Charging Station
Software Test

Aarhus Institue of Technology

Author:
Sune Dyrbye,
Morten Høgsberg
Yevhen Parolia

Date: March 24, 2022

Contents

Introduction

This journal has been writen by the SWT group **SWTF22 3**, consisting of

- Morten Høgsberg Futtrup Kristensen (stud. nr.: 201704542)

- Sune Andreas Dyrbye (stud. nr.: 201205948)

- Yevhen Parolia (stud. nr.: 20112870)

It is the result of the mandatory assignment **Obligatorisk Handin 2** in the course **ST4SWT-02 Software test**. Design and reflections is presented here, while the sourcecode is available in the github repository at:
https://github.com/UniquesKernel/ChargingStation

The system has been tested through a Jenkins CI job at
http://ci3.ase.au.dk:8080/job/SWTF22_3_handin2/

Charging Station Design

Our design relies heavily on interfaces to allow testing the system through hardware simulations. It is thus easy to create a piece of code that sends the same signals to the system as the physical hardware inmplementation would, allowing for testing of the individual units before implementation. This is taken further, as the internal components of the system (*stationControl* and *ChargeController*) interacts through an interface, allowing the *chargeControlle* to be replaced for upgrades or testing of *stationControl*.

The design of the charging station is displayed in Figure 1. Central to the system is the *station-Control* class, which handles all interaction with the user, through various interfaces (display, door and rfid detector). It also interacts with the *chargeController* through another interface. The *charge-Controller* meanwhile interfaces with the USB charger, cobntrolling it, and also displays charging status on the display through the display interface.
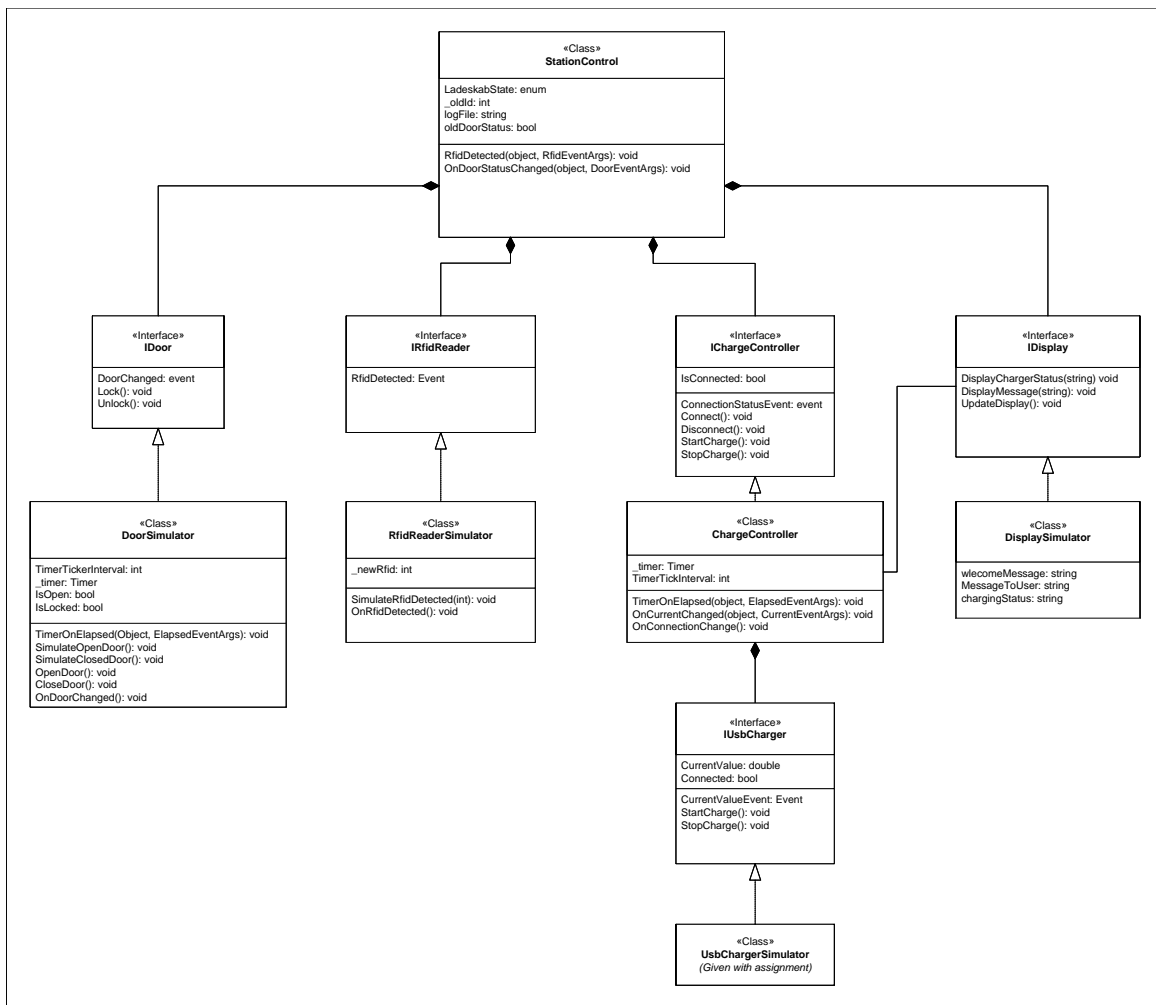

