SWEN30006 Software Modelling and Design

Assignment 1 Report

Paul Hutchins - 1160468
Jade Siang - 1170856
Kian Dsouza - 1142463

THE UNIVERSITY OF MELBOURNE

SWEN30006
The University Of Melbourne
13/09/2021

1 Initial Design Brief

An initial analysis of the task indicated the new system to be an extension of the current system's functionality through the addition of new robot types and ability to calculate the total charge of the delivery service.

To understand the scope of the existing system, we first created a Domain Model Diagram to show the conceptual classes present. With the aid of the Domain Model, we assessed that in order to maintain High Cohesion and Low Coupling, it would not be best practice to preserve functionality of the 'Robot' class, but to abstract and extend it to add the further functionality for the additional Robot types. To reduce Coupling we also decided to move the ChargeFee into its own class outside of Simulation, but still instantiate it within Simulation to maintain the current level of Cohesion and to maintain Simulation as the Information Expert. Our Domain Model which reflects these decisions can be seen in Figure 1.

2 Implementation

Below we will discuss the implementation and our reasoning behind our design choices. We have also included a Design Class Diagram, Figure 2, to show how classes interact within our extension of the program. Also included is a Design Sequence Diagrams, Figure 3, to show how the ChargeFee capability calculates the service charge.

2.1 Additional Robot Types

Within the implementation of new robot types we decided to create new software class in order to maintain High Cohesion and Protected Variation, and the use of an abstract Robot class promoted code reuse between the different Robot types and minimized Coupling.

We created new RegularRobot, FastRobot, and BulkRobot classes which extended the original Robot class. We had considered keeping the RegularRobot as the Robot class and extending the Fast and Bulk Robots. However, we ultimately decided against this as if they decided to change the implementation of the RegularRobot in the future it would require modification of all other robot types and could possibly end up with bloating and Low Cohesion. The use of a separate class for all 3 of the robots, that all extend functionality from a basic robot, results in higher overall Cohesion and less redundancy. It also allows the other robot types to only overwrite the methods they need to without needing to change basic methods, if it's decided that the RegularRobot should operate differently.

When implementing the Bulk robots, it was stated that they have no hands, but can hold up to 5 items in the tube, with items being delivered in a last in first out order. To imitate this, we decided to continue to implement the concept of having a 'hand' to hold an item, along with a tube which could hold a maximum of 4 items by Protected Variation. The item in the hand would act as the last item in (first item out) and would be checked/assigned first, before the tube. This approach ensured that we are maximizing code reuse and minimizing

bloating and the need to unnecessarily override methods, whilst still ensuring first in last out from the stack. The alternative to this was to completely disregard the hand functionality and to override and rewrite the Operate method within Robot to accommodate for the tube. However, this approach would have required more code to be written, and cause unnecessary bloating in the class, reducing overall Cohesion.

2.2 Adding a Charge, a Fee, and a Maintenance Cost to the Log

We decided to implement a separate ChargeFee class outside of the Simulation class in order to minimize Coupling and allow for the possibility of additional fee charging capabilities to be included in the future. An example of this could be if they decided to introduce and implement different methods of calculation for service fee and maintenance fee. Our implementation would allow for this, multiple instances of ChargeFee can be created as a result of an independent class.

The ChargeFee class was created by Pure Fabrication, and contains the instance of the WifiModem, instantiated in the Simulation class. All instances of the current Robots were also instantiated in AutoMail. Assigning the Robots after instantiating ChargeFee, rather than in the constructor, gives us the ability to update which robots we use to aggregate the AvgOperatingTime. It also allows for the possibility of retiring robots/putting robots out of action within AutoMail without having to re-instantiate ChargeFee to reflect this. This could be helpful if they decide to extend the functionality of the system in the future.

Unfortunately, we were unable to reduce the Coupling between Simulation, ChargeFee and WifiModem, which could lead to some issues if they were to change how the WifiModem is instantiated or how look ups are accessed. Although this would be quite a large design change, our use of Coupling in this instance should not prove to be an issue in future extensions of the system.

