Software Specification Report

Panagiotis Panagiotou (1815423), Sjoerd van de Goor (1557815)

February, March 2022

1 Introduction

For this assignment we were asked to design a Flight of Locks System (FLS) using the (non-formal) requirements specifications and UML models. A lock is a device used for raising and lowering boats, ships and other watercraft between stretches of water of different levels on river and canal waterways. A flight of locks is simply a series of locks in close-enough proximity to be identified as a single group. Our goal is to design a system in order for boats to be able to sail through a flight of locks with different water heights and an undefined number of locks.

2 Contents

2.1 Tasks

1. Requirements
2. Use Cases & Diagrams
3. Class Diagram
4. State Machine
5. Sequence Diagram
6. Activity Diagram
7. Reflection

2.2 Attachments

All images included in the report are also attached in *.png* image format and raw UML *.uxf* files. Please note that these models were created using the VSCode extension UMLet, rather than using the provided UMLet IDE, to allow concurrent modification. This may cause or already have caused compatibility issues when trying to read the files in the UMLet IDE. If you experience these issues, please try opening the files in UMLet on VSCode.

3 Requirements

3.1 Design Decisions

We decided to, instead of having *Open* and *Close* buttons, have a *Request Bottom* and *Request Top* button. These buttons come with a visual marker such as a light, called *Indicator Bottom* and *Indicator Top*.

Apart from this, we add a warning light so that the *Lock Keeper* can see the system has an error, called *Indicator Error*. Besides this, there is an *Emergency* button to stop/start the system in case of an emergency, and a *Proceed* button which indicates that, after the lock doors opened and the boat(s) entered the lock, the system may proceed.

Lastly, there are additional sensors for the slides to indicate if they are opened, *Slide Sensor Bottom* and *Slide Sensor Top* for each door respectively.

3.2 Assumptions

We assume that boats that can enter a lock safely (i.e. the door is open and there is no other boat inside/in front of them), that they will enter autonomously.

Also, we assume the water level between locks is consistent (i.e. the 'low' level of a lock is the same as the 'high' level of the lock downstream).

Thirdly, we assume the doors open and close within 20 seconds when properly operating, and the slides open and close within 7 seconds.

Lastly, we assume the water sensor is either reliable enough or there is sufficient redundancy to give accurate water height measurements all the time.

3.3 Actors

Actor	Goals
Button	Controls the doors, slides and directs the boats
Lock Keeper	Observes and corrects where needed, in case of a system malfunction
Boat	Pass through the locks
Water Sensor	Measures the height of the water
Door Sensor	Inform the system of door states

Table 1: Actor Table

3.4 Inputs & Outputs

Before writing down the requirements of the system, it is important to specify the inputs and outputs of a Lock, as well as the different states a Lock can be in.

Input Name	Input Description
InputDoorBottom	Receives signals from *Bottom Door Sensor*
InputDoorTop	Receives signals from *Top Door Sensor*
InputButtonBottom	Receives input from the *Request Bottom* button
InputButtonTop	Receives input from the *Request Top* button
InputButtonEmergency	Receives input from the *Emergency* button
InputButtonProceed	Receives input from the *Proceed* button
InputWaterLevel	Receives input from the *Water Level Sensor*
InputSlideBottom	Receives input from the *Slide Sensor Bottom*
InputSlideTop	Receives input from the *Slide Sensor Top*

Table 2: Inputs Table

Ouput Name	Output Description
OutputIndicateRequestBottom	Sends signals to the *Indicator Bottom* (light).
OutputIndicateRequestTop	Sends signals to the *Indicator Top* (light).
OutputIndicateError	Sends signals to the *Indicator Error* (light).
OutputDoorBottom	Sends signals to the *Bottom Door* to open or close.
OutputDoorTop	Sends signals to the *Top Door* to open or close.
OutputSlideBottom	Sends signals to the *Bottom Slide* to open or close.
OutputSlideTop	Sends signals to the *Top Slide* to open or close.

