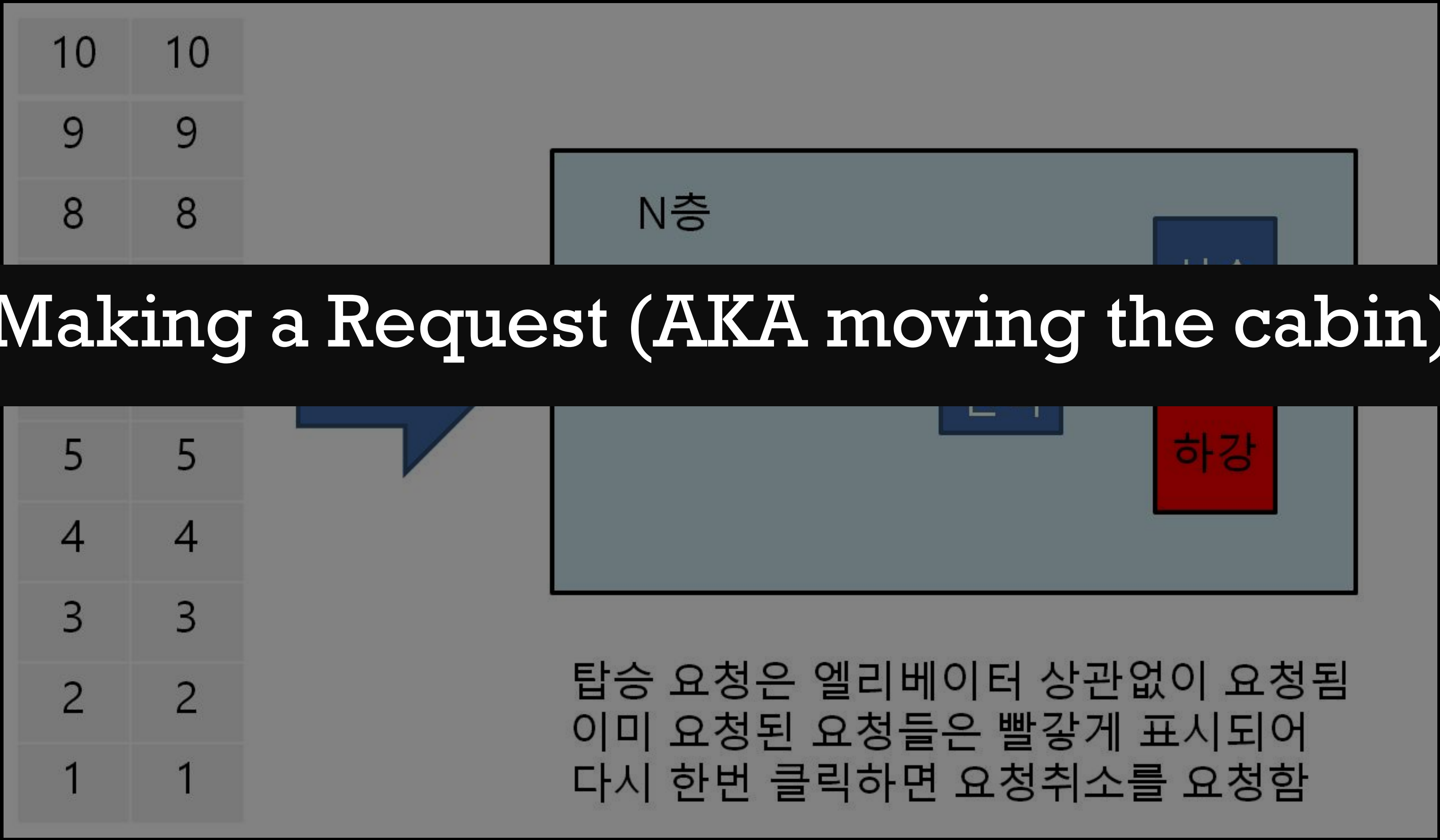


UI Design : State & MaxLoad Configuration



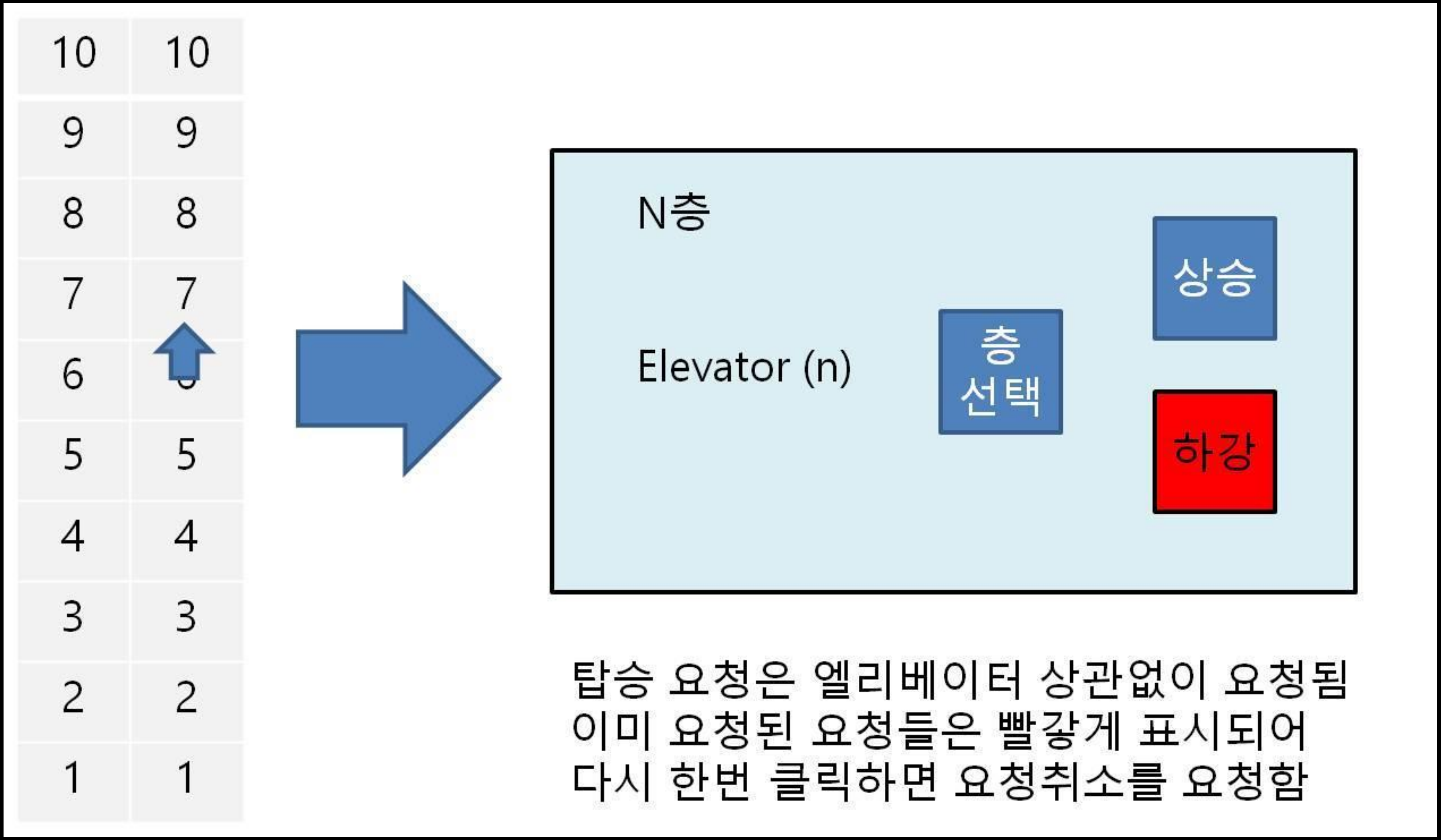
UI Design : State & MaxLoad Configuration

Making a Request (AKA moving the cabin)



탑승 요청은 엘리베이터 상관없이 요청됨
이미 요청된 요청들은 빨갛게 표시되어
다시 한번 클릭하면 요청취소를 요청함

UI Design : State & MaxLoad Configuration



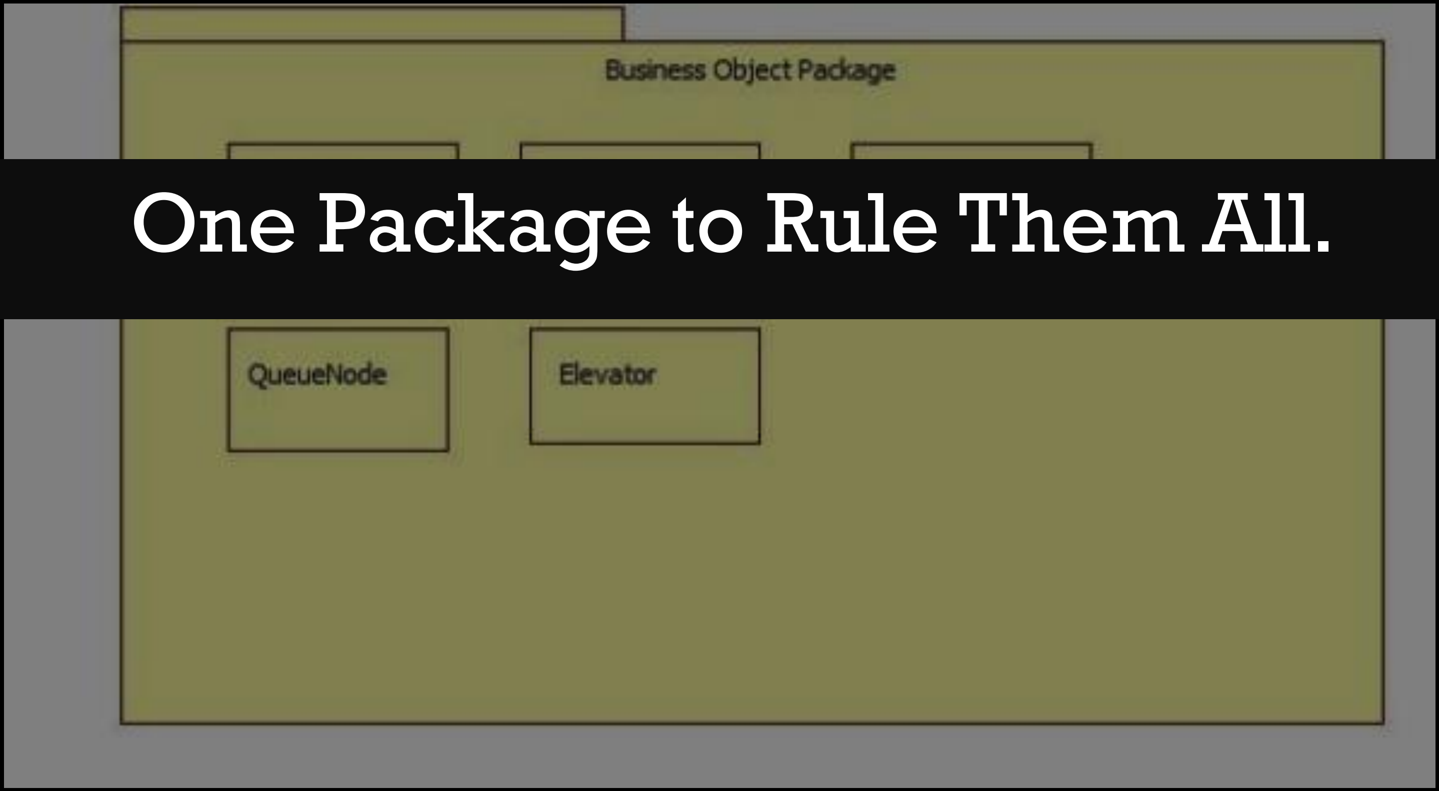
10	10
9	9
8	8
7	7
6	6
5	5
4	4
3	3
2	2
1	1

