# **Report**

The project is split in 3 main fields which are connected through an mqtt-broker. These 3 fields are the webbots visualization of the robots, the server which calculates all the logic and a website user interface.

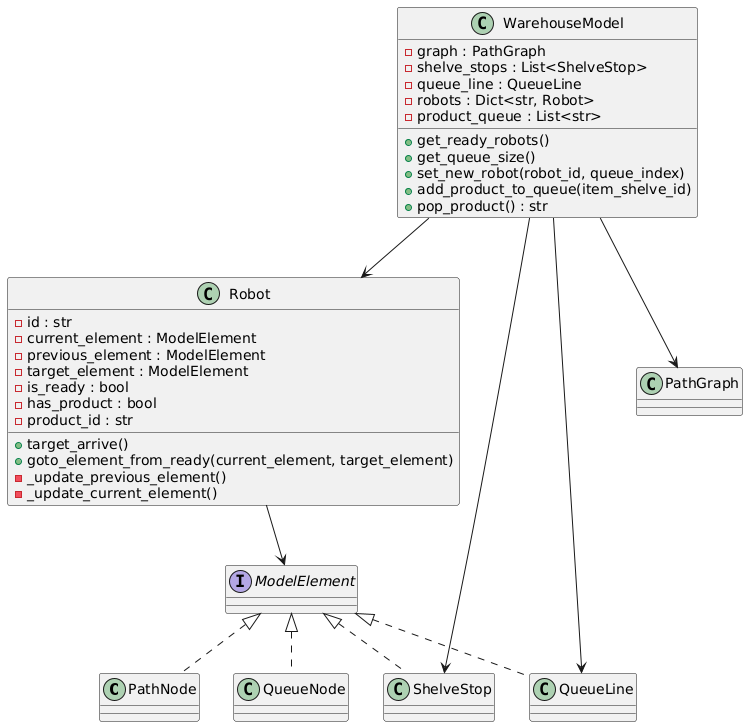
Server:

Figure 1 represents the models that the server initiates and their connections to each other. The WarehouseModel has all the necessary information to guide the Robot instances to their target place in the premade graph.

The scheduling system (figure 2) coordinates the movements of robots in a warehouse, with various components working together:

1. **Scheduler:** Orchestrates assignments to robots with integration of the robot commander. It tracks which robots are ready for their next task, and assigns them and resolves conflicts.
2. **QueueManager:** Manages robots in the queue, moves them to free positions, and ensures robots drop off products when needed.
3. **TaskManager:** Uses the path planner and product queue to break every robot mission down into atomic assignments.
4. **PathPlanner:** Calculates the shortest paths for robots to move from one location to another within the warehouse.
5. **ReservationManager:** Handles reservation of path nodes. Robots can reserve nodes so that others cannot cross this path until a reservation is released. This ensures no collisions. Conflicts however can happen, such as deadlocks and livelocks. These are handled by the scheduler.

Figure 1: UML diagram server model

1. **ChainPathManager**: Groups path sections that are unbranched. Unbranched sections could trap robots unless robots could backtrack. The reservation manager uses this to reserve entire chains in the path atomically. This avoids that multiple robots enter linear paths such as corridors.
2. **RobotCommander:** Hosts the MQTT client. Sends commands to robots to move, pick up or drop off products, and receives updates on their status.

Together, these components ensure efficient and smooth robot operation in the warehouse, preventing delays and collisions.