Table 3: Outputs Table

State Name	State Description
StateBoatBottom	Used to indicate there is a request for a boat passage from below (downstream)
StateBoatTop	Used to indicate there is a request for a boat passage from above (upstream)
StateBoatEntered	Used to indicate there is a boat in the lock (state 1 = up, state 2 = down, state 0 = none)
StateWaterHeightTop	Used to indicate the water height of the water upstream from the lock
StateWaterHeightBottom	Used to indicate the water height of the water downstream from the lock
StateSystemError	Used to indicate the system has reached an error or the Lock Keeper has pressed the emergency button

Table 4: State Table

3.5 Requirements

For the requirements, we split up the requirements into three sections: System (R1), Safety (R2), and Indication (R3).

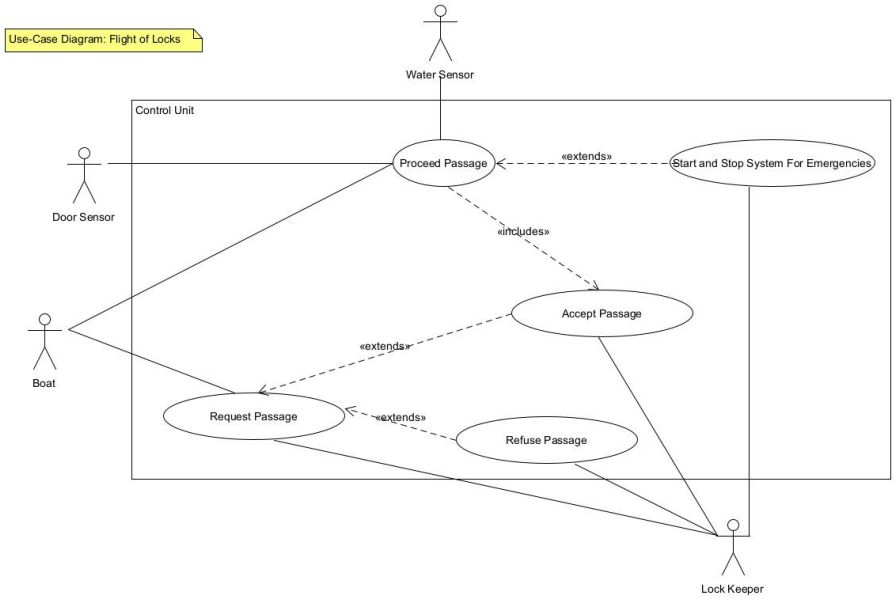
#	Requirements
R1	**System Requirements**
R1.0	When the 'pressed' signal is received by *InputButtonBottom*, the system sets *StateBoatBottom* to true within 500ms.
R1.1	When the 'pressed' signal is received by *InputButtonTop*, the system sets *StateBoatTop* to true within 500ms.
R1.2	When the 'open' signal is received by *InputDoorBottom* and *StateBoatBottom* is true, the system sets *StateBoatBottom* to false and *StateBoatEntered* to 1.
R1.3	When the 'open' signal is received by *InputDoorTop* and *StateBoatTop* is true, the system sets *StateBoatTop* to false and *StateBoatEntered* to 2.
R1.4	When the *StateBoatEntered* is currently state 1 and the 'proceed' signal is received by *InputButtonProceed* and the 'closed' signal is not given by *InputDoorBottom*, the system shall set *OutputDoorBottom* to 'close'.
R1.5	When the *StateBoatEntered* is currently state 2 and the 'proceed' signal is received by *InputButtonProceed* and the 'closed' signal is not given by *InputDoorTop*, the system shall set *OutputDoorTop* to 'close'.
R1.6	When the *InputWaterLevel* has been constant for 5 seconds and the 'open' signal is received by *InputSlideBottom* and not by *InputSlideTop*, the system shall store the water level in *StateWaterHeightBottom* within 100ms.
R1.7	When the *InputWaterLevel* has been constant for 5 seconds and the 'open' signal is received by *InputSlideTop* and not by *InputSlideBottom*, the system shall store the water level in *StateWaterHeightBottom* within 100ms.
R1.8	When the *StateBoatEntered* is state 1 and the *InputWaterLevel* indicates the water level is the same as *StateWaterHeightTop*, the system shall set *OutputSlideTop* to 'close' and *OutputDoorTop* to 'open'.
R1.9	When the *StateBoatEntered* is state 2 and the *InputWaterLevel* indicates the water level is the same as *StateWaterHeightTop*, the system shall set *OutputSlideBottom* to 'close' and *OutputDoorBottom* to 'open'.
R1.10	When the upstream lock stores its *StateWaterHeightBottom*, it stores the same value in *StateWaterHeightTop* of the current lock.
R1.11	When the downstream lock stores its *StateWaterHeightTop*, it stores the same value in *StateWaterHeightBottom* of the current lock.

