Report Tinlab Advanced Algorithms

Noëlle Clement (0935050)
&
Tommie Terhoeve (0926280)

April 15, 2019

Contents

1 Introduction

This report serves as additional information and justification for the model developed for the Advanced Algorithms course.

The assignment states that students need to model a fully automated water lock. The assignment description is provided in appendix II in Dutch. The given model represents our vision of this. Further information on the specifics will be provided in the following chapters, including the decisions that led to our final model. Finally, the model has been verified in Uppaal to check whether our model meets the specified requirements.

For the scope of our assignment certain components or aspects haven't been included. Where required, additional information is provided.

2 Requirements

In the following chapter the requirements and specifications for our model are specified. Additionally, considerations on both are provided.
The main source used to define the requirements is the given assignment description. Additionally where required, we have used scientific resources. In some cases we have used our educational knowledge and common sense.

As stated in the assignment description, we're not taking unexpected circumstances into account in our model. We have however made some exemplary non-functional requirements, of which we have implemented a few in the model.

2.1 Requirements

#	**Requirement**
1.	A boat can move from one side of the lock to the other (both directions)
2.	The lock works fully automated
3.	The water level in the lock chamber can change (increased/decreased)
4.	There are two boat-queues, one for each side of the lock
5.	Multiple boats can enter the lock (chamber)
6.	Both sides of the lock have a door, which can be opened and closed
7.	Signals are given to the outside world *7.1 Door is opening* *7.2 Door is open* *7.3 Door is closing* *7.4 Door is closed* *7.5 Water level is changing*
8.	Entrance to and departure from the lock may take maximum one hour (including opening and closing of doors)

Table 1: Functional requirements

#	**Requirement**
9.	Both lock doors cannot be open at the same time
10.	The water level may not change while a lock door is open
11.	Both doors may not close while a boat should be entering or departing the lock chamber
12.	The lock chamber may not flood or be pumped until fully empty
13.	The lock doors are closed when the lock isn't in use

Table 2: Non-functional requirements: Safety

#	**Requirement**
14.	The lock has more than 1 water level sensor
15.	Both lock doors open and close successfully 95% of the time

Table 3: Non-functional requirements: Reliability

2.2 Considerations

2.2.1 Functional requirements

During our lectures the teacher prescribed certain features of the to-be-modelled lock. Some of these have been described in the written assignment, however, some have not. We have still included those in the list of prescribed requirements, since it was clear that these needed to be included. See Appendix I for an overview of these prescribed requirements.

As shown in Appendix I, a few requirements have been partially or more extensively implemented than prescribed. Below is described how these differ from prescribed, and why they have been implemented in this way.

Req. #	**Implemenation**
1.	A core functionality of a lock is that a boat needs to be able to move from one side to another. In our model we wanted the boat to be being able to move in both directions.
4.	Since the boats are able to move through the lock from both sides, queues need to be implemented for both sides.
5.	A minimum of one boat needs to be able to enter the lock (chamber). We added the ability to have multiple boats in the lock chamber, to have added reality.

Table 4: Expanded implementation of prescribed requirements

2.2.2 Non-functional requirements

As stated before, we chose to focus on the functional requirements in the model. However, we wanted to describe and (where possible) implement some non-functional requirements. We chose to focus on the quality attributes 'safety' and 'reliability'.

Oxford Dictionary defines safety as follows : *"the condition of being protected from or unlikely to cause danger, risk, or injury"*. With this in mind we created requirements which will decrease the possibility of a dangerous scenario occurring.

One of the key elements of increasing the reliability of a system, is including some way of redundancy to the given system. Generally, this is achieved by including back-up systems such as extra sensors and actuators. By comparing the readings of multiple identical sensors a faulty one can be detected and replaced [1].

Security and reliability are terms that are often used in the same context, even though they are very different from each other. Scenarios can be unsafe (safety) without any components failing (reliability). On the other hand, components can also fail (reliability) without any dangerous scenarios occurring (safety) [2].

2.3 Specifications

For this assignment we are focusing on the required components for the model. Hence, we will not specify the technical components (the specifications) of the to-be-developed lock. For more information on the components, see section 3.1: 'Modelled components'.

