SWEN30006 Software Modelling and Design
Team 118
Author 1 (Aneesh Chattaraj), Author 2 (Ian Wen Jing Teh), Author 3 (Alice)

1 Introduction

The task of this assignment was to update the simulation software of the robomail management system to incorporate a special case where a robot gets a pair of special hands when the caution mode is activated which should be used to deliver fragile items. This would also include a few extra functionalities to work with the special hands when caution mode is turned on. To improve the design, we applied GRASP patterns, the one we mainly focused on was to keep high cohesion and also lower coupling.

2 Solution Design

2.1 Code Implementation

- **Robot class**

Major changes were made in the Robot class because the task of fragile item delivery seemed apt to be carried out by this class. The Robot class has a more cohesive implementation since it focuses on a few tasks like moving between floors, delivering normal items (switch delivering) or fragile items(switch caution). It is important to note that our robots make it their priority to deliver fragile items first even if the item in its normal arms has a closer destination floor from floor 1.

- **MovementController class**

This class's primary focus is to control the movement of the robot depending on the availability of the next floor and utilises the Controller pattern. This was separated from the robot class to lower coupling between AutoMail and Robot class. The strategy we went with was to check the next floor of the current robot and its availability by if another robot is occupying it and if it is finishing delivery of a fragile item. If two robots want to go to the same floor the one with the fragile item would be allowed to move first and complete its delivery.

- **Mail Pool class**

A bit of unavoidable coupling exists to check if caution mode is turned on or not with the help of the CautionMode class. I think the red statement is fine as it's just a note of unavoidable coupling. The class just checks if created Robots are available to delivery and also contains the pool of Mail Items to be delivered.

- **Statistics class**

This class is the information expert. It has all the information regarding the packages delivered and time taken to deliver, wrap and unwrap packages and contains a method to print out the statistics. By making the statistics an individual class, it allows simulation, the class we originally planned to hold the statistics, more cohesive as it does not need to do anything related to statistics

- **Simulation and Automail**

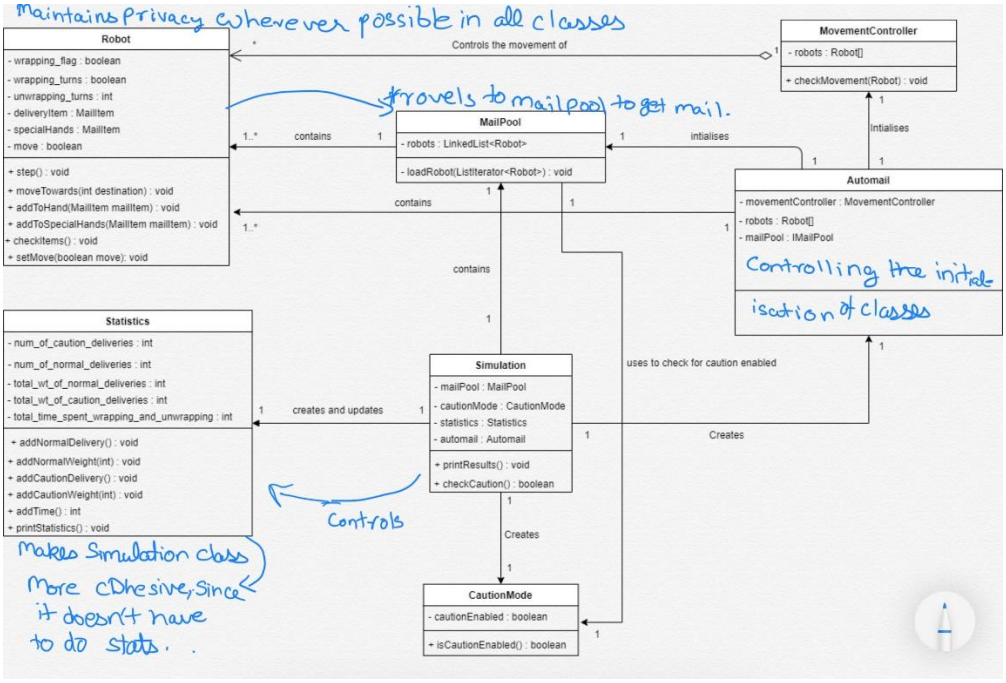
The simulation class has the responsibility of creating the different classes and the automail class acts as a controller since it contains the robots and also initialises the class, Movement Controller, which controls and manages every robot's movement. There is unavoidable coupling among the classes Simulation and MailPool due to the makeup of the Simulation class, as it uses the step () methods from the 2 classes stated previously.