탑승 요청은 엘리베이터 상관없이 요청됨
이미 요청된 요청들은 빨갛게 표시되어
다시 한번 클릭하면 요청취소를 요청함

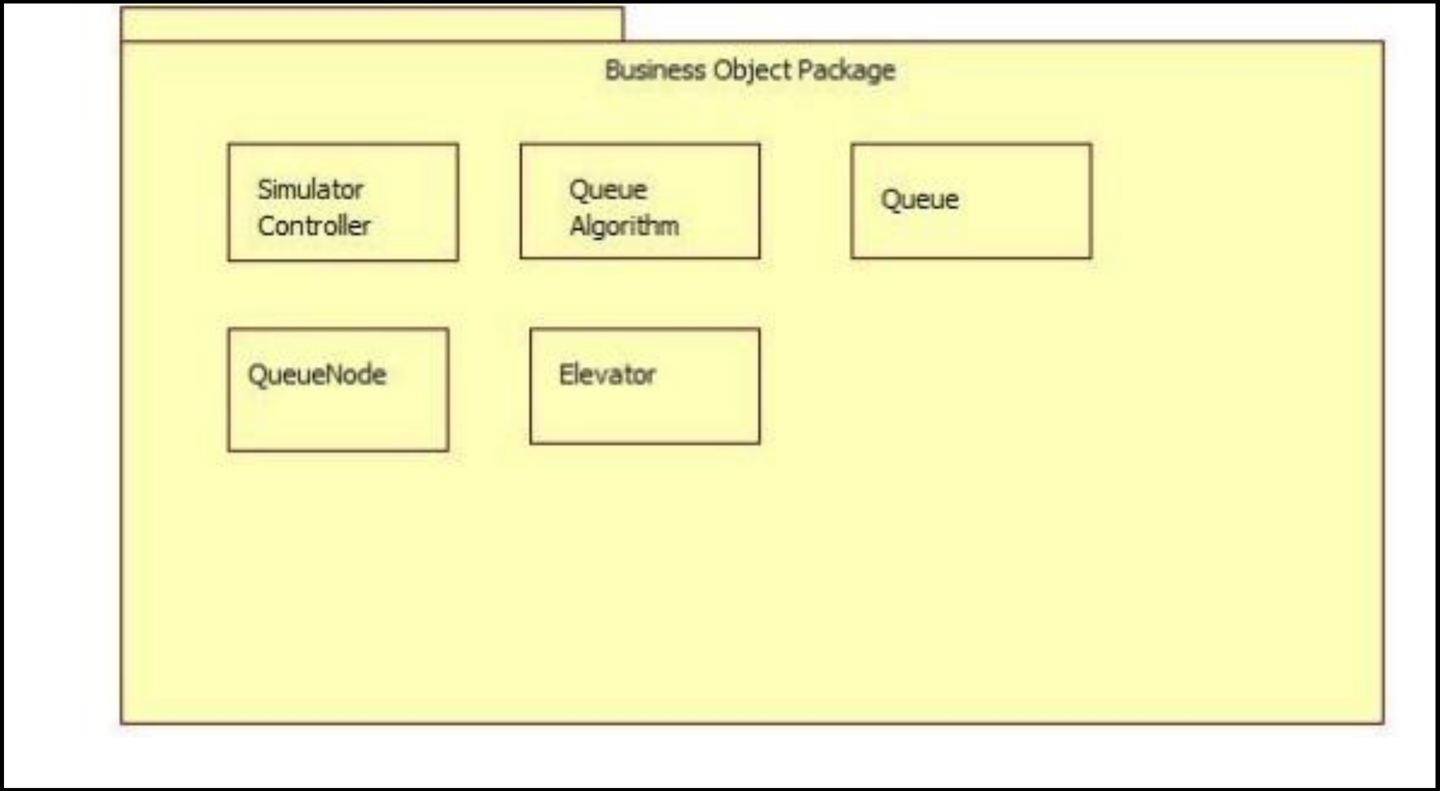
UI Design : State & MaxLoad Configuration

System Architecture

One Package to Rule Them All.



System Architecture



System Architecture

Sequence Diagrams

Disclaimer

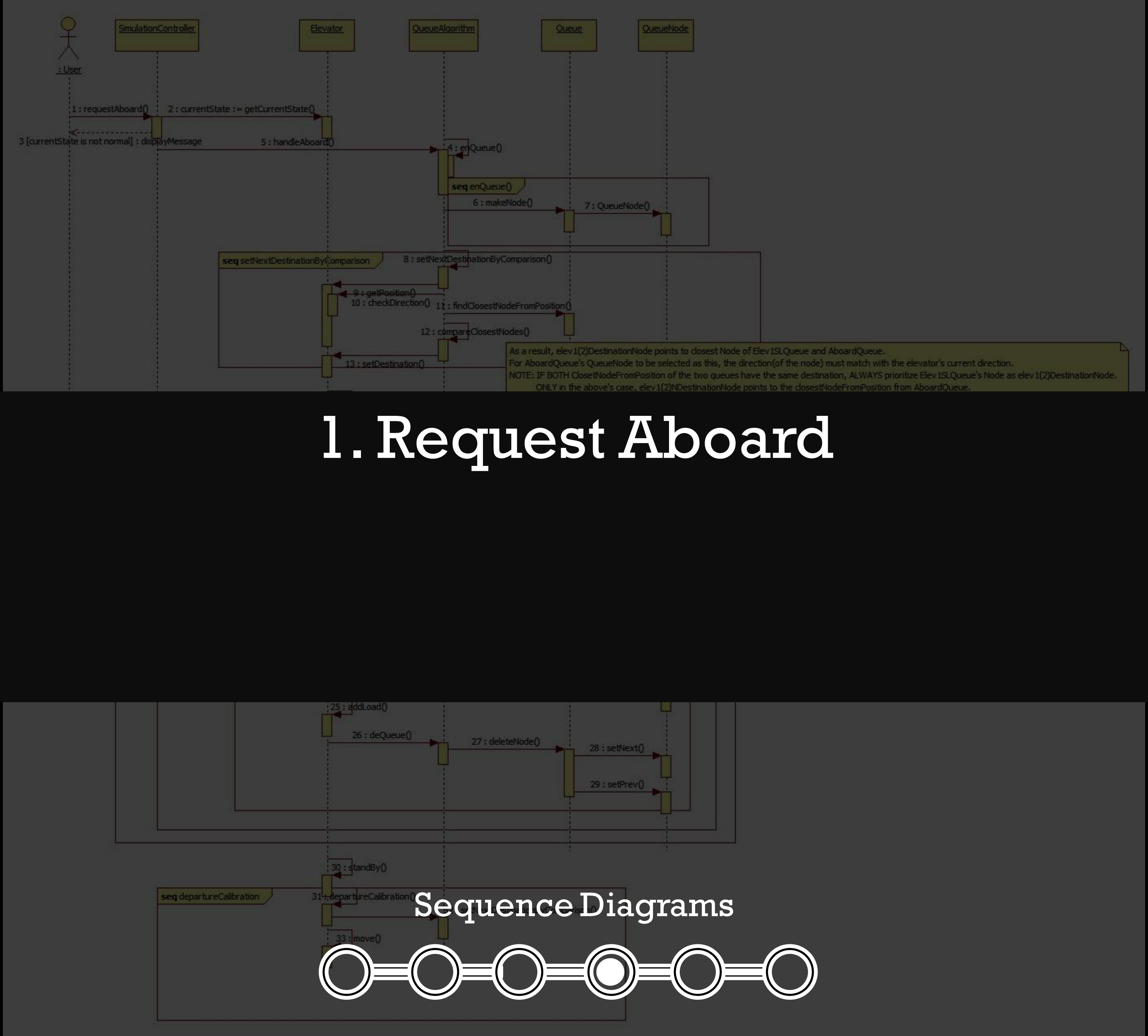
Since 6 out of 18 use cases are evident ones, only six interactions with the user were visualized.

Disclaimer

Because of this, we had to cram in all hidden operations into those diagrams.

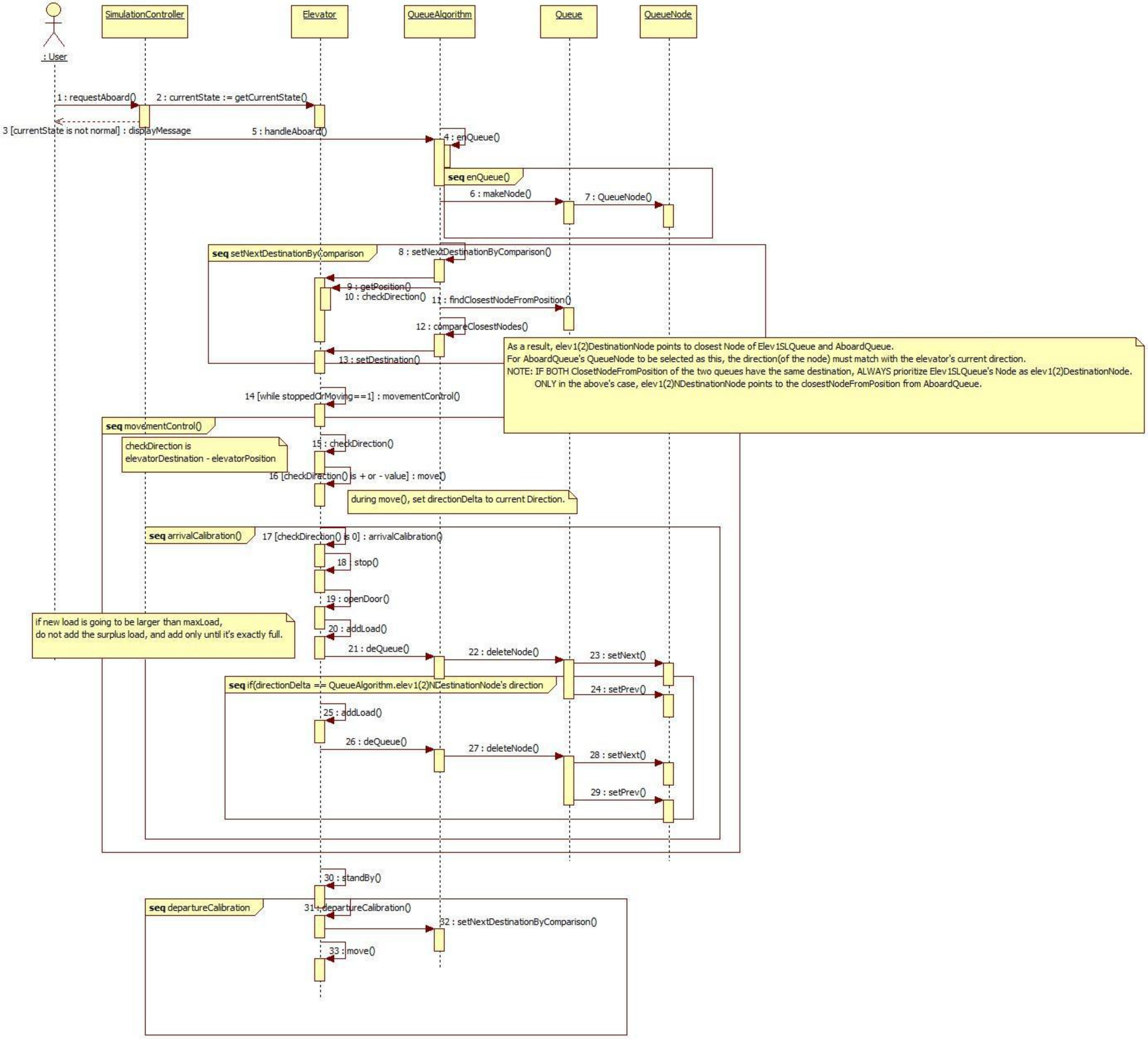
(i.e. massive diagrams)

