Design document

Igus robot arm project

Rik van Heesewijk, Luuk Aarts, Dimitar Dyulgeruv, Jorn Kersten

6-16-2023

Table of Contents

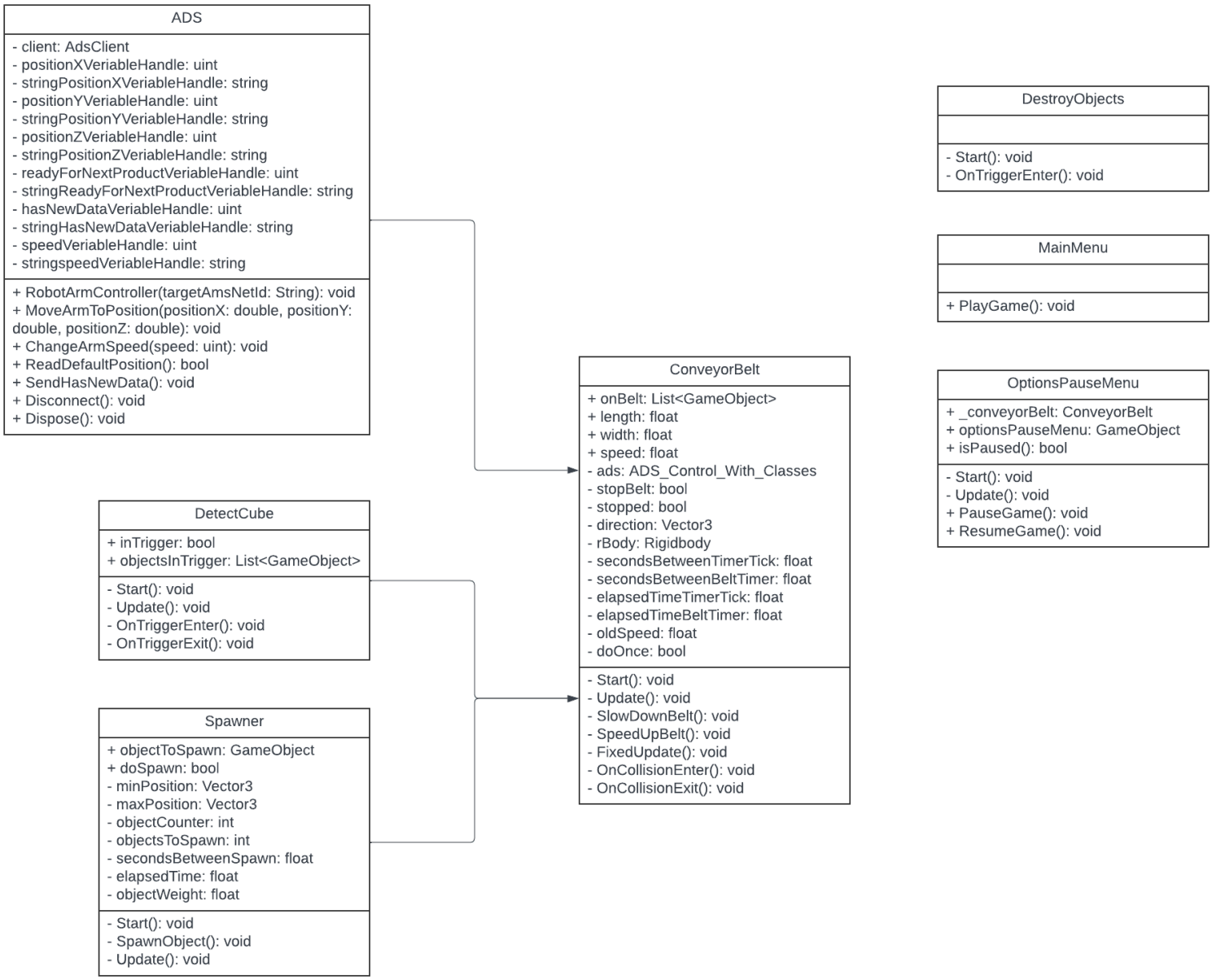
Introduction

In this document you will find all created designs relevant to our project. One Class diagram was created for Unity, in addition there is also a section on why the objects in Unity are laid out and designed this way.
For TwinCAT, there are several state diagrams and a class diagram. Both will be reviewed and explained as well.