3 The Model

The xml file that is included with this report as Appendix IV contains our model. The model has been made in Uppaal, an application in which one can create 'labeled timed state transition diagrams'. In this chapter a description is given of the modelled components. Furthermore, the operation of the model, our considerations are also included in this chapter. Finally a comparison with the model criteria of Vaandrager is made.

3.1 Modeled components

For our model we have modelled the following components:

- main controller
- request handler
- water-pump
- a pair of doors
- signals
- a simple boat

3.1.1 Main controller

The main controller is the central controlling unit of the system, hence the name. The main controller waits for input from the request handler to start the process of moving the boats from one side to the other.

3.1.2 Request handler

The request handler handles the requests and modifies the three queues in the water lock. The three queues are defined as follows:

- Queue for the boats waiting on the left side of the lock.
- Queue for the boats currently in the lock chamber.
- Queue for the boats waiting on the right side of the lock.

3.1.3 Water-pump

The water pump is a simple component that has two main functions: To pump water into the lock and to pump water out of the lock.

3.1.4 Doors

The doors are identical and serve to regulate the water level in the lock chamber.

3.1.5 Signals

The signals provide information to the crew on the boats about the state of the water lock.

3.1.6 A simple boat

The boat's only purpose is to request to be added to a queue. This request will be handled by the request handler.

3.2 Operation of the model

To explain the full process of our modelled system we have provided a detailed description.
The scenario is as follows: The queue at the left side of the lock is full and the water level in the lock chamber is higher than the water level at the left side of the lock. Note: the lock works in both directions, the scenario provided below is for explanatory reasons.

1. The main controller waits for either the left or right queue to be full. Meanwhile the boat will keep sending requests to the request handler which fills up either one of the queues.
2. The left queue is full, the main controller defines the left door as the first door to be opened and the right door as the last door to be opened.
3. The main controller checks the water level and detects that the water level in the lock chamber is too high.
4. The main controller turns the pump on and pumps out water until the water level in the lock chamber is equal to the water level left of the lock.
5. The signal signals that the left door is opening.
6. The main controller opens the left door.
7. When the left door is fully open the signal will signal that the left door has fully opened and that it is safe to enter the lock chamber.
8. The main controller transfers the left side queue to the queue inside the lock chamber.
9. The main controller closes the left door.
10. The signal signals that the left door will be closing.
11. The signal signals that the left door is fully closed when the door is fully closed.
12. The signal signals that the water level will be changing.

13. The main controller compares the water level in the lock chamber with the water level at the right side of the lock.
14. The signal signals that the water level will be changing.
15. The main controller turns the pump on to pump in water mode until the water level in the lock chamber is equal to the right side of the lock.
16. The signal signals that the right door will open.
17. The main controller opens the right door to open.
18. When the right door is fully open the signal will signal that the right door has fully opened and that it is safe to exit the lock chamber.
19. The main controller empties the queue in the lock chamber.
20. The signal signals that the right door will close.
21. The main controller closes the right door.
22. The signal signals that the right door has fully closed.
23. The signal turns off.

3.3 Considerations

See Appendix I for an overview of all the implemented requirements. Our considerations will focus on choices made during the design process of the model.

To keep our model organized and easily readable we decided to make a separate template for each component. This also allows us to easily expand our model in the future. A possible disadvantage of this is that one is not able to verify whether two components are simultaneously in a certain state. We experienced this during the verification process.

To simulate the three queues of our water lock we used integer arrays with a capacity of 3 integers. This allows us to use unique boat ID's in the future if necessary. Even though we do not make use of those at the moment (to prevent added complexity). Using arrays instead of a simple integer counter also means that debugging is easier. Since we do not use unique boat ID's at the moment, some aspects are not verifiable with our current model, such as verifying that a particular boat successfully passes through the lock.

We decided to not implement water level sensors in our model to prevent the model from becoming too complex. Instead we let the main controller check and regulate the water level by switching on and off the water pump. However, the model has been designed in such a way that makes it possible to implement this expansion easily.