R2	**Safety Requirements**
R2.1	When the 'pressed' signal is received by *InputButtonEmergency*, the system shall set *StateSystemError* to the negation of *StateSystemError* (like a switch).
R2.2	When the *StateSystemError* is true, the system shall not open and/or close doors and slides.
R2.3	When the *StateSystemError* is set to false after being true, the system shall resume operations as it was right before the *StateSystemError* was set to true, within 100ms.
R2.4	When the *InputWaterLevel* is not the same as *StateWaterHeightTop*, the system shall never open the top door.
R2.5	When the *InputWaterLevel* is not the same as *StateWaterHeightBottom*, the system shall not allow the bottom door to be opened.
R2.6	When the *InputDoorTop* is not 'closed', the system shall not allow the bottom door to be opened.
R2.7	When the *InputDoorBottom* is not 'closed', the system shall not allow the top door to be opened.
R2.8	When the *StateSystemError* is true, the slides and doors shall be operatable without the system (manual / bypass)
R3	**Indication Requirements**
R3.1	When *StateSystemError* is true, the system shall turn on *OutputIndicateError* within 100ms.
R3.2	When the doors do not close within 30 seconds of giving the 'close' signal, the system sets *StateSystemError* to true.
R3.3	When the doors do not open within 30 seconds of giving the 'open' signal to *OutputDoorX*, the system sets *StateSystemError* to true.
R3.4	When the slides do not close within 10 seconds of giving the 'close', the system sets *StateSystemError* to true.
R3.5	When the slides do not open within 10 seconds of giving the 'open', the system sets *StateSystemError* to true.
R3.6	When the 'pressed' signal is received by *InputButtonBottom*, the system shall set the *OutputIndicateRequestBottom* to true, within 100ms.
R3.7	When the 'pressed' signal is received by *InputButtonTop*, the system shall set the *OutputIndicateRequestTop* to true, within 100ms.

Table 5: Requirements Table

4 Use Cases & Diagrams

Now that the goals of the system are specified in the requirements, we are able to come up with Use Cases that we'll be able to achieve these goals.

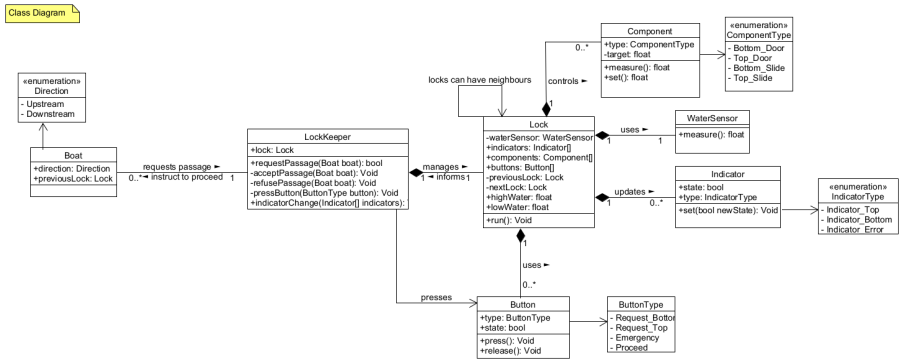


Desc = Short Description — Pre = Precondition
Post = Postcondition — Error = Error Situation
Er.S = System Error State — Actor = Actor
Trig = Trigger — Proc = Standard Process