Unity

This chapter explains how everything communicates with each other and why the scenes are laid out the way they are. First you see the class diagram, there follows a brief explanation of what talks to each other and what is used next.

Class diagram

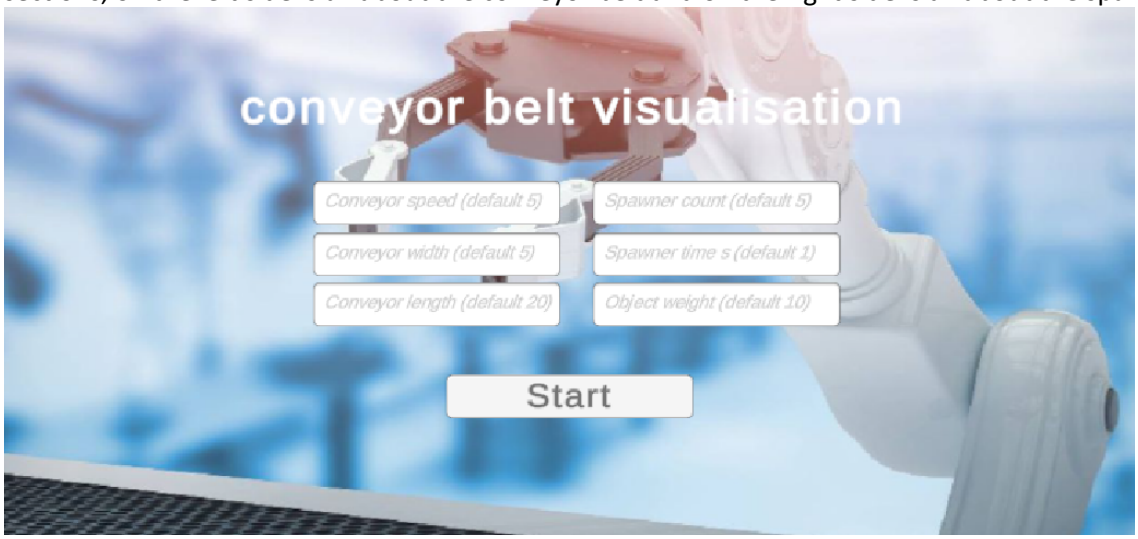


Above is the class diagram for Unity, Unity's classes are: ADS, ConveyorBelt, DetectCube, Spawner, DestroyObjects, MainMenu and OptionsPauseMenu. There are 4 classes connected to each other,

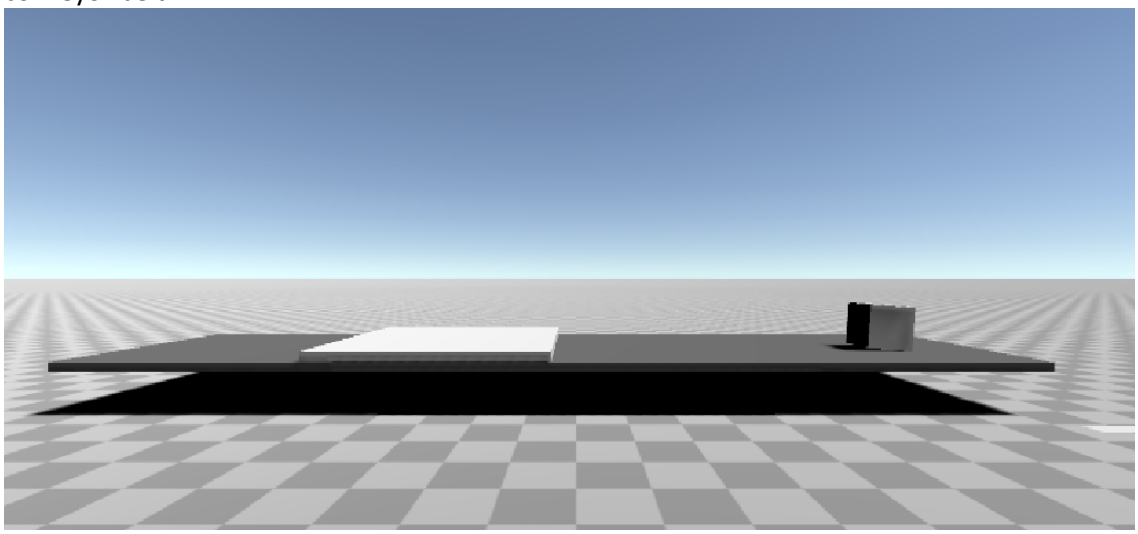
the only one of those four that calls other classes is the ConveyorBelt class. The ConveyorBelt class reads some variables and methods from the other 3 classes (ADS, DetectCube and spawner). From ADS a number of methods are called, RobotArmContorler, MoveArmToPosition, ReadDefaultPosition and SendHasNewData with these methods a number of variables are read from the PLC and also variables are set from the PLC. From DetectCube, inTrigger are read to check if there is a cube in the detection area. In addition, objectsInTrigger is also read to read the location of the first cube that is in the trigger. Finally, the variable maxZdetectonArea is read to pass the correct position of the cube relative to the detection area to the robot arm. From Spawner, the ConveyorBelt class sets the boolean doSpawn to true or false so that no more cubes are spawned when the conveyor belt is stopped.