1. Request Aboard



As a result, elev1(2)DestinationNode points to closest Node of Elev1SLQueue and AboardQueue.
For AboardQueue's QueueNode to be selected as this, the direction(of the node) must match with the elevator's current direction.
NOTE: IF BOTH ClosetNodeFromPosition of the two queues have the same destination, ALWAYS prioritize Elev1SLQueue's Node as elev1(2)DestinationNode.
ONLY in the above's case, elev1(2)NDestinationNode points to the closestNodeFromPosition from AboardQueue.

Sequence Diagrams

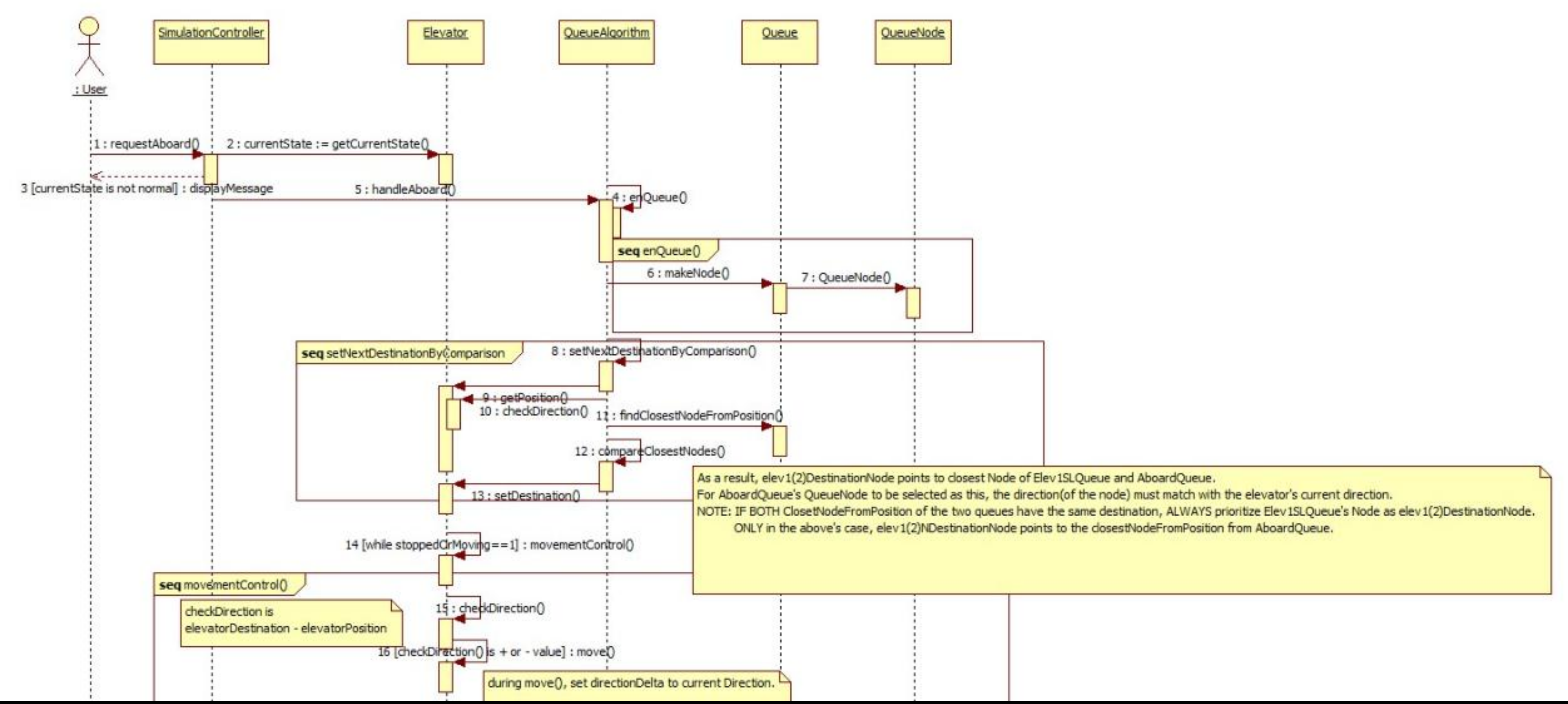

: User
SimulationController
Elevator
QueueAlgorithm
Queue
QueueNode
1 : requestAboard()
2 : currentState := getCurrentState()
3 [currentState is not normal] : displayMessage
5 : handleAboard()
4 : enQueue()
seq enQueue()
6 : makeNode()
7 : QueueNode()
seq setNextDestinationByComparison
8 : setNextDestinationByComparison()
9 : getPosition()
10 : checkDirection()
11 : findClosestNodeFromPosition()
12 : compareClosestNodes()
13 : setDestination()
14 [while stoppedOrMoving==1] : movementControl()
seq movementControl()
15 : checkDirection()
16 [checkDirection() is + or - value] : move()
seq arrivalCalibration()
17 [checkDirection() is 0] : arrivalCalibration()
18 stop()
19 : openDoor()
20 : addLoad()
21 : deQueue()
22 : deleteNode()
23 : setNext()
seq if(directionDelta == QueueAlgorithm.elev1(2)NDestinationNode's direction
24 : setPrev()
25 : addLoad()
26 : deQueue()
27 : deleteNode()
28 : setNext()
29 : setPrev()
30 : standBy()
seq departureCalibration
31 : departureCalibration()
32 : setNextDestinationByComparison()
33 : move()

As a result, elev1(2)DestinationNode points to closest Node of Elev1SLQueue and AboardQueue.
For AboardQueue's QueueNode to be selected as this, the direction(of the node) must match with the elevator's current direction.
NOTE: IF BOTH ClosetNodeFromPosition of the two queues have the same destination, ALWAYS prioritize Elev1SLQueue's Node as elev1(2)DestinationNode.
ONLY in the above's case, elev1(2)NDestinationNode points to the closestNodeFromPosition from AboardQueue.

checkDirection is
elevatorDestination - elevatorPosition

during move(), set directionDelta to current Direction.

if new load is going to be larger than maxLoad,
do not add the surplus load, and add only until it's exactly full.

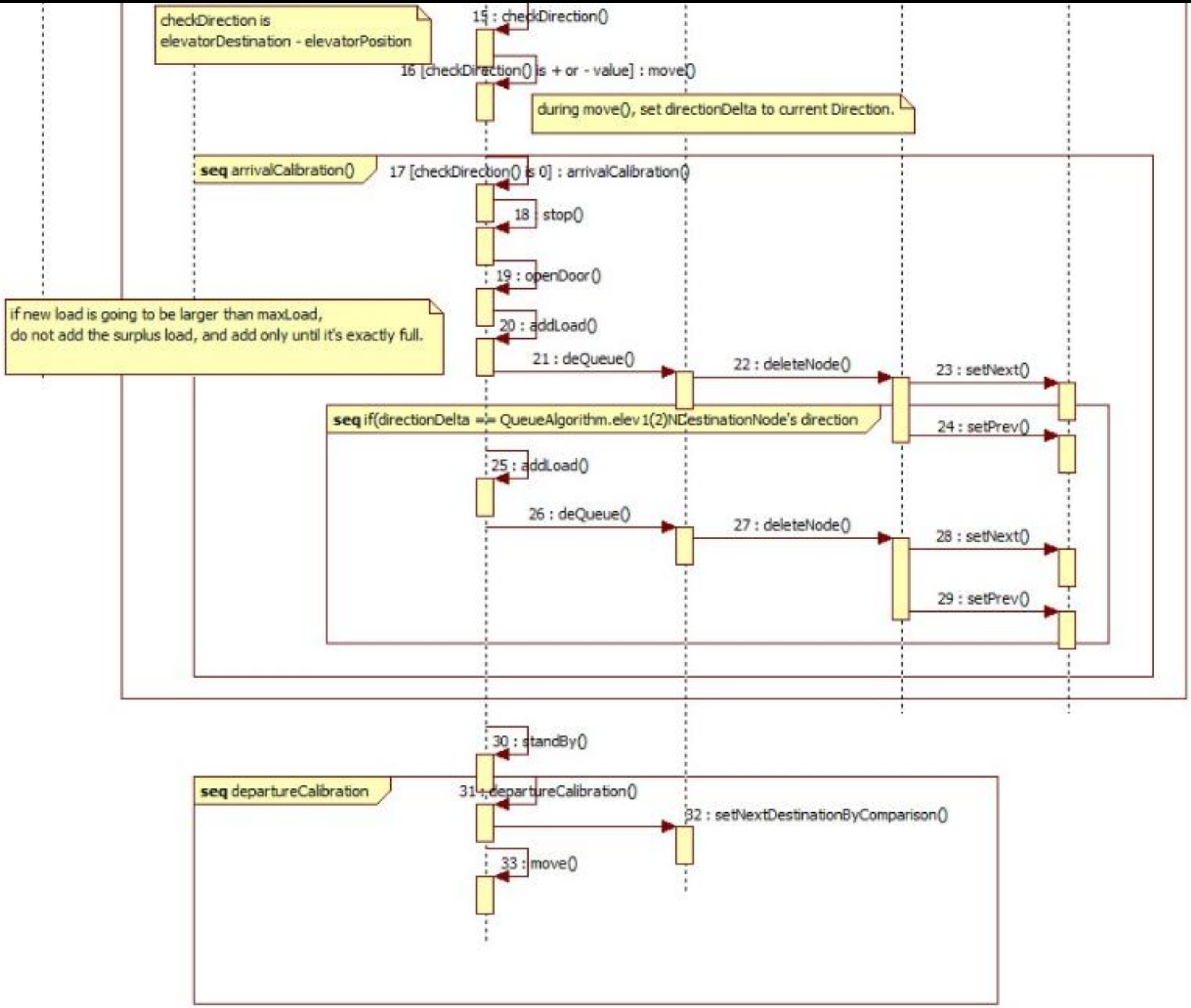

: User
SimulationController
Elevator
QueueAlgorithm
Queue
QueueNode
1 : requestAboard()
2 : currentState := getCurrentState()
3 [currentState is not normal] : displayMessage
5 : handleAboard()
4 : enQueue()
seq enQueue()
6 : makeNode()
7 : QueueNode()
seq setNextDestinationByComparison
8 : setNextDestinationByComparison()
9 : getPosition()
10 : checkDirection()
11 : findClosestNodeFromPosition()
12 : compareClosestNodes()
13 : setDestination()
14 [while stoppedOrMoving==1] : movementControl()
seq movementControl()
15 : checkDirection()
16 [checkDirection() is + or - value] : move()

As a result, elev1(2)DestinationNode points to closest Node of Elev1SLQueue and AboardQueue.
For AboardQueue's QueueNode to be selected as this, the direction(of the node) must match with the elevator's current direction.
NOTE: IF BOTH ClosetNodeFromPosition of the two queues have the same destination, ALWAYS prioritize Elev1SLQueue's Node as elev1(2)DestinationNode.
ONLY in the above's case, elev1(2)NDestinationNode points to the closestNodeFromPosition from AboardQueue.

checkDirection is
elevatorDestination - elevatorPosition

during move(), set directionDelta to current Direction.

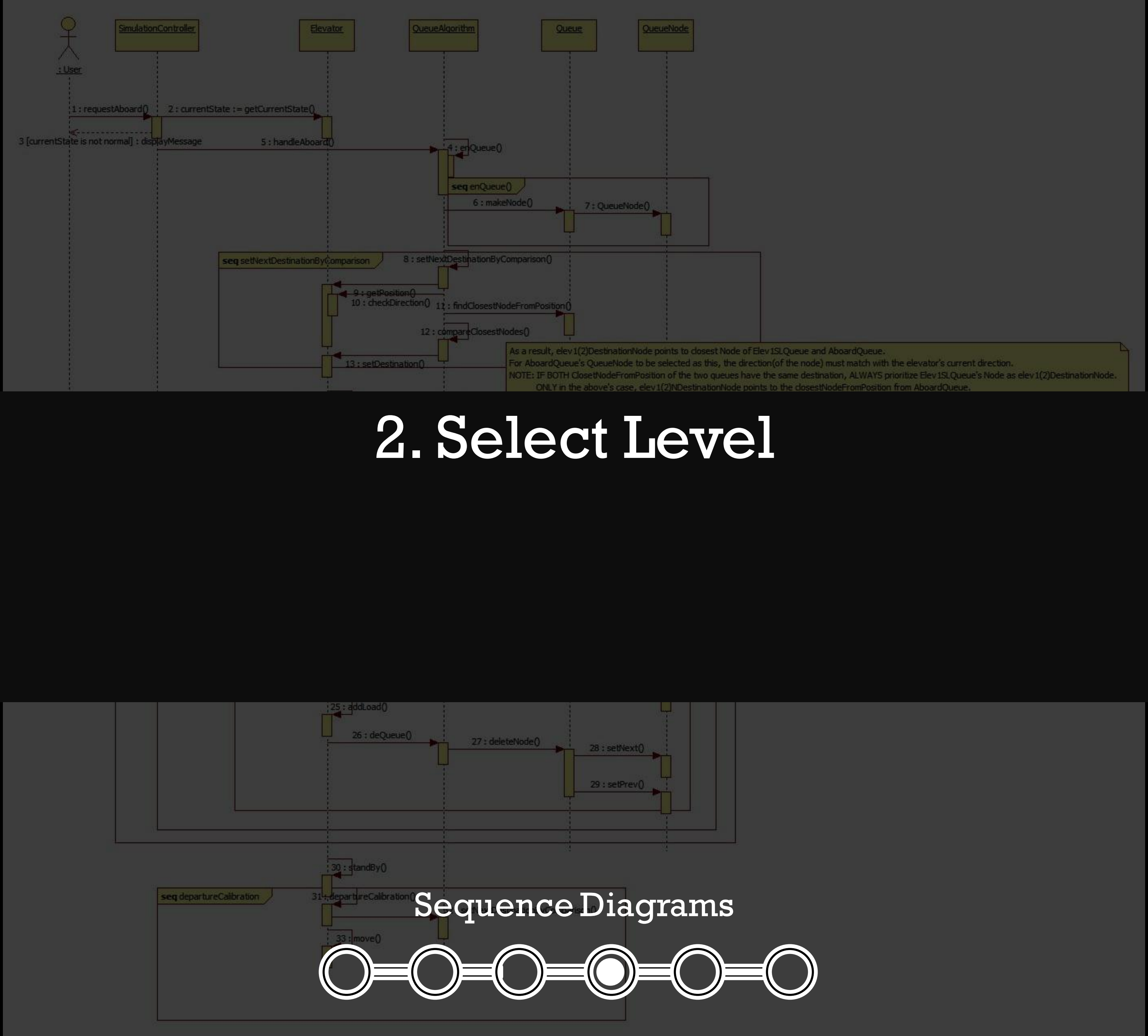
Sequence Diagrams



checkDirection is elevatorDestination - elevatorPosition

during move(), set directionDelta to current Direction.

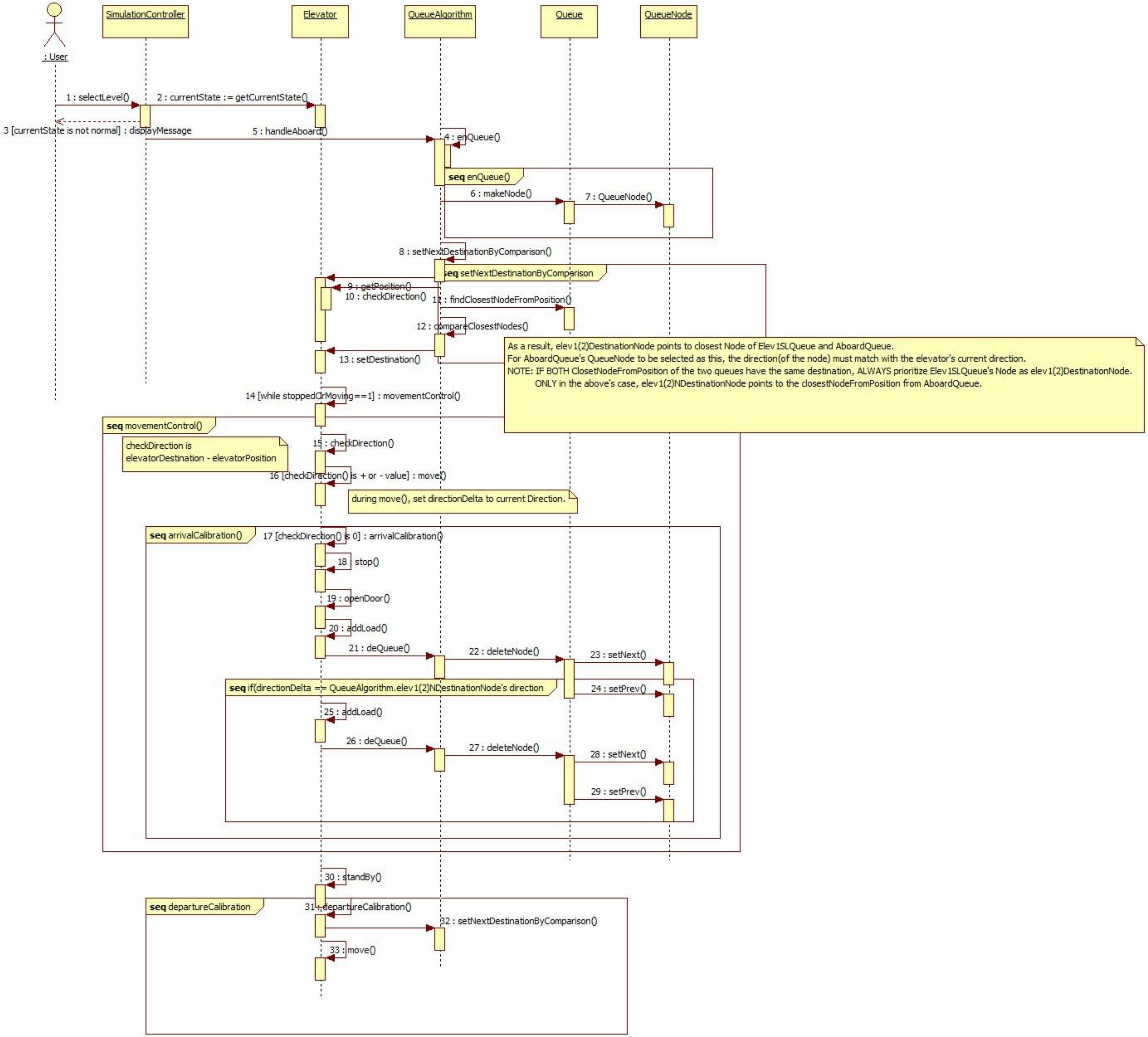
if new load is going to be larger than maxLoad,
do not add the surplus load, and add only until it's exactly full.



As a result, elev1(2)DestinationNode points to closest Node of Elev1SLQueue and AboardQueue.
For AboardQueue's QueueNode to be selected as this, the direction(of the node) must match with the elevator's current direction.
NOTE: IF BOTH ClosetNodeFromPosition of the two queues have the same destination, ALWAYS prioritize Elev1SLQueue's Node as elev1(2)DestinationNode.
ONLY in the above's case, elev1(2)NDestinationNode points to the closestNodeFromPosition from AboardQueue.