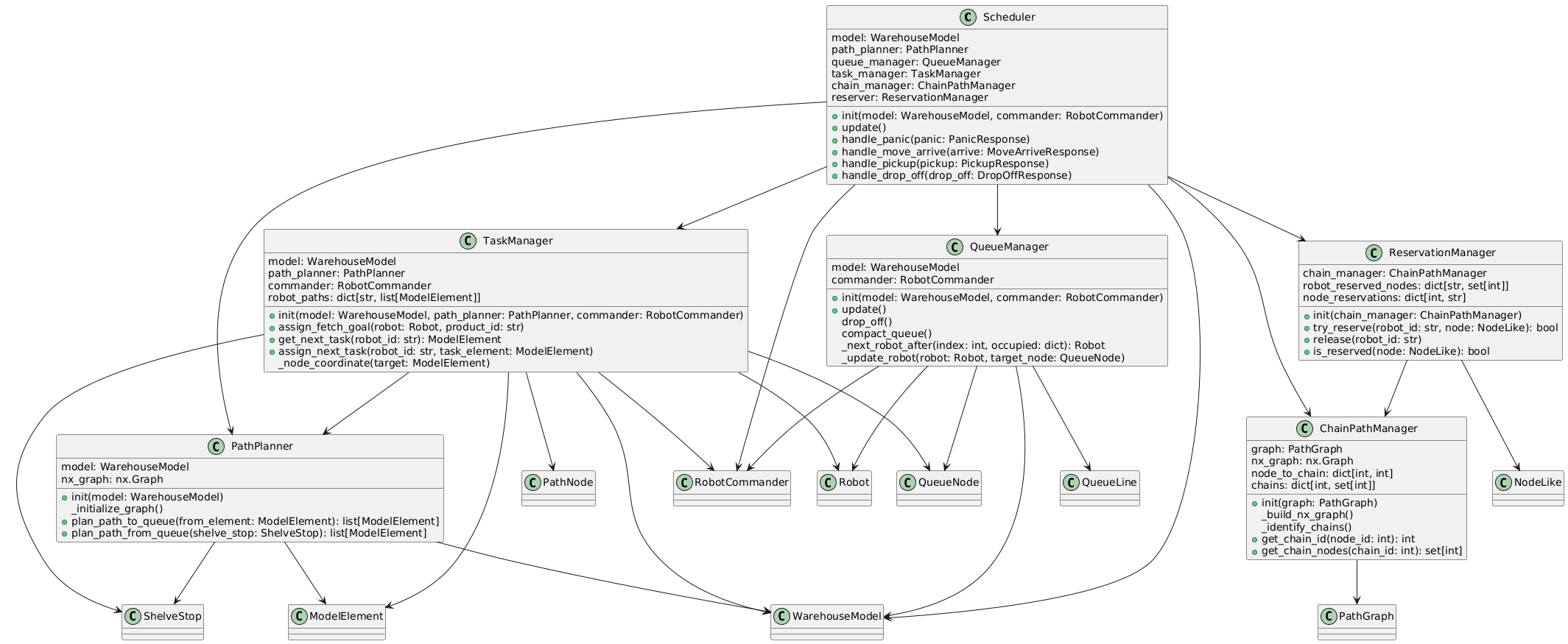


Figure 2: UML diagram scheduling algorithm

Resolving conflicts and avoiding collision:

There are two mechanisms applied. To avoid collision, a reservation system is used. Robots will always reserve their target node. Only when a robot has left a node entirely (by arriving at the next node after the path-graph edge), that node will get released. This means that the current edge of the robot is always enclosed in reserved nodes. When other robots try to cross this path, their reservation will fail because another robot still has ownership over the node. This ensures that there is always at least one edge of distance between two robots.  
When a reservation fails, the next robot task will simply be ignored, keeping the robot idle. Whenever the schedular updates and the reserved node is released, the robot continues.  
However waiting can lead to mutual locks: two robots wait on each other. To prevent this, a dodging mechanism is applied. Given multiple parameters, a priority is chosen to assign the robots into a dodging robot and a passing robot. The priority will make sure that of one of the robots stands on a node of degree 2 or lower (degree = number of neighbors), the dodging robot will always be the robot with a degree of 3 or higher. The dodging robot will observe the path of the passing robot. Eliminating the conflicting node and the next target node of the passing robot, a node is chosen that is free from any path. The dodging robot can then move into this position and let the passing robot pass its original path. After this, both robots can continue.  
in the figure below, the node with robot A has a degree of 2. Therefore B is the dodging robot. (if both have a high degree, other parameters are prioritized, using robot id as eventual tiebreaker). The conflicting node is the green node. One robot has reserved this node and the other could not enter it. In case of a conflict, B will actually move into the conflict node tough. B will scan the neigbors of this node. Eliminating the current node of A, and target node of A, the yellow node remains. B being at the green node during the conflict, will dodge to the yellow node, letting A pass. After the confict, the reservation of the green node is cleared and B can continue its path.

One final addition is the chain path manager.  
When two robots are trapped in a linear path segment (for example a long corridor),  
then it is impossible to dodge (or replan), since all bots stand on nodes of  
degree <= 2. To avoid this situation, ‘chain paths’ (linear segments), are grouped  
to be reserved atomically. This way when a chain is entered, the entire chain is  
reserved, preventing two robots in the same chain.

Afbeelding met tekst, schermopname

Door AI gegenereerde inhoud is mogelijk onjuist.The connection robot <-> server is completely managed by the RobotCommander class. This class connects to the mqtt-broker, subscribes to the relevant topics to communicate with the robots and sends the correct messages through the correct topics to give commands to the robots.

The added code shows the different commands the RobotCommander can send to the robots and what kind of information every command contains. Every command has a different topic to prevent intervention between different instructions. All commands are processed with the from\_dict() and to\_dict() functions make the dataclasses more compatible with other procedures.

Afbeelding met tekst, schermopname

Door AI gegenereerde inhoud is mogelijk onjuist.

Afbeelding met Schaalmodel, schermopname

Door AI gegenereerde inhoud is mogelijk onjuist.Webots:

The depot made in webots has a few key features which contribute to the correct functioning of the robots. The small black dots on the ground are placed on the nodes of the graph. These will be used for correction if the robot stops on them, after driving to a node, the robot will check if it has a black dot right beneath itself using visual computing. If not, the robot corrects itself until the dot is right beneath it. By doing this, the robot removes slight differences and eventually prevents a significant error.

The robot consist of a main platform (green square) with 4 omniwheels, this design was made by MasdikaAliman (<https://github.com/MasdikaAliman/Kinematic-Omniwheels-in-Webots>). A lidar sensor was added on top of the robot which detect the nearest object and prevents collisions. A color sensor was added beneath the robot to detect the black dots used for correction.

Afbeelding met schermopname, rood, Karmijn

Door AI gegenereerde inhoud is mogelijk onjuist.

Afbeelding met tekst, schermopname, software

Door AI gegenereerde inhoud is mogelijk onjuist.The robot controllers connect to the mqtt-broker with the relevant topics so they can receive commands given by the RobotCommander. The movements by the omniwheels are calculated using a special kinematic file made by MasdikaAliman. Using the omniwheels, the robot does not need to turn around which removes the risks of turning short or long. In other words, the robot can drive straight ahead, sideways and even diagonally.

All path nodes are marked by a black dot (assume a sticker on the floor). Given the imperfections of the robot, the dots can be used to correct any deviations of the robot. The robot uses a camera to detect dots (when applicable) and then continues to center itself. When the robot does not detect a black dot underneath itself, a panic message is send to the broker so the server knows something is wrong. Same when another object is within 0,5m of the robot.

Afbeelding met cirkel, schermopname

Door AI gegenereerde inhoud is mogelijk onjuist.