Figure 1: Class diagram for the charging station system

The charging station is first activated by the door opening, causing the station to ask the user to connect the phone. The user then connects the phone and closes the door, which causes the station to ask for an RFID. If the phone has not been correctly connected (as cheked by the chargeController)

the station reports this to the user through the display. If the phone has been properly connected when the RFID is detected the door locks and the station starts charging the phone. This continues until the charging is complete or the same RFID as was used to start the charging is detected. When the RFID of the user is detected after charging has begun, the system stops the charging and unlocks the door, alowing the user to disconect their phone. This sequence is illustrated in Figure 2.
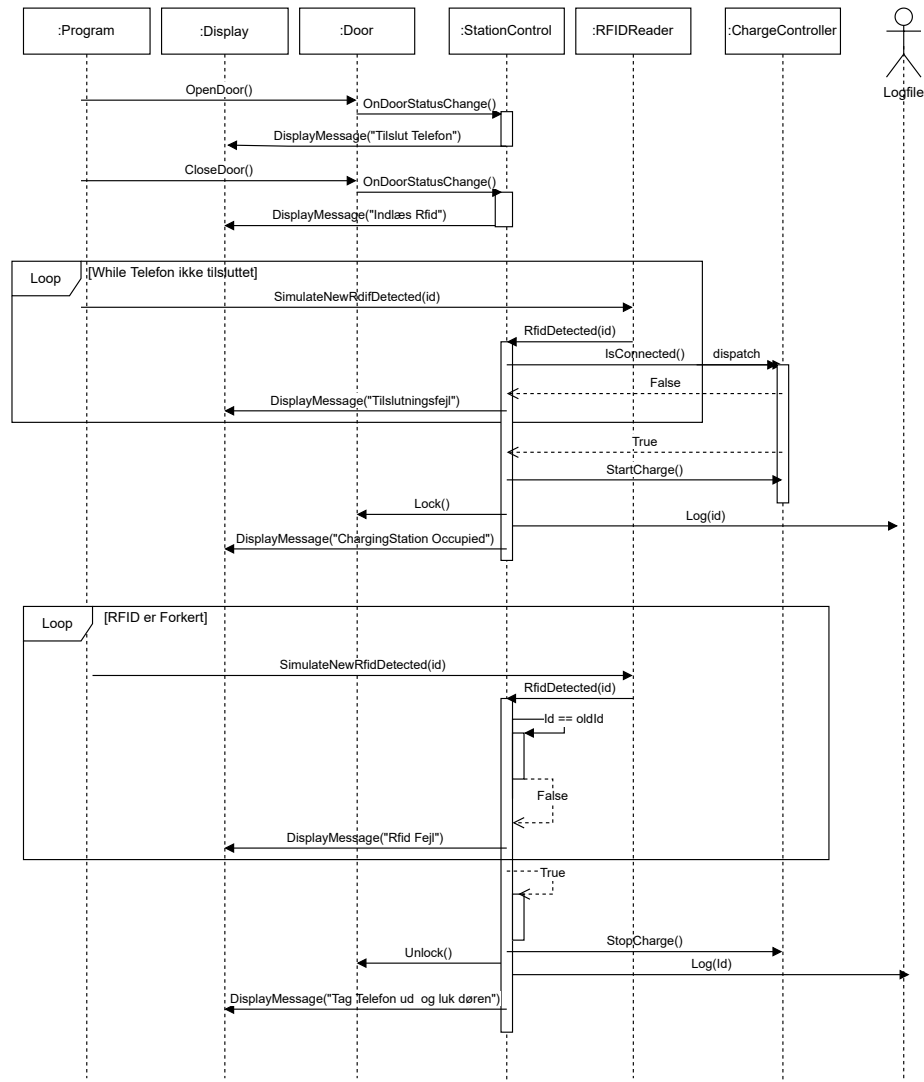
Figure 2: Sequence diagram describing normal opperation of the charging station

Design Reflection

The major component in design philosophy for the project have been to allow for a low coupled and testable design. In extension of that the use of events in the observer pattern means that it has been unnessesary to use threads to handle multiple events.

SOLID

Take cognizance of the SOLID principles. We have divvied up the respondsible of each part of the ChargingStation system into seperate classes.

This seperation of responsibility between the class makes it effortless to change one part of the system without impacting or breaking the other functionalities. It also ensures that each class can be tested seperately.

All components of the system is assembled with the use of Interfaces to ophold the open and close principles. It makes it painless to extend the functionalities of the system without having to change existing implementaions.

The principles of Interface segregation and dependency inversion are used extensively to ensure that Station controller (a high level module) in no way depends on specific implementations of its components but rather interface abstractions.

Events

To handle event, we implemented the observer pattern to handle any changes in values either from user actions or through time based events such a regular variable updates with regular time intervals.

While such things could have been handled with the use of threads, we chose to use event driven programming instead as implementing threaded system can be more complex and harder to test.

Conclusion

While the SOLID principles makes everything loosely coupled, it does mean a marginally more complex codebase as the solution increases in scope. It is however, the better solution for the reasons mentioned above, making it lossely coupled and easilier testable.

The use of event driven programming rather than threaded programming follow the same argument where testability and simplicity was chosen of complexity.

In conclusion, due to these design considerations it means that mock classes could be used to ensure a high test coverage where each aspect of a class is regiously tested.

References