2. Select Level

Sequence Diagrams

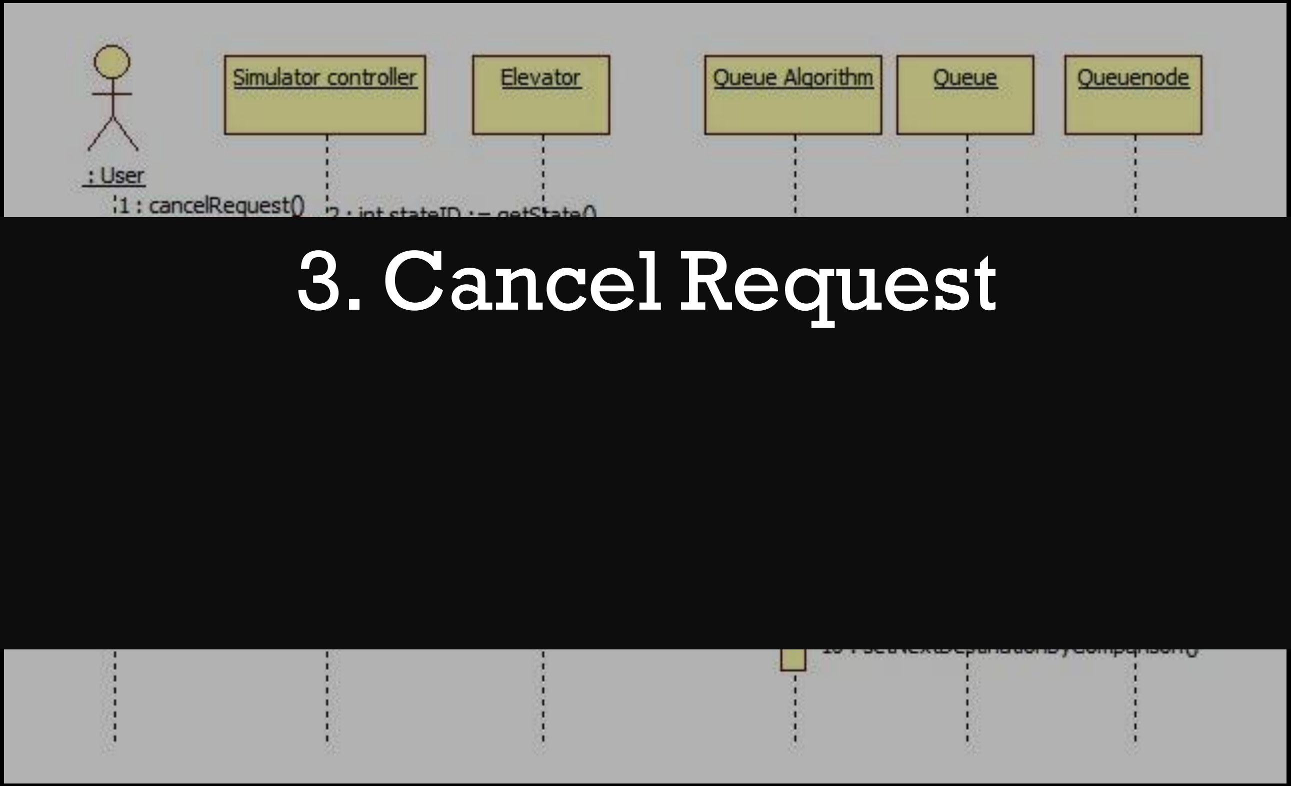

: User
SimulationController
Elevator
QueueAlgorithm
Queue
QueueNode
1 : selectLevel()
2 : currentState := getCurrentState()
3 [currentState is not normal] : displayMessage
5 : handleAboard()
4 : enQueue()
seq enQueue()
6 : makeNode()
7 : QueueNode()
8 : setNextDestinationByComparison()
seq setNextDestinationByComparison
9 : getPosition()
10 : checkDirection()
11 : findClosestNodeFromPosition()
12 : compareClosestNodes()
13 : setDestination()
14 [while stoppedOrMoving==1] : movementControl()
seq movementControl()
15 : checkDirection()
16 [checkDirection() is + or - value] : move()
seq arrivalCalibration()
17 [checkDirection() is 0] : arrivalCalibration()
18 : stop()
19 : openDoor()
20 : addLoad()
21 : deQueue()
22 : deleteNode()
23 : setNext()
seq if(directionDelta == QueueAlgorithm.elev1(2)NDestinationNode's direction
24 : setPrev()
25 : addLoad()
26 : deQueue()
27 : deleteNode()
28 : setNext()
29 : setPrev()
30 : standBy()
seq departureCalibration
31 : departureCalibration()
32 : setNextDestinationByComparison()
33 : move()

As a result, elev1(2)DestinationNode points to closest Node of Elev1SLQueue and AboardQueue.
For AboardQueue's QueueNode to be selected as this, the direction(of the node) must match with the elevator's current direction.
NOTE: IF BOTH ClosetNodeFromPosition of the two queues have the same destination, ALWAYS prioritize Elev1SLQueue's Node as elev1(2)DestinationNode.
ONLY in the above's case, elev1(2)NDestinationNode points to the closestNodeFromPosition from AboardQueue.

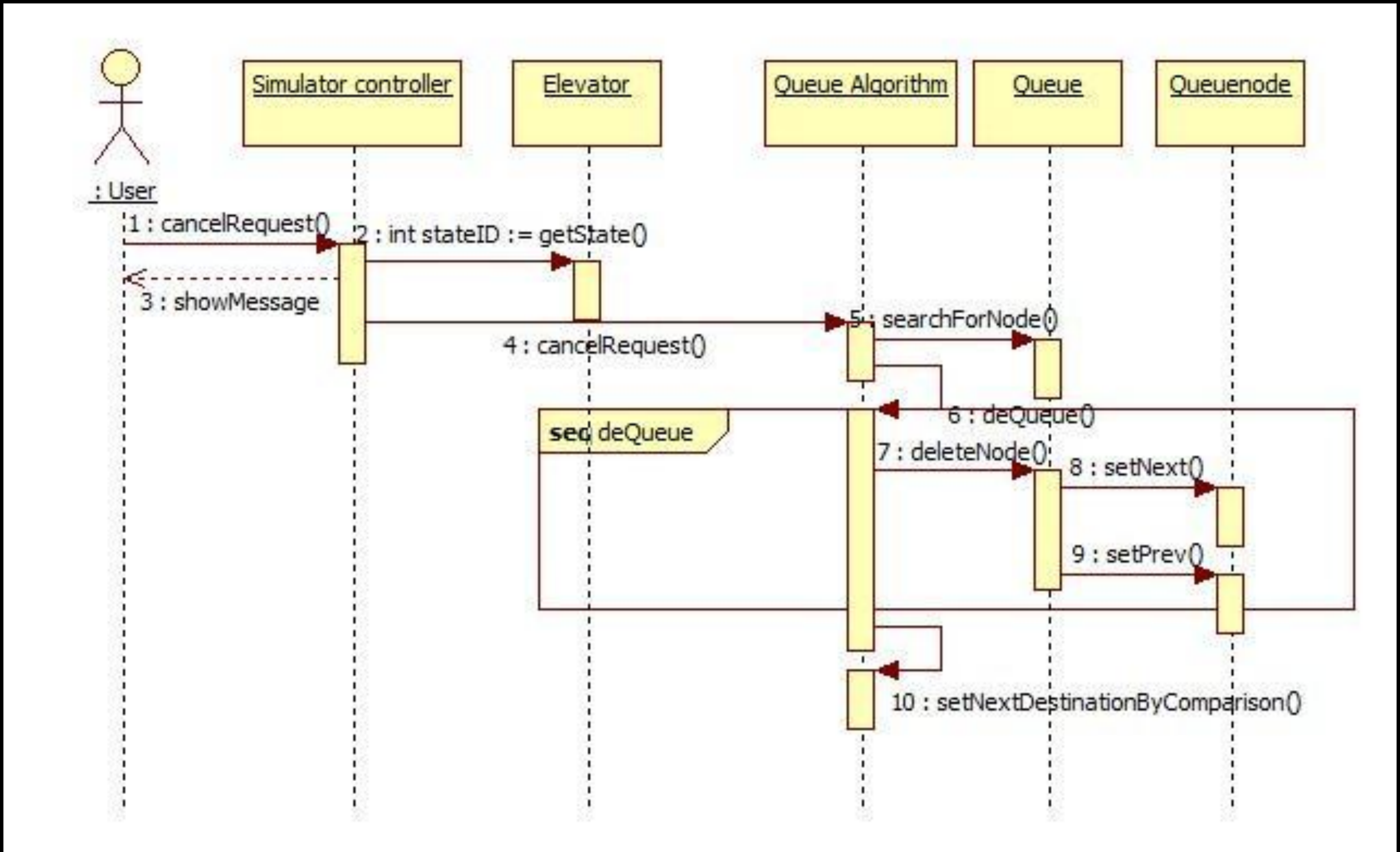
checkDirection is
elevatorDestination - elevatorPosition

during move(), set directionDelta to current Direction.

3. Cancel Request

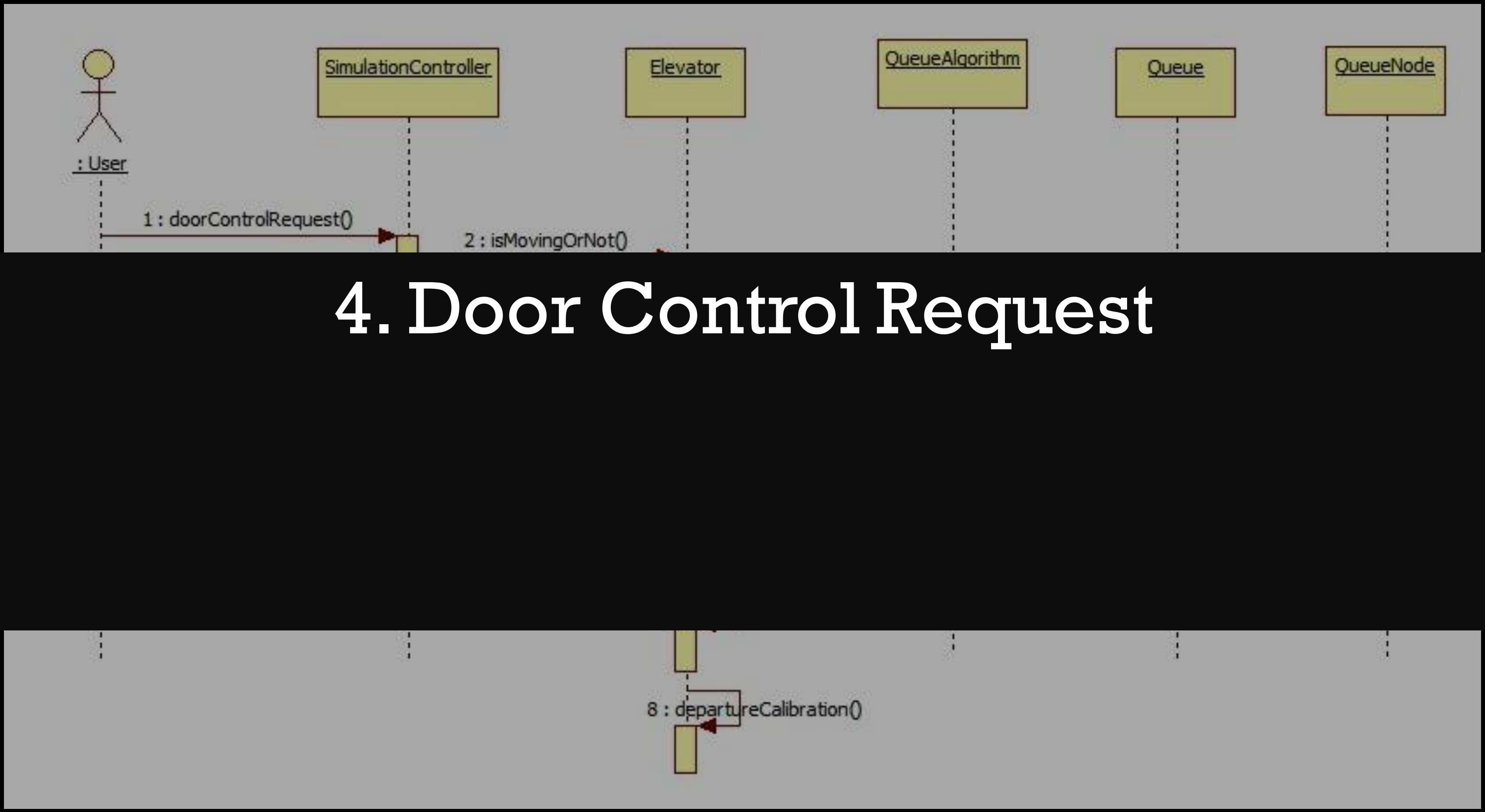


Sequence Diagrams

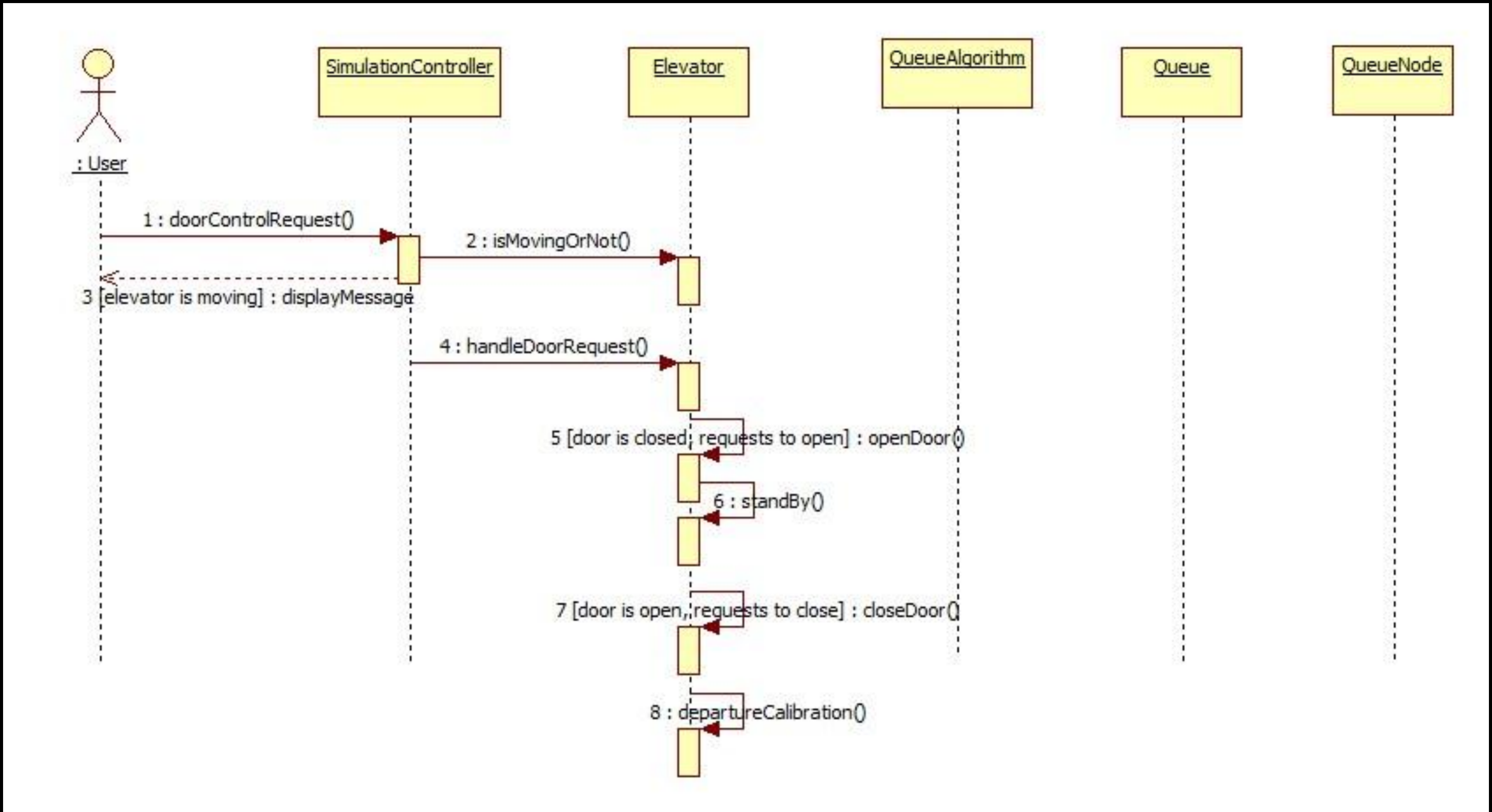


Sequence Diagrams

4. Door Control Request

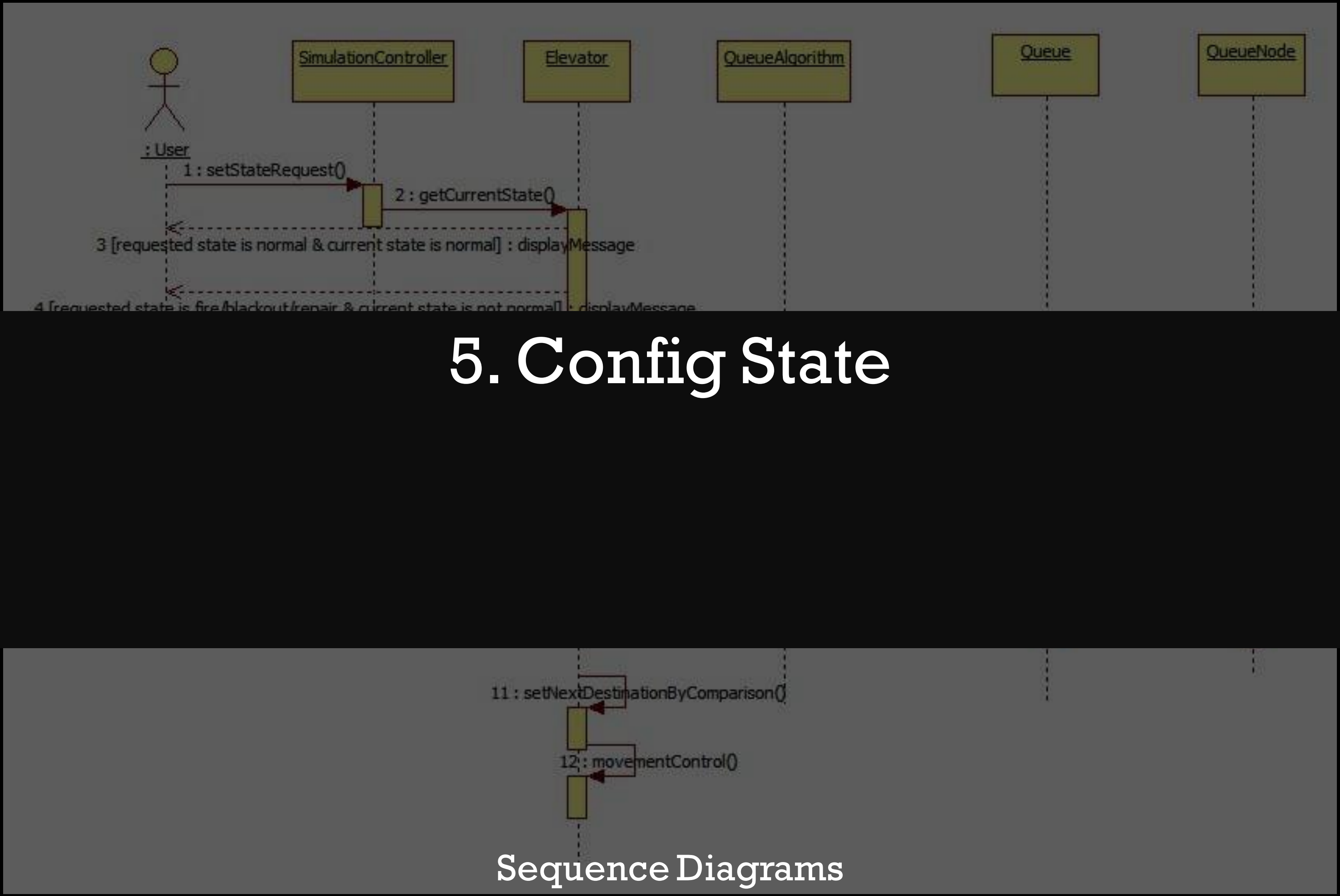


Sequence Diagrams

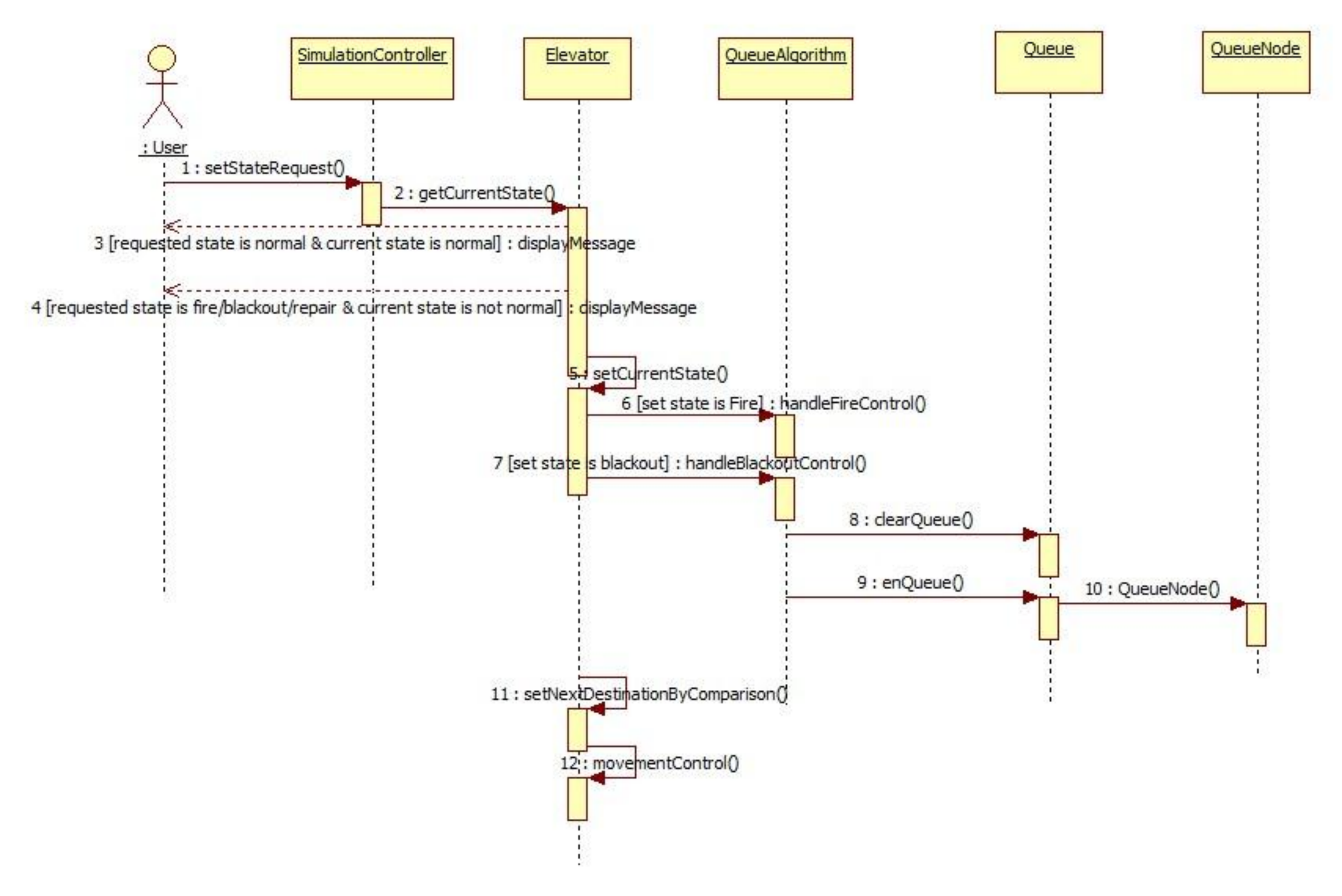

: User
SimulationController
Elevator
QueueAlgorithm
Queue
QueueNode
1 : doorControlRequest()
2 : isMovingOrNot()
3 [elevator is moving] : displayMessage
4 : handleDoorRequest()
5 [door is closed, requests to open] : openDoor()
6 : standBy()
7 [door is open, requests to close] : closeDoor()
8 : departureCalibration()