For the signaling system we decided to not go too much in-depth since it is irrelevant for this version of our model. For example we decided to make use of one signaling system for both directions. We also modeled the type of signal that will be displayed, not the actual (electrical) signal itself.

3.4 Modelcriteria

To validate the quality our model we checked if our model meets the criteria made by Frits Vaandrager [3]. Please note, that the following statements reflect our opinions.

3.4.1 Criteria I

"A good model has a clearly specified object of modelling, that is, it is clear what thing the model describes. The object of modelling can be (a part of) an existing artefact or physical system, but it may also be a document that informally specifies a system or class of systems (for instance a protocol standard)..."

Our model meets this criteria since the model clearly models an automated water lock.

3.4.2 Criteria II

"A good model has a clearly specified purpose and (ideally) contributes to the realization of that purpose..."

Our model meets this criteria since it can be used to verify the logic behind our water lock system.

3.4.3 Criteria III

"A good model is traceable: each structural element of a model either (1) corresponds to an aspect of the object of modelling, or (2) encodes some implicit domain knowledge, or (3) encodes some additional assumption..."

Our model meets this criteria since every structural element represents a component of the water lock.

3.4.4 Criteria IV

"A good model is truthful: relevant properties of the model should also carry over to (hold for) the object of modelling..."

Our model meets this criteria since it's an accurate representation of a real water lock.

3.4.5 Criteria V

"A good model is simple (but not too simple)..."

Our model meets this criteria since it's simple enough to be easily readable, but detailed enough to be verified properly.

3.4.6 Criteria VI

"A good model is extensible and reusable, that is, it has been designed to evolve and be used beyond its original purpose..."

Our model meets this criteria since it's modular and more components can be easily added.

3.4.7 Criteria VII

"A good model has been designed and encoded for interoperability and sharing of semantics."

Our model does not meet this criteria since it can't be linked to any other models.

4 Verification

To verify the logic behind the various components of our model we used CTL* and Uppaal. We observed that some components were theoretically impossible to verify in our current model. In this chapter we will go over these components and elaborate on the verification results.

4.1 Verified components & results

Req.#	**CTL Uppaal**	**Results**
1.1	A[](Main_Controller.servingQueue==0 imply Main_Controller.Process_done	not satisfied
1.2	A[](Main_Controller.servingQueue==1 imply Main_Controller.Process_done)	not satisfied
2.	A[] not deadlock	satisfied
4.	A[](Request_Handler.size > 1)	satisfied
6.	A[](Door(0).Opened || Door(1).Opened || Door(0).Closed || Door(1).Closed)	satisfied
7.	Impossible to check in Uppaal	not satisified
9.	A[]not(Door(0).Opened && Door(1).Opened)	satisfied
10.1	A[]not(Door(0).Opened && (Waterpump.Pump_in || Waterpump.Pump_out))	satisfied
10.2	A[]not(Door(1).Opened && (Waterpump.Pump_in || Waterpump.Pump_out))	satisfied
11.1	A[]not(Door(0).Is_Closing && (Main_Controller.Boats_enter |||*Main_Controller.Empty_chamber*))	satisfied
11.2	A[]not(Door(1).Is_Closing && (Main_Controller.Boats_enter || Main_Controller.Empty_chamber))	satisfied

Table 5: Verified requirements

Requirement 1 and 2 both use the liveness operator (in Uppaal: A[]p imply q). Liveness is dependent on the passage of time, which was not fully implemented in the model.

Requirement 8 is impossible to verify due to not being able to verify if the the model is in multiple states in different components at the same time.

4.2 Non-verified components

In table 6 requirements are shown that we tried to verify, but couldn't due to syntax problems. We have not found a way to verify these requirements in a proper way with the right syntax.

Req.#	CTL Uppaal	Results
5.	A[](Request_Handler.queue0 != NULL && Request_Handler.queue1 != NULL)	syntax error
14.	A[](Main_Controller.Wait_for_boat && Door(0).Closed && Door(1).Closed)	syntax error

Table 6: Non-verified requirements: tried but failed

In table 7 requirements are shown that we did not attempt to verify, because our model was not designed in such a way to make it possible to verify these specific requirements.