Name	Element	Text
Request Passage	Desc	A boat requests to enter the lock either upstream or downstream.
	Pre	The system is up and running and all sensors function nominally.
	Post	The driver of the boat goes on standby, awaiting the go-ahead of the Lock Keeper.
	Error	The Lock Keeper does not notice the incoming boat's request.
	Er.S	The boat is not able to request entering the lock.
	Actor	Boat, Lock Keeper.
	Trig	A boat approaches the lock.
	Proc	
	1.	The Lock Keeper notices the boat.
	2.	The Boat awaits further instructions.
	1*	The Lock Keeper does not notice the boat.
	2*	The Boat honks its horn.
	3*	The Lock Keeper notices the boat.
	4*	The Boat awaits further instructions.
Accept Passage	Desc	Lock Keeper grants the boat passage.
	Pre	There are no boats entering from the side of the arriving boat, and the boat has requested entry.
	Post	The Lock Keeper has pressed the request button.
	Error	There are boats entering the lock from the side of the arriving boat.
	Er.S	The Lock Keeper's boat request is cancelled.
	Actor	Boat, Lock Keeper.
	Trig	A boat requests to enter the lock.
	Proc	
	1.	The Lock Keeper decides the boat is allowed access.
	2.	The Lock Keeper presses the boat request button.
	3*	There are boats currently entering the lock from the side of the arriving boat.
	4*	The Lock Keeper's request is cancelled.
	5*	The Lock Keeper waits for the boats to finish entering.
	6*	If the boat is still requesting entry, the Lock Keeper presses the button again.
Reject Passage	Desc	The Lock Keeper denies the boat passage.
	Pre	The boat has requested entry.
	Post	The Lock Keeper informs the requesting boat of the entry denial.
	Error	-
	Er.S	-
	Actor	Boat, Lock Keeper.
	Trig	A boat requests to enter the lock.
	Proc	
	1.	The Lock Keeper decides the boat is not allowed access.
	2.	The Lock Keeper informs the boat of the access denial.
	3.	The boat receives the message and goes away or requests access again.

Proceed Passage	Desc	The boat passes through the lock.
	Pre	Access has been granted (by pressing the button), and passage is safe (Sensors OK & no boats inside)
	Post	The boat passed through the lock safely.
	Error	The boat had an accident while passing through.
	Er.S	The boat is no longer functional (cannot pass through).
	Actor	Boat, Water Sensor, Door Sensor (& Lock Keeper).
	Trig	The door on the side of the boat is open.
	Proc	
	1.	The boat enters the lock.
	2.	The doors close behind it.
	3.	The boat is elevated/lowered by the water level.
	4.	The doors in front of it open.
	5.	The boat exits the lock.
	¿	*The alternative process can occur at any of the 5 steps.*
	?*	The boat has an accident.
	?*	The 'Start & Stop System for Emergencies' use-case triggers.
	?*	The process continues where it left off.
Start & Stop System for Emergencies	Desc	Stops the system in case of an emergency.
	Pre	There is an issue with one or more boats or with the hardware of the lock.
	Post	The issue has been resolved safely.
	Error	-
	Er.S	-
	Actor	Lock Keeper, System.
	Trig	An emergency happens with the hardware, or with the boats passing through.
	1.	The Lock Keeper presses the emergency button OR the System triggers the emergency mode.
	2.	The boat/hardware issue(s) is/are resolved.
	3.	The Lock Keeper presses the emergency button again to resume the system.

5 Class Diagram



For our class diagram we decided upon the following:

1. The Lock Keeper plays a key role in the system, without them, the lock cannot function properly, which is made more extreme by saying the lock cannot exist without the lock keeper. This means that lock keepers cannot be swapped between locks, and only one lock keeper manages a lock at any time, and every lock is only managed by one lock keeper.
The only interaction the lock keeper has with the lock is through the use of the Buttons, and the only interaction the lock has with the lock keeper is through the indicators.

2. To reduce complexity and enhance readability enumerations are used to group, what might otherwise be separate, classes into Components (Slides Doors), Indicators (Top, Bottom Error), and Buttons (Request top bottom, Emergency, and Proceed).

3. Boats are passive and only request access through the lock keeper. Boats are autonomous and do not run into closed doors, and know when to proceed (e.g. when the boat is in a lock and the door in front of it opens, it passes through without requiring a function call)

4. The locks are autonomous systems with access to previous and next locks if available.

5. The Boat has a pointer to the current lock. The Lock itself has pointers to the next and previous lock in the sequence.

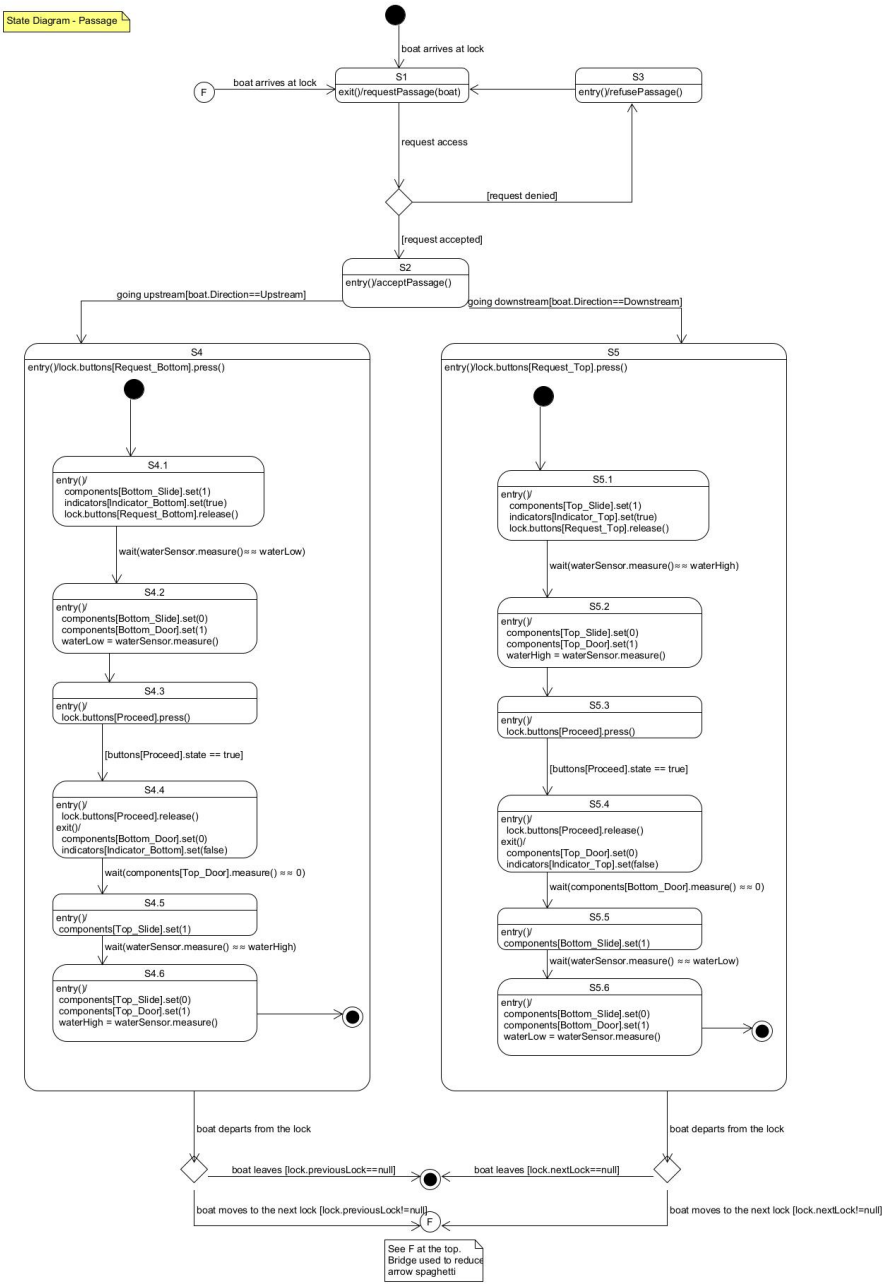
6 State Machine

For the first diagram "*State Diagram - Passage*" we assume that the system is already running when a boat requests passage for a lock. The second diagram is the "*State Diagram - Emergency*", where we take into consideration what happens once the system runs into an error or that a Lock Keeper decides to put the Lock in an emergency state.

State	**Description**
S1	The boat is outside a lock.
S2	The boat is granted passage.
S3	The boat is not granted passage.
S4	The boat is travelling upstream.
S5	The boat is travelling downstream.
S6	The boat is inside the lock.