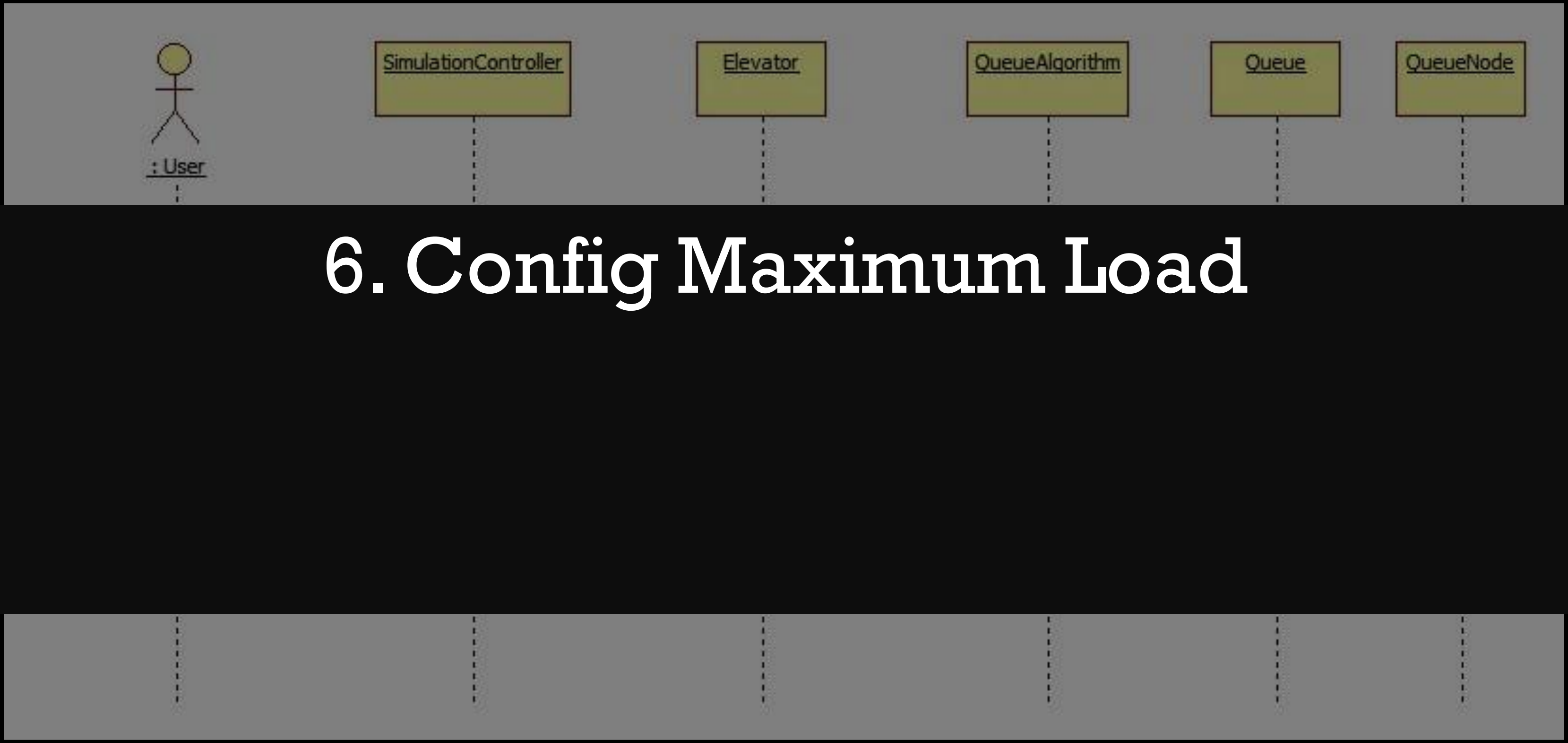
Sequence Diagrams

5. Config State


: User
SimulationController
Elevator
QueueAlgorithm
Queue
QueueNode
1 : setStateRequest()
2 : getCurrentState()
3 [requested state is normal & current state is normal] : displayMessage
4 [requested state is fire/blackout/repair & current state is not normal] : displayMessage
11 : setNextDestinationByComparison()
12 : movementControl()

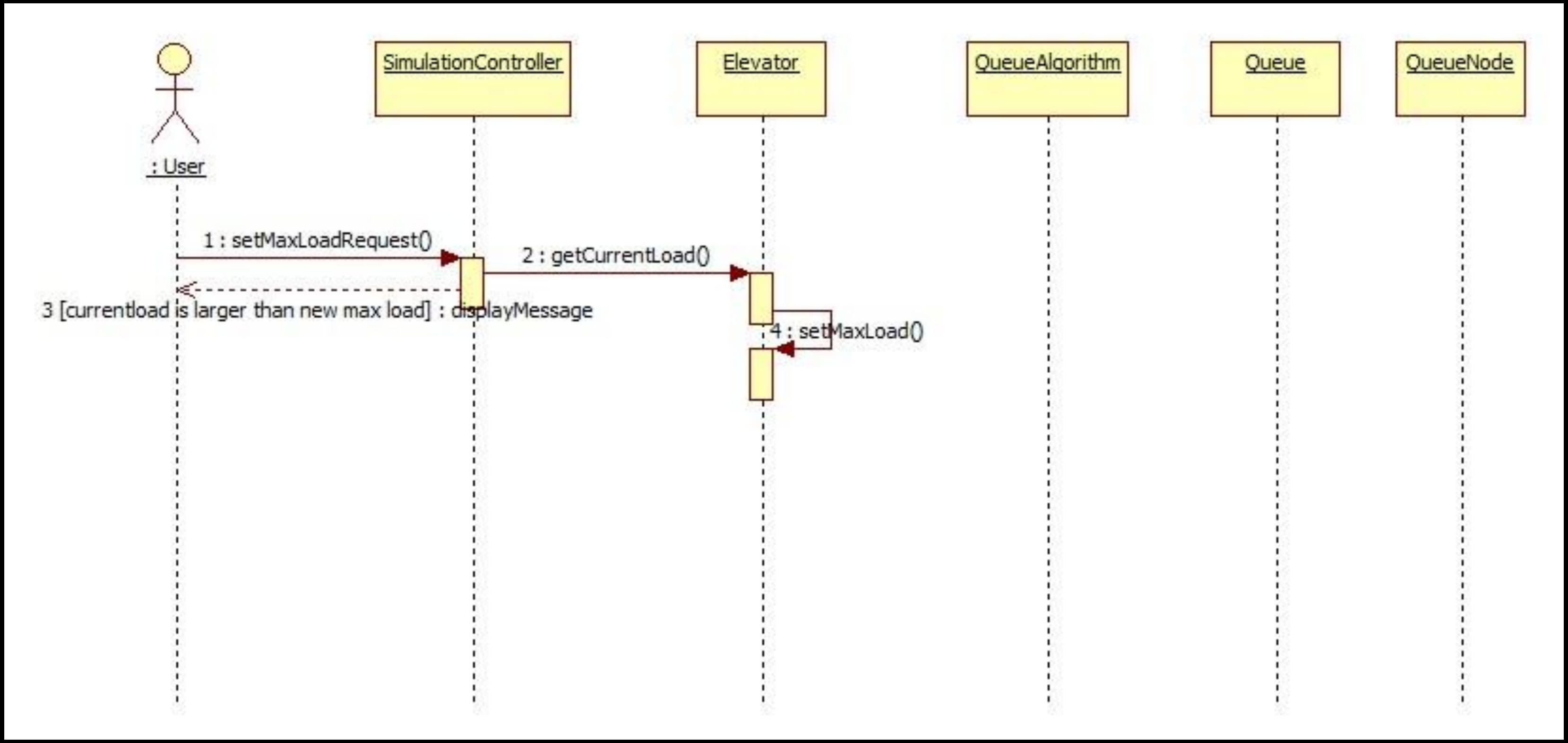
Sequence Diagrams


: User
SimulationController
Elevator
QueueAlgorithm
Queue
QueueNode
1 : setStateRequest()
2 : getCurrentState()
3 [requested state is normal & current state is normal] : displayMessage
4 [requested state is fire/blackout/repair & current state is not normal] : displayMessage
5 : setCurrentState()
6 [set state is Fire] : handleFireControl()
7 [set state is blackout] : handleBlackoutControl()
8 : clearQueue()
9 : enQueue()
10 : QueueNode()
11 : setNextDestinationByComparison()
12 : movementControl()