- **CautionMode**

This class was created so that the cohesion of the Simulation class can be increased and also decrease coupling between Mailpool and Simulation without affecting its privacy. The coupling is lowered as Mailpool doesn't need to access Simulation's variables in order to check if caution mode has been enabled.

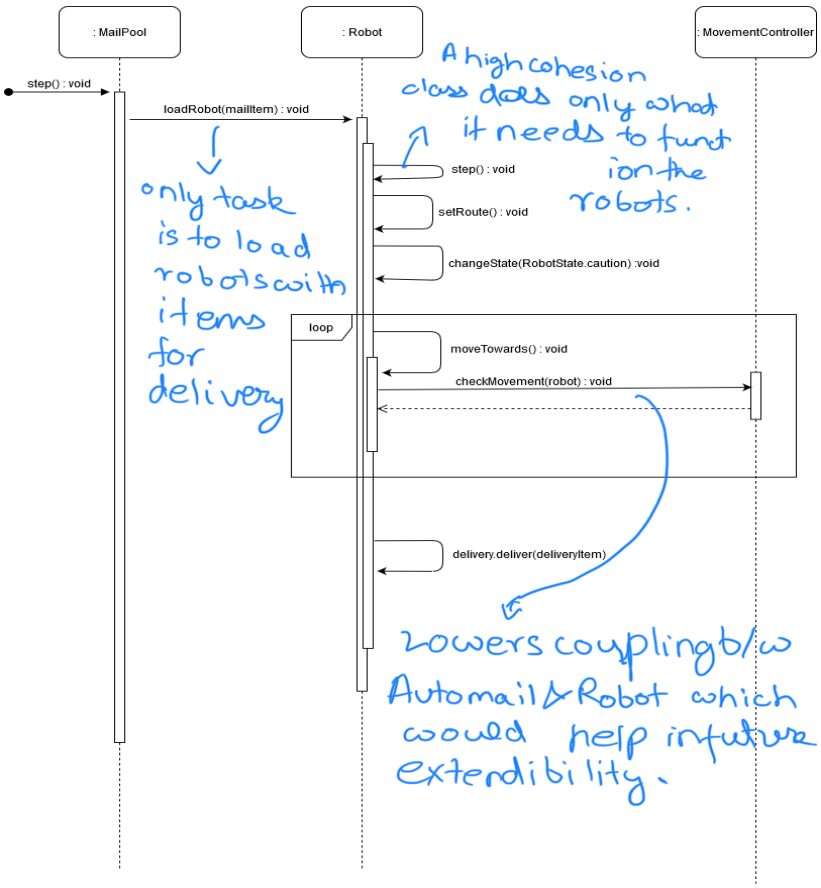
2.2 Static design model (design class diagram)

For the design class diagram, we only included the classes that had their implementations changed and also the newly added classes from the original version (Statistics class, CautionMode class and Movement Controller class). Comparing it to the domain model we could see that there are a few changes to classes and also additional classes were added to increase cohesion. The domain model helped us to understand where to begin our implementation.



2.3 Sequence Design Diagram

The system sequence design describes the main functionalities from receiving items from that mailpool to delivering them via robots. At the mailpool, once mail items are generated mailpool.step() is used to load any robots if present with mail items. The Robot class then checks its state and then sets the next destination or floor to travel to. It completes the task and then changes state again and repeats the process. The MovementController decides if the robot is allowed to move to the next floor towards the destination or not depending on the availability of the floor. We designed our diagram in such a way to only focus on the movement and transition states of the robot.



3 Alternative Solutions

There were a few alternatives which we thought of working with. We considered dividing the Robot class into classes to work with or without caution state. This would have helped us achieve a polymorphic pattern which could have been used for extendibility in a future version if required but we decided not to since all the tasks could be completed in the same Robot class while maintaining lower coupling and higher cohesion since focused on just delivering and travelling between floors. That is why our step() method seems long but it consists of all the required information to deliver items in the Robot class, maintaining high cohesion.

In a different version of our code, we made changes to the mailpool loading system and instead of adding an item to hand first we added it to the tube and then added the next item to the hand if any. We made a few interesting observations regarding the delivery time and final score, it slightly performed better in cases when the seed was changed and, in some cases, slightly worse. Although we went with the previous version since it seemed to work well.