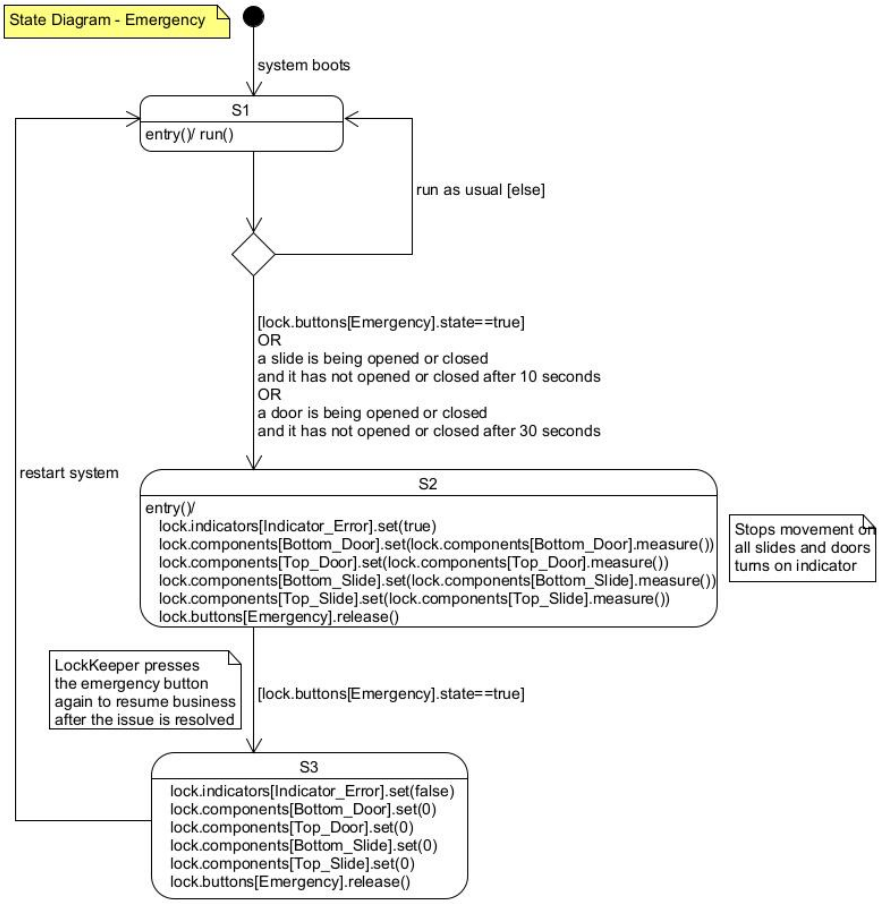
Table 6: State Table for State Diagram - Passage

There are specific sub-states inside S4 and S5 respectively regarding the activities that will take place in order to change the water level for a boat to enter.

