SWEN30006 Software Modelling and Design Project 1
Robomail Revision
Report

In the original version of RMS design and implementation, mailpool is responsible for assigning the mail item to robot's hand and tube, then sends a dispatch command to send off robots to deliver mails across the building. To support the delivering of heavy mail items, our team has come up with an idea to change how mailpool assigns a heavy item to a group of robots.

We have noticed that the mailpool might be doing too many jobs, because it does not only add mail to itself, it also does the job of assigning the mail to the robots. A better approach could be separating the distribution job from the mailpool and create a new distribution system to handle this task, such that we can achieve a more sophisticated design for our new system.

After this separation of jobs, mailpool will only focus on its role as a temporal mail storage facility. If there is any further change to the mail delivery strategy in the future, we can simply replace the distribution system without affecting the functionality of mailpool. Objects only do jobs that belongs to themselves and do not do jobs that should belong to others. This improvement has made our system to have higher cohesion since now different objects can focus on their own responsibilities. Besides, when we want to change the implementation of mail pool or distribution system, it is not necessary for us to change other classes. This low change impact supports low coupling since objects are more independent and do not rely on other objects.

IdistributeSystem is an interface that builds based on the Strategy pattern, future changes to the delivery strategy can easily implement this interface and override the distribute() function to have a customized mail delivery strategy.

SimpleDistributeSystem and WeightDistributeSystem are 2 available distribution system in the new RMS. One is a simple version which has the same mail distribution and delivers logic that inherits from the old RMS design. The WeightDistributeSystem is our new delivery strategy that is capable to handle heavy mail item. We used polymorphism to achieve this. IdistributeSystem provides the basic function prototype for the distribute system. SimpleDistributeSystem and WeightDistributeSystem uses the same prototype

but alternative implementation of distribute() function. Polymorphism makes our system to be more extendable and flexible.

Our team has also used the factory pattern. The task to create distribution systems is assigned to the DistributeSystemFactory. There is a desire to separate the creation responsibility of distribution system for better cohesion. The distribution system is designed to help mail pool to load mails to robots according to different specification (e.g. load all robots identically if all mails are light; load heavy mail to a team if there are presents of heavy mail). More variations of loading might be needed in the future market. By centralizing the creation responsibility, we can easily support more variation through adding more cases in the Factory, without affecting other related classes. And we can always get different distribution system from factory, by only changing the implementation of Factory.

Factory needs a global and single point of access whenever we need the creation of objects from the factory. One instance of factory is sufficient to create the required distribution system. In order to make sure that there is only one instance of distribution system factory, we used the singleton pattern and has avoided complexity.

In the WeightDistributeSystem.java, when we were writing the function to load the item to the robot, we created a helper class with all the static method to help loading mail item to robots. Therefore, if there is a need in the future to build a more complicated distribution system, those static methods can be accessed from the helper class without the creation of extra objects.

Our team has also moved the dispatch() function from Robot to distribute systems, different distribute system may require different dispatch strategies. It is the responsibility for each distribution system to dispatch a robot after loading mail items on to it, but not robot to dispatch itself. This change helps to improve the cohesion in the system. Robot and distribute system class is now more focused on their own tasks.

To control the moving speed of a robot's team, the easiest way is to add a variable to the robot class to indicate the differences in speed. However, such approach will introduce more variable and a lot of messy functions in the class, which is not an extendable design.

Hence we used another strategy to solve this issue. Single robot and team robots have different moving speed, but eventually, they are performing the same action, which is moving towards the destination floor. Using a strategy pattern, a new interface called IBehavior to allow different variation in the moveTowards() function. When we need robots to move, we can just call moveTowards() from the strategy(IBehavior) without worrying about what exactly are the specific movement.

After the adding of strategy(IBehavior), we need some extra information about the robot, in order to control its movement. So, we use Information expert pattern to create a separate class to store extra information for the Robot, called Info. By adding this class, we can put information that will be used for robot behavior into it. This has improved clarity when we need access to behavior related information.

After taking care of both distribution and robot behavior. We need to take care of the Delivery, according to the original model, if we have a team of three robots to deliver the mail, it will produce the exception that said the mail is already delivered. So, using the polymorphism pattern, we create a TeamReportDelivery with extends from the basis ReportDelivery and add a counter for the heavy mail, so the system will not report this “problem” again. In this way, we have extended the delivery without changing the original performance.

After refactoring of the original system and adding new functionalities, we have successfully implemented the team behavior to deal with heavier mail items. We have also preserved the original behavior of the existing system.