Software Modelling design report

In the current version of RMS design and implementation, mailpool is responsible to assigns the mail item to robot’s hand and tube, then a dispatch command is made by mailpool to send off robots to deliver mails across the building. To support the delivering of a heavier mail item, our team have come up with an idea that to change how mailpool will assign a heavy item to a group of robots.

We have noticed that the mailpool might be doing too many jobs, because it not only added mail to itself, it also does the job of distributing the mail to the robots. A better approach could be to separate the distribution job from the mailpool, and assign it to a distribution system, such that we can achieve high cohesion in our new system.

After this separation of logic, 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.

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 custom 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.

Our team has used the Singleton pattern here as there might be multiple distribution strategies available in the future, and run time change to the strategy may be required to adjust the system efficiency. Therefore, a DistributeSystemFactory is created to obtain the single instance of the distribution system.

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 also have moved the dispatch() function from Robot to a different distributeSystem, be different distributeSystem may require different dispatch strategies. It is also the responsibility of each distribution system to dispatch a robot after loading mail item on to it, but not robots to dispatch itself.

To control the moving speed of robot’s team, the easiest way is to add a variable to the robot class for indicate the differences in speed.

to tell if the robot in a team and then control it’s movement by that. However, this will introduce more variable and a lot of messy function in the class and not extendable. So, according the indirection principle, using another strategy pattern, create a new interface called IBehavior, and the distributeSystem dispatch the robot to control the behavior of the robot, and the moveforward() function of the robot is now moved to the interface as they in different behavior, the movement will be different. There are only 2 different behavior in the scenario: TeamBehaviour and SoloBehaviour. But people can added more in the future.

During the process of using indirection pattern with the Interface Behaviour. We need some information about the robot, in order to control its movement. So, we create an Information expert for the Robot, called Info, to record some information of the robot and then share to the Behaviour to use it.

After taking care of both distribution and the 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 deliver. So, using the polymorphism pattern, we create a TeamReportDelivery with extends from the basis ReportDelivery and add a counter of mail for the heavy mail, so It won’t be showed as excess delivery again.