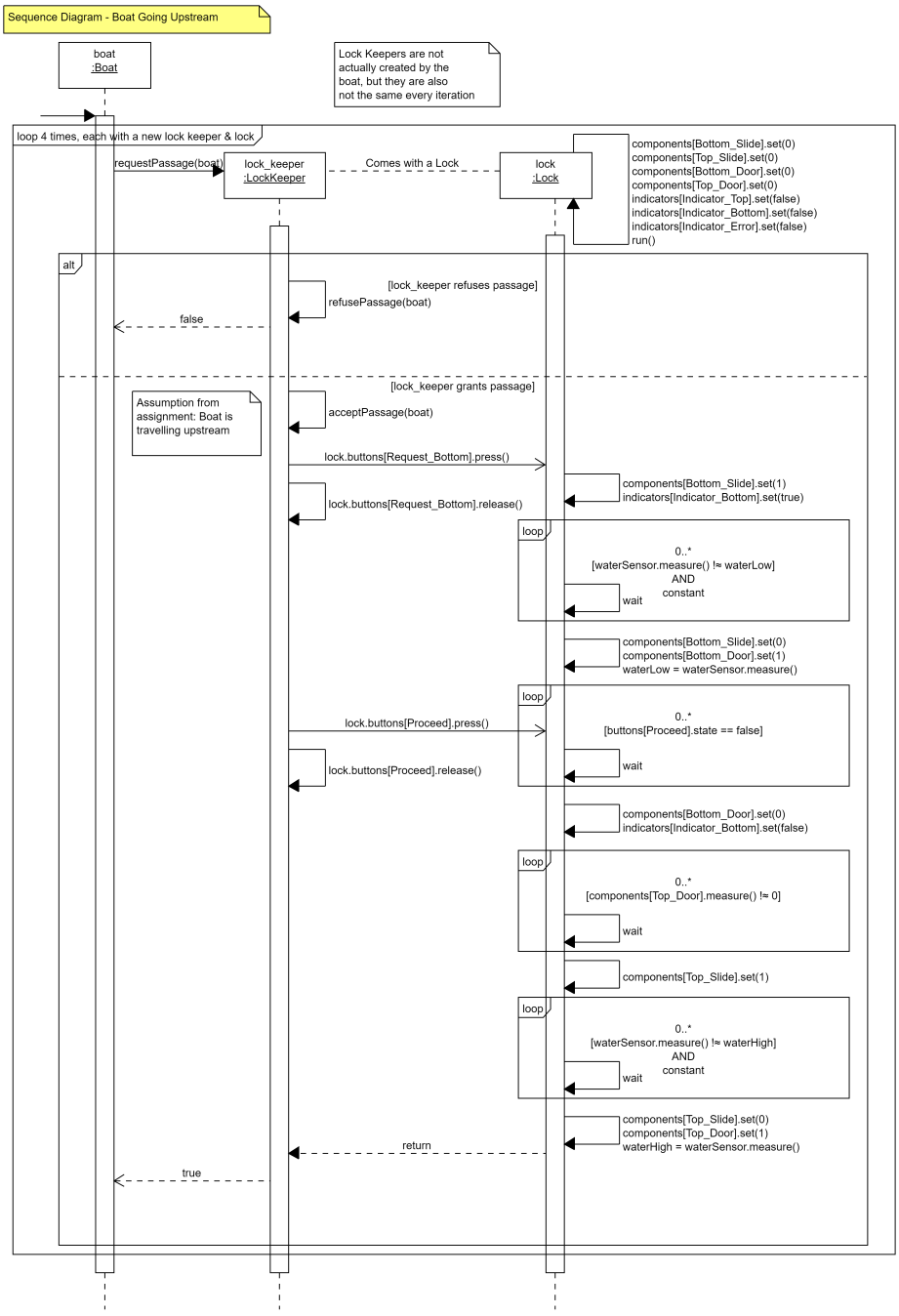

State Diagram - Passage
boat arrives at lock
F
boat arrives at lock
S1
exit()/requestPassage(boat)
S3
entry()/refusePassage()
request access
[request denied]
[request accepted]
S2
entry()/acceptPassage()
going upstream[boat.Direction==Upstream]
going downstream[boat.Direction==Downstream]
S4
entry()/lock.buttons[Request_Bottom].press()
S4.1
entry()/
components[Bottom_Slide].set(1)
indicators[Indicator_Bottom].set(true)
lock.buttons[Request_Bottom].release()
wait(waterSensor.measure()≈≈ waterLow)
S4.2
entry()/
components[Bottom_Slide].set(0)
components[Bottom_Door].set(1)
waterLow = waterSensor.measure()
S4.3
entry()/
lock.buttons[Proceed].press()
[buttons[Proceed].state == true]
S4.4
entry()/
lock.buttons[Proceed].release()
exit()/
components[Bottom_Door].set(0)
indicators[Indicator_Bottom].set(false)
wait(components[Top_Door].measure() ≈≈ 0)
S4.5
entry()/
components[Top_Slide].set(1)
wait(waterSensor.measure() ≈≈ waterHigh)
S4.6
entry()/
components[Top_Slide].set(0)
components[Top_Door].set(1)
waterHigh = waterSensor.measure()
S5
entry()/lock.buttons[Request_Top].press()
S5.1
entry()/
components[Top_Slide].set(1)
indicators[Indicator_Top].set(true)
lock.buttons[Request_Top].release()
wait(waterSensor.measure()≈≈ waterHigh)
S5.2
entry()/
components[Top_Slide].set(0)
components[Top_Door].set(1)
waterHigh = waterSensor.measure()
S5.3
entry()/
lock.buttons[Proceed].press()
[buttons[Proceed].state == true]
S5.4
entry()/
lock.buttons[Proceed].release()
exit()/
components[Top_Door].set(0)
indicators[Indicator_Top].set(false)
wait(components[Bottom_Door].measure() ≈≈ 0)
S5.5
entry()/
components[Bottom_Slide].set(1)
wait(waterSensor.measure() ≈≈ waterLow)
S5.6
entry()/
components[Bottom_Slide].set(0)
components[Bottom_Door].set(1)
waterLow = waterSensor.measure()
boat departs from the lock
boat departs from the lock
boat leaves [lock.previousLock==null]
boat leaves [lock.nextLock==null]
boat moves to the next lock [lock.previousLock!=null]
boat moves to the next lock [lock.nextLock!=null]
F
See F at the top.
Bridge used to reduce
arrow spaghetti