Scene layout

The scenes are divided into two parts, the main menu section and the game scene. In the main menu scene, a menu was created with a number of input boxes on it. These input boxes are split into two sections, on the left side is all about the conveyor belt and on the right side is all about the spawner.



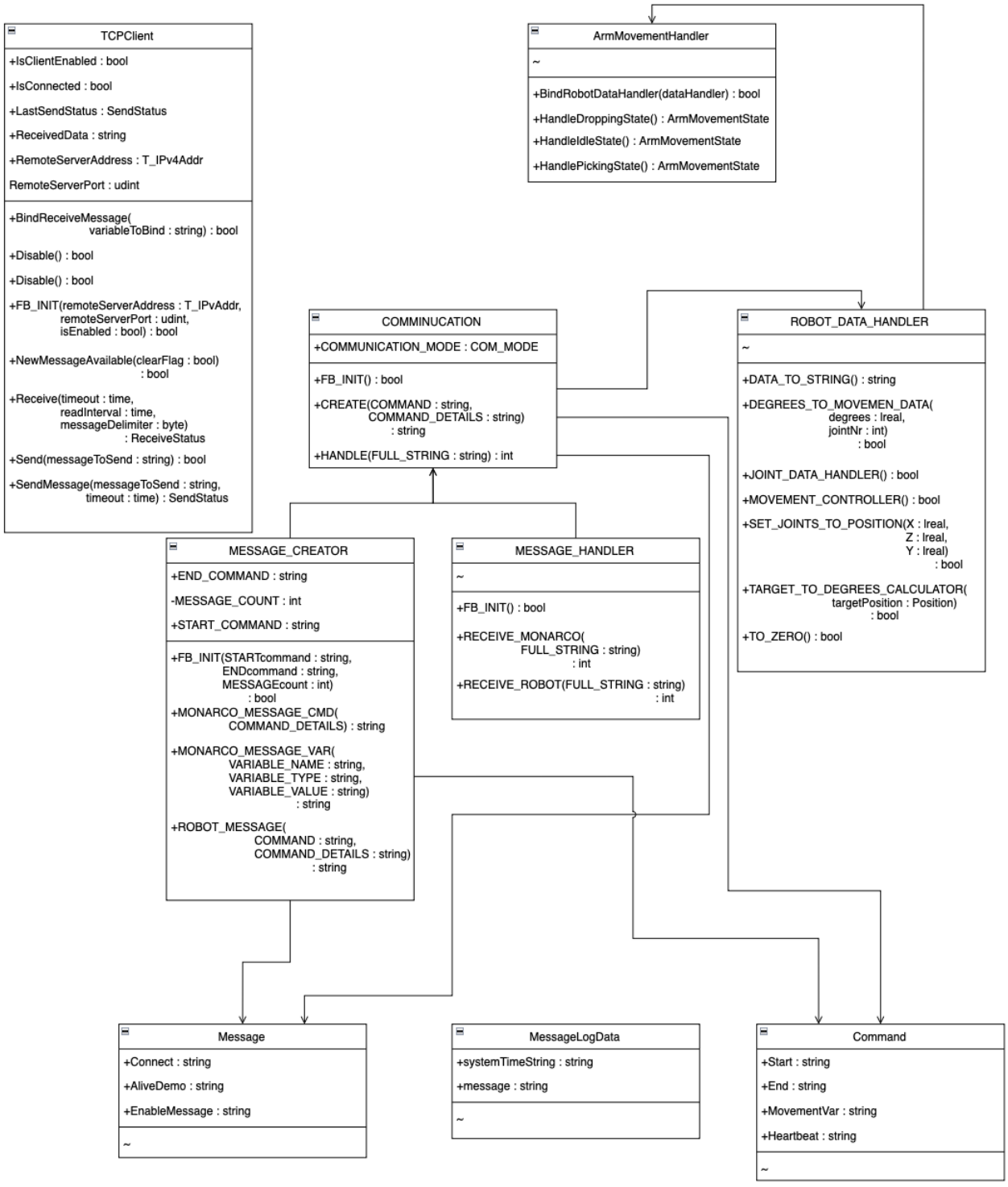
When start is clicked, the game scene is loaded. A deliberate choice was made here to spawn cubes, this is a simple object that can be moved without rolling or making strange movements. The conveyor belt has been kept simple so that the full focus can be on its functionality. In addition, a colour has been given to the conveyor belt so that it is clearly different from the rest of the scene. The terrain was also deliberately left simple, it is only there so that it looks a little neater as a floating conveyor belt.