6. Config Maximum Load

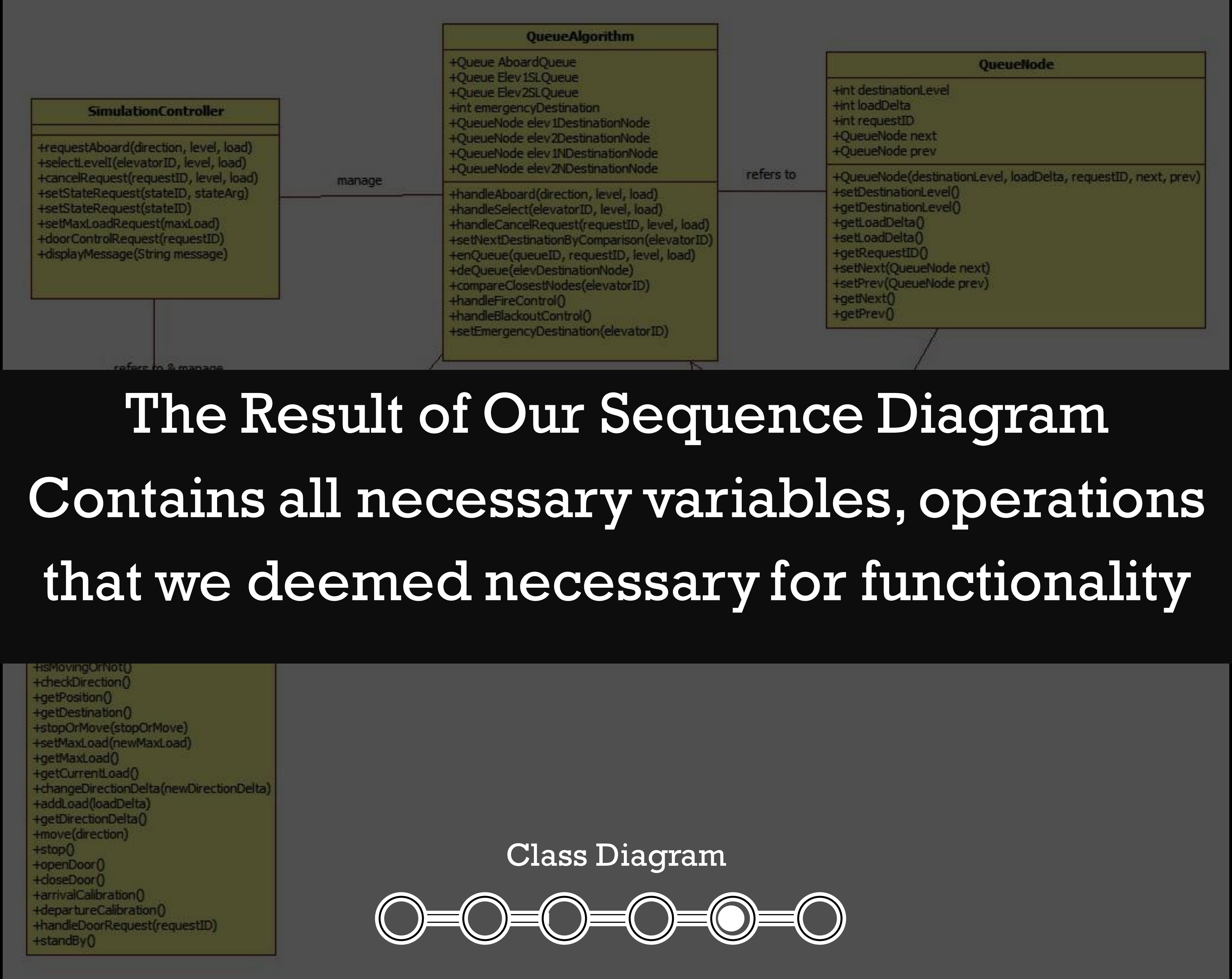
Sequence Diagrams



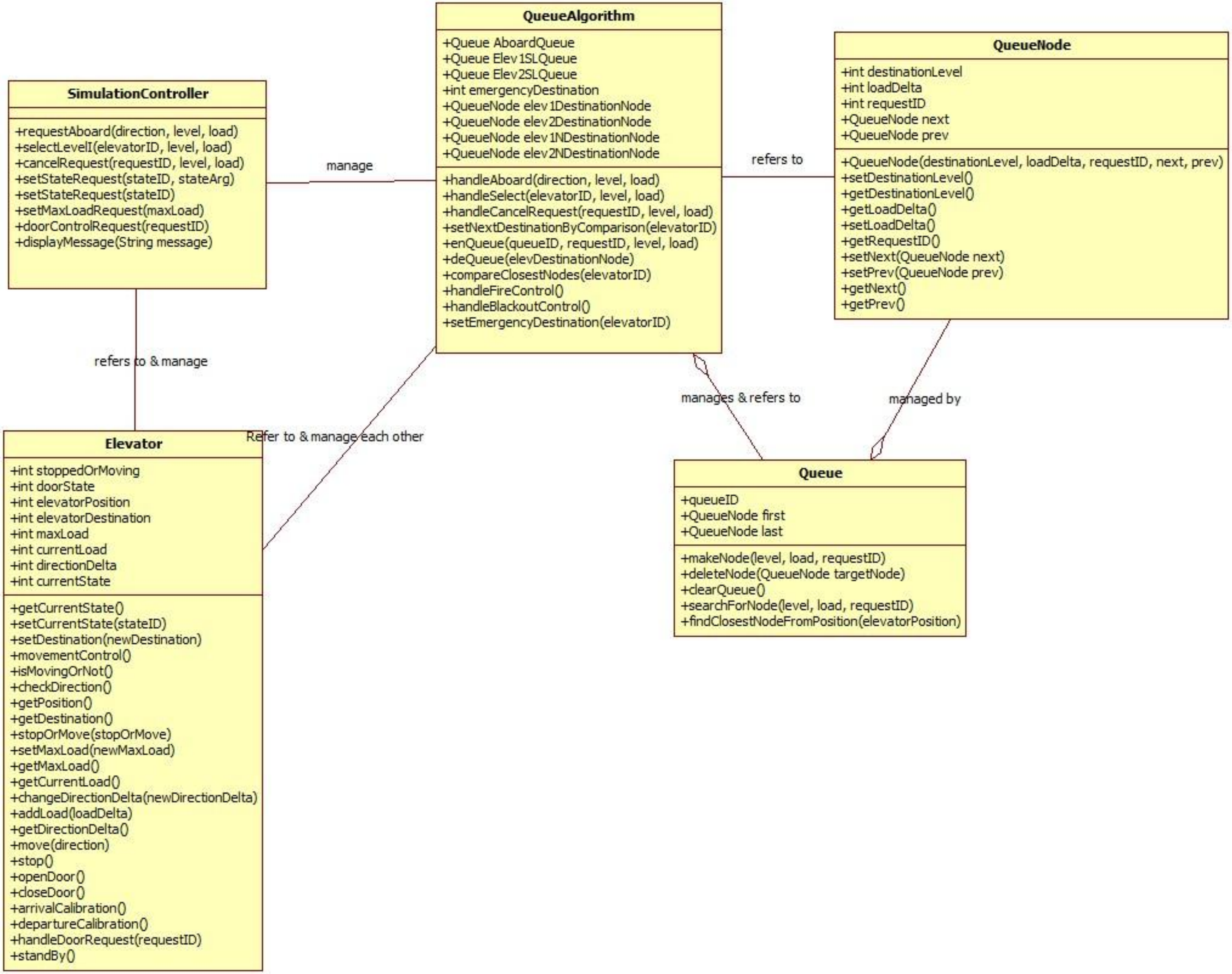
Sequence Diagrams

Class Diagram

The Result of Our Sequence Diagram Contains all necessary variables, operations that we deemed necessary for functionality


QueueAlgorithm
+Queue AboardQueue
+Queue Elev1SLQueue
+Queue Elev2SLQueue
+int emergencyDestination
+QueueNode elev1DestinationNode
+QueueNode elev2DestinationNode
+QueueNode elev1NDestinationNode
+QueueNode elev2NDestinationNode
+handleAboard(direction, level, load)
+handleSelect(elevatorID, level, load)
+handleCancelRequest(requestID, level, load)
+setNextDestinationByComparison(elevatorID)
+enQueue(queueID, requestID, level, load)
+deQueue(elevDestinationNode)
+compareClosestNodes(elevatorID)
+handleFireControl()
+handleBlackoutControl()
+setEmergencyDestination(elevatorID)
SimulationController
+requestAboard(direction, level, load)
+selectLevelI(elevatorID, level, load)
+cancelRequest(requestID, level, load)
+setStateRequest(stateID, stateArg)
+setStateRequest(stateID)
+setMaxLoadRequest(maxLoad)
+doorControlRequest(requestID)
+displayMessage(String message)
manage
refers to
QueueNode
+int destinationLevel
+int loadDelta
+int requestID
+QueueNode next
+QueueNode prev
+QueueNode(destinationLevel, loadDelta, requestID, next, prev)
+setDestinationLevel()
+getDestinationLevel()
+getLoadDelta()
+setLoadDelta()
+getRequestID()
+setNext(QueueNode next)
+setPrev(QueueNode prev)
+getNext()
+getPrev()
refers to & manage
+isMovingOrNot()
+checkDirection()
+getPosition()
+getDestination()
+stopOrMove(stopOrMove)
+setMaxLoad(newMaxLoad)
+getMaxLoad()
+getCurrentLoad()
+changeDirectionDelta(newDirectionDelta)
+addLoad(loadDelta)
+getDirectionDelta()
+move(direction)
+stop()
+openDoor()
+closeDoor()
+arrivalCalibration()
+departureCalibration()
+handleDoorRequest(requestID)
+standBy()

Class Diagram


QueueAlgorithm
+Queue AboardQueue
+Queue Elev 1SLQueue
+Queue Elev2SLQueue
+int emergencyDestination
+QueueNode elev 1DestinationNode
+QueueNode elev2DestinationNode
+QueueNode elev 1NDestinationNode
+QueueNode elev2NDestinationNode
+handleAboard(direction, level, load)
+handleSelect(elevatorID, level, load)
+handleCancelRequest(requestID, level, load)
+setNextDestinationByComparison(elevatorID)
+enQueue(queueID, requestID, level, load)
+deQueue(elevDestinationNode)
+compareClosestNodes(elevatorID)
+handleFireControl()
+handleBlackoutControl()
+setEmergencyDestination(elevatorID)
QueueNode
+int destinationLevel
+int loadDelta
+int requestID
+QueueNode next
+QueueNode prev
+QueueNode(destinationLevel, loadDelta, requestID, next, prev)
+setDestinationLevel()
+getDestinationLevel()
+getLoadDelta()
+setLoadDelta()
+getRequestID()
+setNext(QueueNode next)
+setPrev(QueueNode prev)
+getNext()
+getPrev()
SimulationController
+requestAboard(direction, level, load)
+selectLevelI(elevatorID, level, load)
+cancelRequest(requestID, level, load)
+setStateRequest(stateID, stateArg)
+setStateRequest(stateID)
+setMaxLoadRequest(maxLoad)
+doorControlRequest(requestID)
+displayMessage(String message)
manage
refers to
refers to & manage
Refer to & manage each other
manages & refers to
managed by
Elevator
+int stoppedOrMoving
+int doorState
+int elevatorPosition
+int elevatorDestination
+int maxLoad
+int currentLoad
+int directionDelta
+int currentState
+getCurrentState()
+setCurrentState(stateID)
+setDestination(newDestination)
+movementControl()
+isMovingOrNot()
+checkDirection()
+getPosition()
+getDestination()
+stopOrMove(stopOrMove)
+setMaxLoad(newMaxLoad)
+getMaxLoad()
+getCurrentLoad()
+changeDirectionDelta(newDirectionDelta)
+addLoad(loadDelta)
+getDirectionDelta()
+move(direction)
+stop()
+openDoor()
+closeDoor()
+arrivalCalibration()
+departureCalibration()
+handleDoorRequest(requestID)
+standBy()
Queue
+queueID
+QueueNode first
+QueueNode last
+makeNode(level, load, requestID)
+deleteNode(QueueNode targetNode)
+clearQueue()
+searchForNode(level, load, requestID)
+findClosestNodeFromPosition(elevatorPosition)

Ending Remarks

Sequences Were Made, Class Diagrams Were Had.

Possible Risks

Over-simplification of operations

Unintentional bundling of different functions

Encapsulation gone wrong

Irreversible design flaw

To-Dos

Actual Code Implementation

Assess for Any Unaccounted Necessary Operations

Possible Document Revisions

That is all.