3 Summary

Overall, we were quite content with our design decision to implement new classes for relevant changes, as well as the extended capabilities of AutoMail. Our additions to the system maintained the High Cohesive nature of the original system, without adding too much bloating and minimal Coupling.

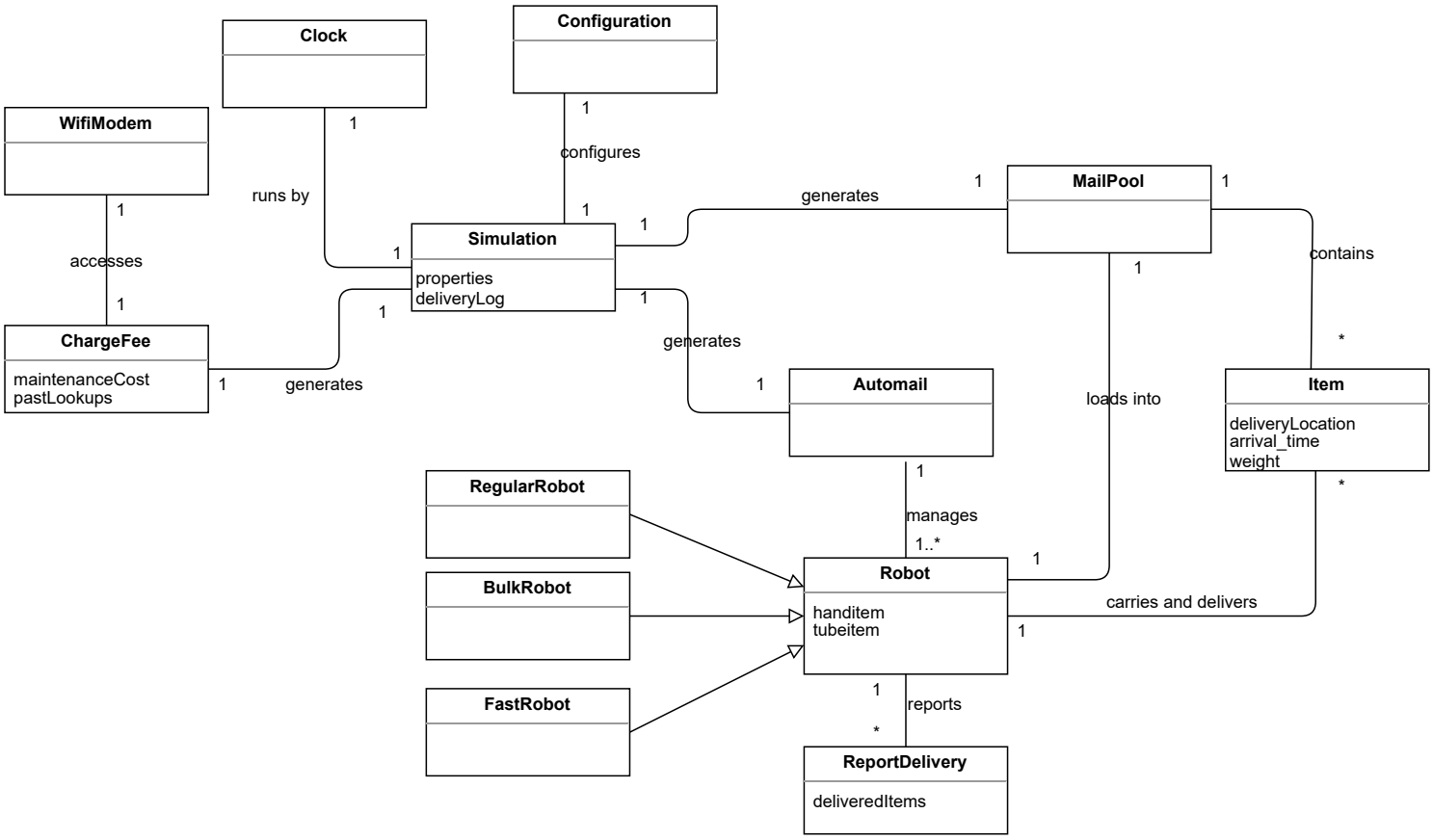

Figure 1: Domain Model Diagram

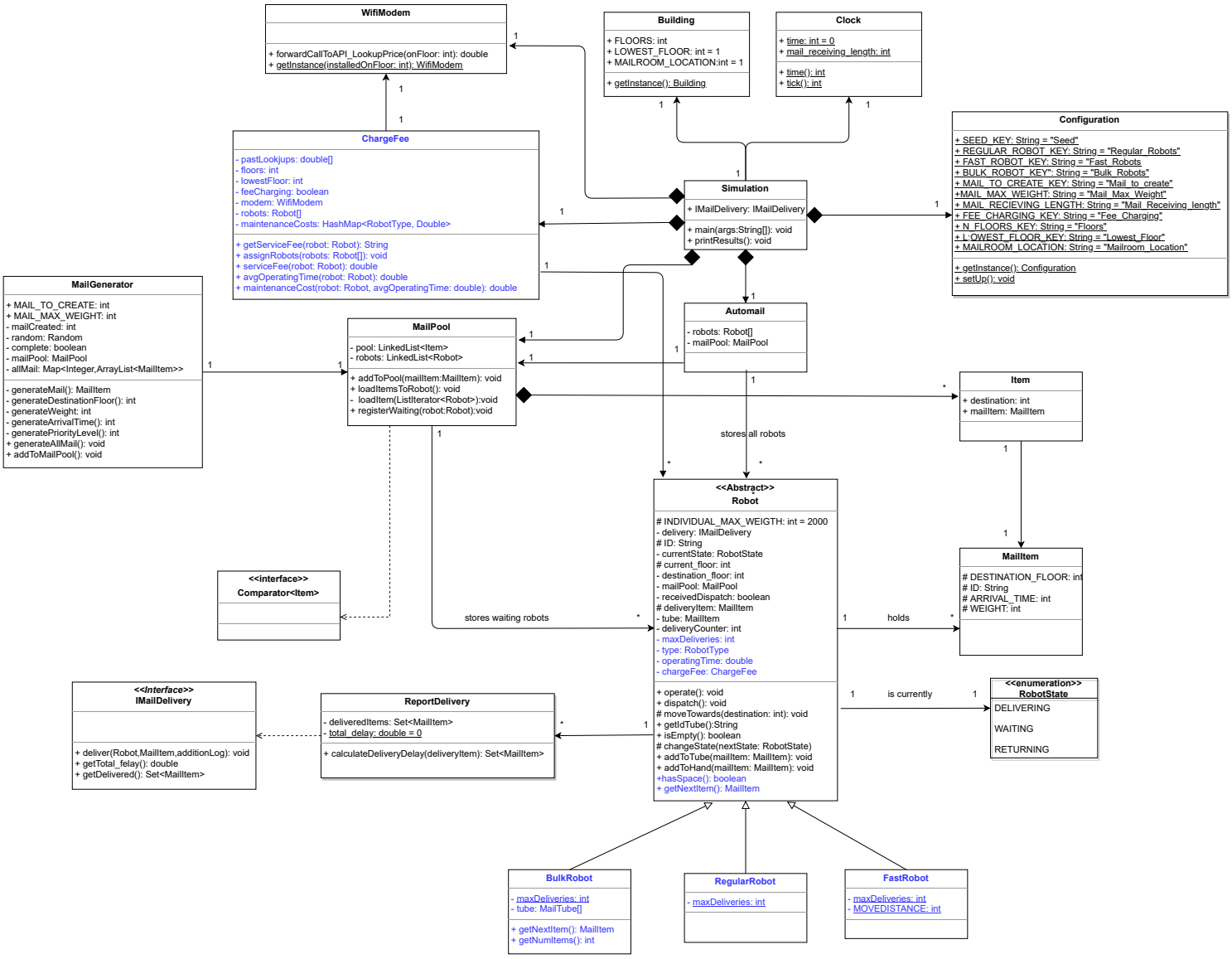

Figure 2: Design Class Diagram

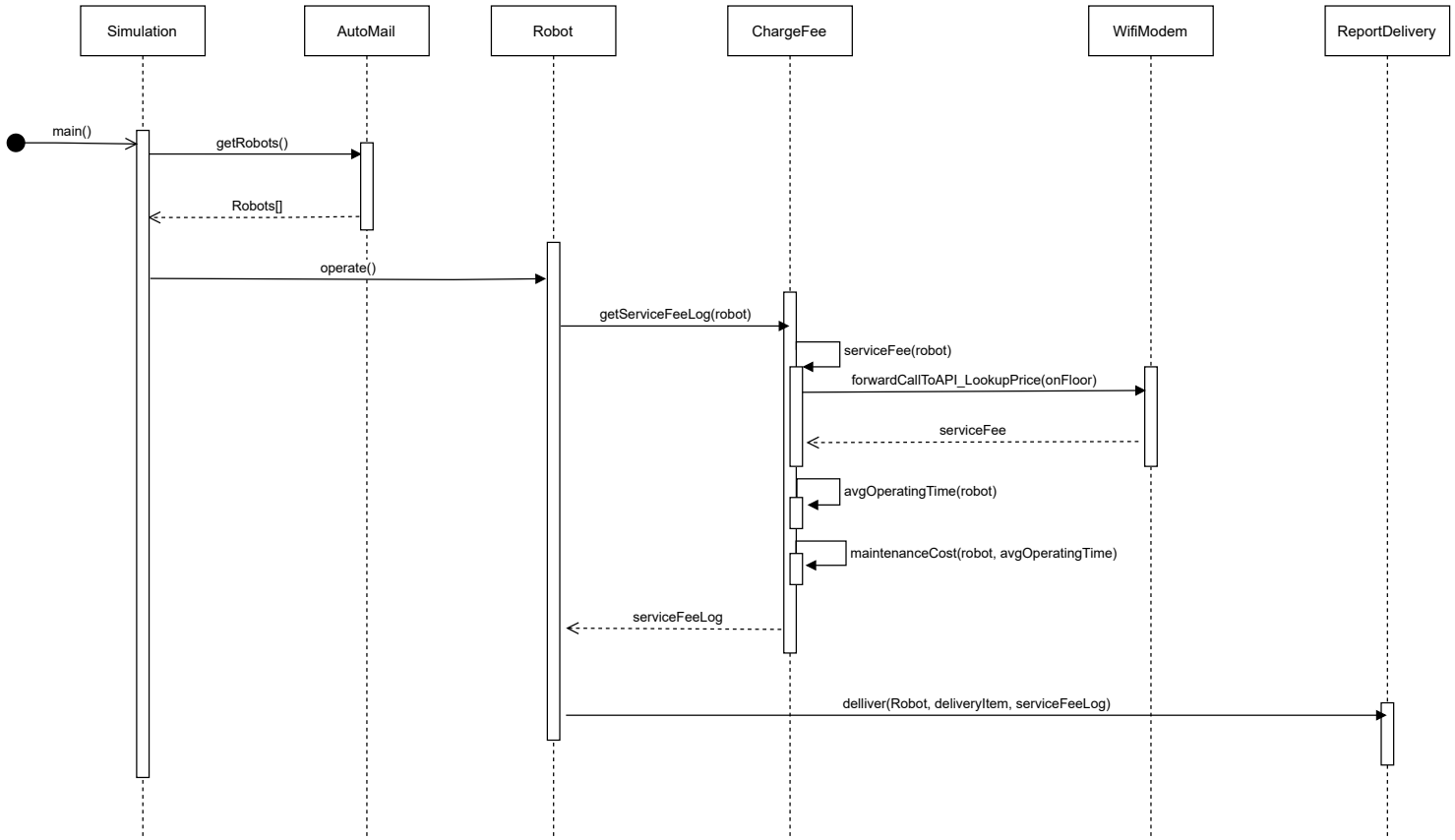

Figure 3: Design Sequence Diagram: Calculate a Total Charge