TwinCAT

We will explain the layout of the OOP based design features used in the BECKHOFF PLC robot arm communication software. First you will see the class diagram including an explanation on the purpose of all classes and their coherence together, second will come the state diagrams, these will show the cycle of action used to run the program.

Class diagram



Explanation

All classes work together to form one machine that can send, receive, handle, time and create the messages that the server is supposed to receive or has send.

TCPClient

The TCPClient class will control the complete TCP IP connection with the server that it is trying to connect with. In the case of the robot arm, it will only set the ethernet connection and form the portal to send and receive messages. The handling, timing and creation of those messages will be handled by the other classes.

ArmMovementHandler

The ArmMovementHandler class will keep track of the current state that the arm is in and handle the communication with the Unity program on when the arm is available and when a package is handled. So the main State diagram is handled by this class but it uses the ROBOT_DATA_HANDLER class to get and set the information about the robot arm.

ROBOT_DATA_HANDLER

The ROBOT_DATA_HANDLER class decides what to do with the received data and what calculates what data should be send to for fill the task given by the ArmMovementHandler class. It can do so by calculating the joint angles out of the given XYZ coordinates, calculating the time and speed the robot must move to get to that position and controls the direct movement of the robot.

COMMUNICATION

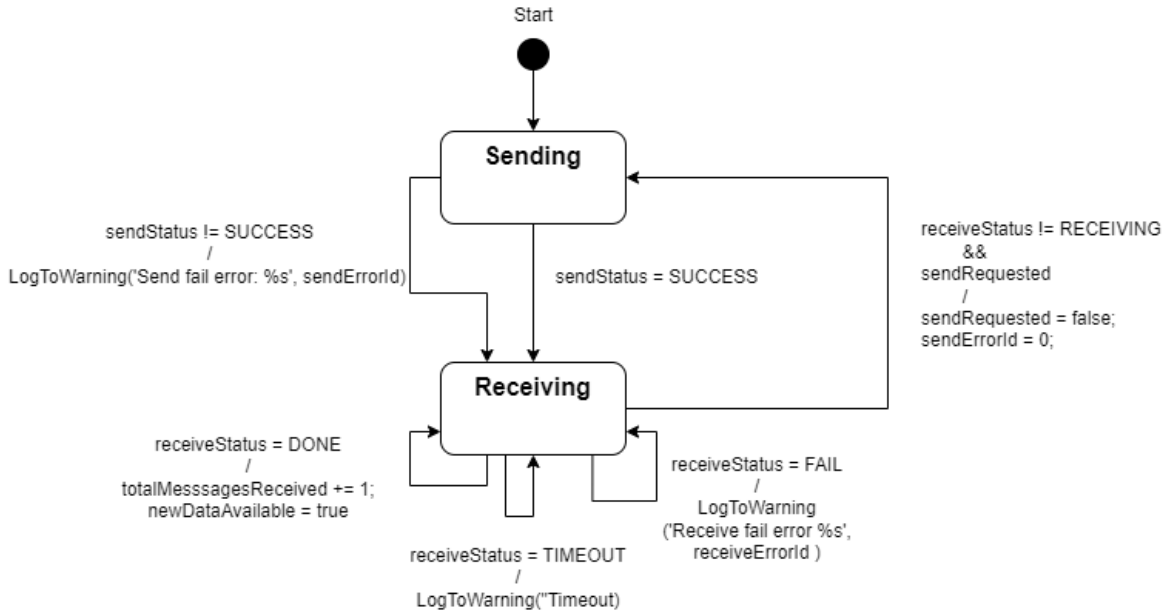
The COMMUNICATION class will turn the by the ROBOT_DATA_HANDLER class given data into a string the robot or any other server can read. It uses the MESSAGE_CREATOR to turn the given data into a string and keep track of the message counter, and the MESSAGE_HANDLER to turn the received string into usable data. Since this software can also connect with a MONARCO pi, both the creator and handler can do so for both the robot and the pi. By setting the mode, the COMMUNICATION class will trigger the correct method in the classes.

Review

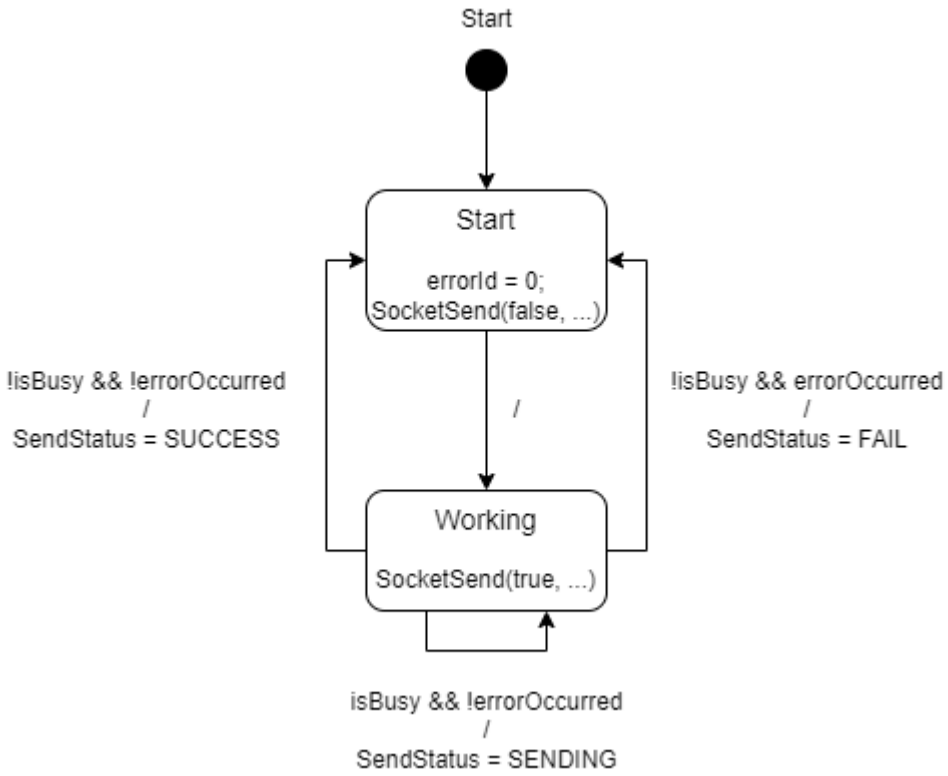
Although this design is most definitely not perfect, it does work and is easily understandable and adjustable. For the short amount of time, that where the main goals in our design for now, readability and adjustability