State	Description
S1	The system is running properly.
S2	The system has runned into an error.
S3	The system secures all doors and slides.

Table 7: State Table for State Diagram - Emergency



7 Sequence Diagram



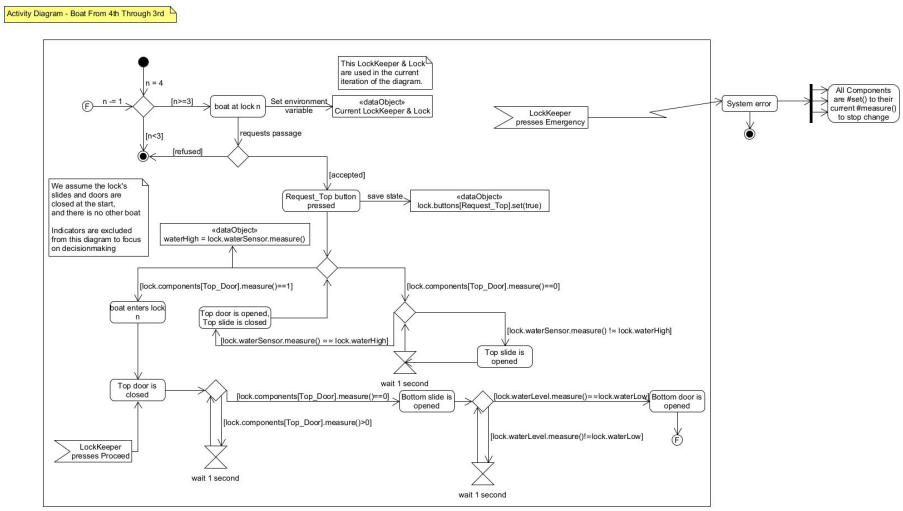
7.1 Notes

The sequence diagram shows the order of events from when a boat arrives at a lock and requests passage, to the moment it leaves the lock to move on to the next.
As per the assignment, the sequence repeats 4 times, each time with a new Lock Keeper and subsequent Lock.
The call to the lock keeper does not create the lock keeper (and lock), but rather pulls it from the environment; it is denoted this way to indicate the lock (keeper) is not the same every iteration.

7.2 Connection with other Diagrams

The sequence diagram is closely connected with the Class diagram. It uses many of the methods, fields and types indicated there. Transitively, it relates to the Use-case diagram (as the Class diagram is closely related to the Use-case diagram) to define the main (most important) classes.

8 Activity Diagram



9 Reflection

In the making of this assignment we almost exclusively worked together on all elements involved. Spending time during the tutorials, after the tutorials and at moments scheduled together outside of regular hours. We experienced difficulty with keeping all diagrams, requirements, etc. streamlined and consistent with one-another. Besides this, it took more time than we expected to create some of the diagrams, but that problem was in large resolved after we created the class diagram, which turned out to be the backbone of other diagrams like the activity and sequence diagram. We also ran into the challenge of incorporating requirements into the software. Perhaps this is still a question we have to this day: How detailed should our requirements be, and to what degree are they reflected in the other diagrams? (see for example R3.1) Lastly, we had some issues with lock-to-lock communication, which led us to create a system that works mostly autonomously within a single lock, and only has limited connectivity between neighbouring locks. We could improve system responsiveness by strengthening the connection, but it would also drastically increase complexity.