Req.#	Reason
3.	There is no state in which there is always a difference in water level that can be detected
8.	The model doesn't keep track of passed time
12.	Unexpected circumstances aren't taken into account
14.	Unexpected circumstances aren't taken into account
15.	Unexpected circumstances aren't taken into account

Table 7: Non-verified requirements: not verifiable

5 Conclusion

For the TinLab Advanced Algorithms we received the assignment to design a system model for a water lock. To realize this we needed to set up requirements, create a model and finally verify the requirements for the model. Like with most projects, we discovered some limitations and possible improvements for our current version.

5.1 Limitations

During the verification process, it became apparent that certain requirements would not be verifiable (in the current model). For some this was due to a liveness problem, for others we could not verify whether multiple components were in certain states simultaneously. Also, some requirements have not been verified because the requirement itself was not implemented in the model.

It is important to note that what has been verified is limited to what we set as requirements. During the development process, we mainly focused on the functional requirements. Some non-functional requirements have been formulated as well, but mostly to serve as an example.

For more specific limitations to our model, please refer to section 3.3. For an overview of what components have not been verified, please refer to section 4.2.

5.2 Advice

During the verification process for requirement 1 and 2, both returned 'not satisfied' because of a liveness issue. Should one want to satisfy these requirements, a timer has to be put on every state so that it can not infinitely stay in one certain state.

To make it possible to verify whether multiple components are in certain states simultaneously, one would be forced to modify the model. Parallel composition can be used to make sure both components can only transition after certain states in the main components. However, for this a major and complex redesign is required.

It is advised to include and implement more non-functional requirements. These requirements (like safety and reliability) are immensely important when a system such as a lock will be put into use, also with ethical responsibility in mind.

Should one wish to expand the functionality of the communication system between the lock and the boats, security should be kept in mind. With more external communication, also comes an increased chance of vulnerabilities. While a more advanced communication / queuing system would have its advantages, these are things that need to be taken into account.

References

[1] Roy Billinton and Ronald Norman Allan. *Reliability evaluation of engineering systems*. Springer, 1992.

[2] Nancy Leveson. *Engineering a safer world: Systems thinking applied to safety*. MIT press, 2011.

[3] Frist Vaandrager. What is a Good Model? http://www.cs.ru.nl/ fvaan/PV/what_is_a_good_model.html.

6 Appendix

I. Prescribed requirements by teacher

II. Assignment description, provided by teacher (in Dutch)

III. Personal reports (in Dutch)

IV. Model

V. Version administration (github log)

Appendix I: Prescribed and implemented requirements

In the following table an overview is given of the prescribed requirements. An 'X' states this exact requirement has been prescribed. An 'x' states that a form of this requirement (often a more simple one) has been prescribed, and we have modified it to our specific lock.

Also an overview is given of which functional requirements have been implemented in the model. Please note, that in the table none of the non-functional requirements have been marked as 'implemented' because we focused on implementing the functional requirements.

#	**Requirement**	**Prescr.?**	**Impl.?**
1.	A boat can move from one side of the lock to the other (both directions)	x	X
2.	The lock works fully automated	X	X
3.	The water level in the lock chamber can change (increased/decreased)	X	X
4.	There are two boat-queues, one for each side of the lock	x	X
5.	Multiple boats can enter the lock (chamber)	x	X
6.	Both sides of the lock have a door, which can be opened and closed	X	X
7.	Signals are given to the outside world	X	X
8.	Entrance to and departure from the lock may take maximum one hour (including opening and closing of doors)		
9.	Both lock doors cannot be open at the same time	x	
10.	The water level may not change while a lock door is open	x	
11.	Both doors may not close while a boat should be entering or departing the lock chamber	x	
12.	The lock chamber may not flood or be pumped until fully empty		
13.	The lock doors are closed when the lock isn't in use		
14.	The lock has more than 1 water level sensor		
15.	Both lock doors open and close successfully 95% of the time		

Table 8: Prescribed requirements