State diagram

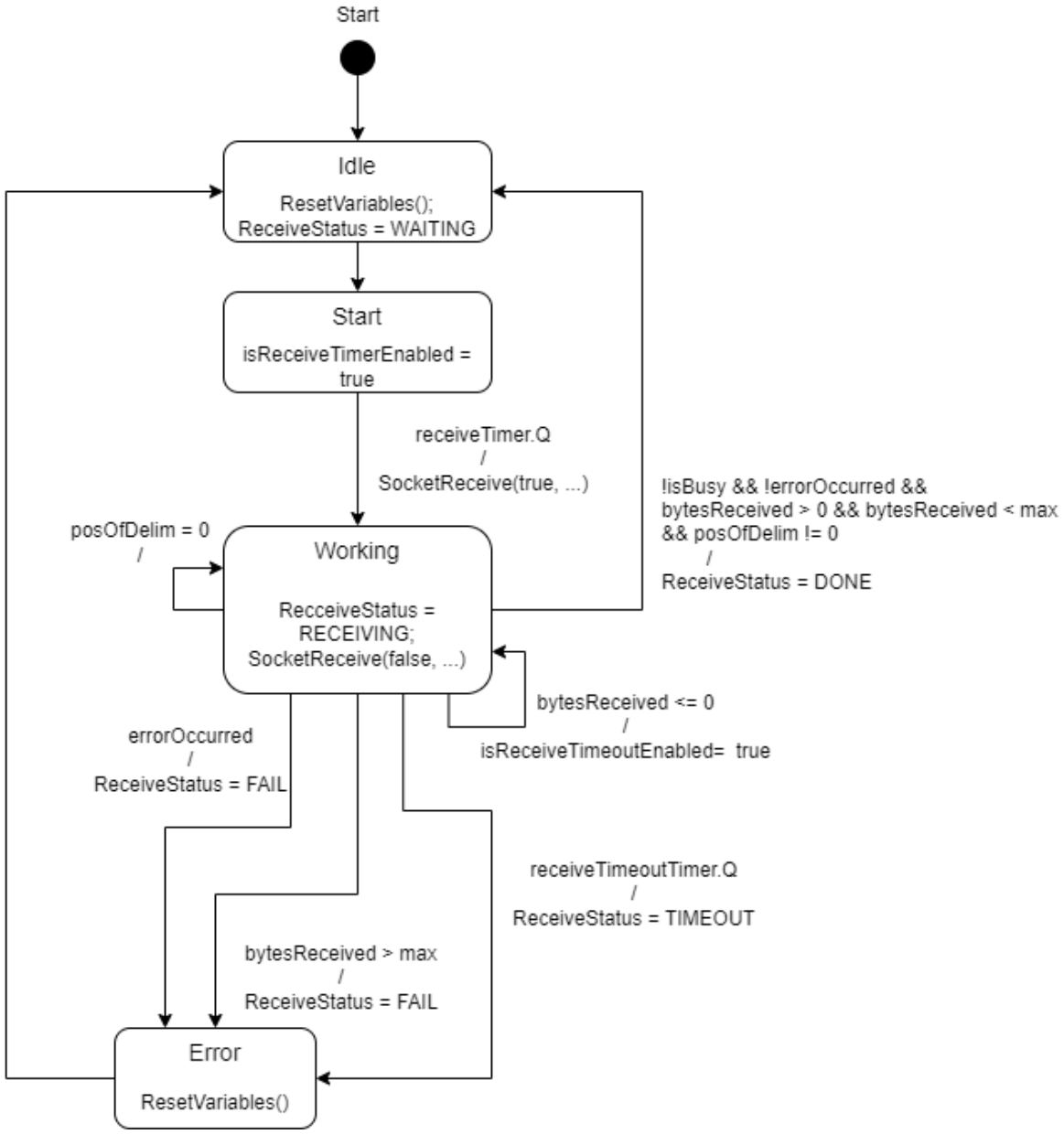
TCPClient State Machine:



Sending Message State Machine:



Receiving Message State Machine:



Pick-and-Place State Machine: