System Architecture and Specifications

O. Can Görür*, Jacob Seibert, Malte Siemers, Marc Schmidt

1 System Architecture

A visualisation of our system architecture is provided in Figure 1 below. It shows the connections between the different system components that will be implemented as ROS nodes. The data show is represented by the arrows. Since the agents are the part of the system doing the most extensive computation work, their operations are presented there too.

The same architecture is visualized as ROS node graph using the ros package rqt_graph in Figure 2.

2 System Specification

2.1 Warehouse Management

The Warehouse Management has access to all the data, that is required to obtain a certain model of a warehouse. This static data is used to describe the physics of the system as well as to configure the different components of it as required.

The Warehouse Management is responsible for getting the whole system started with the predefined configurations and to stop the incoming of new tasks to shutdown the system if that is demanded.

i) Data

- Map: one map per simulation only
  - geometry of the environment
  - locations of input, output & storage
  - charging zones: rectangle locations in the x-y coordinate plane with preset charging rates
  - start / rest locations for robots: robots rest there if they are idle
- Package Config: contains as many different package configurations as different packages needed
  - $Package\_Type$: all packages of a type are considered as equal, so if a package is requested at the output the software system only needs to care about the type of that package
  - $Package\_Weight$: the weight of the package determines which robots are able to carry that package
  - $Package\_Geometry$: the geometry of the package determines which robots are able to carry that package
- Robot Config: contains as many robot configurations as different robot types needed Since we use only robots with differential drive the robots are capable of turning on spot so that we don't need to set a minimal turning radius in the robot configurations.
  - Max_Speed: the maximal velocity the robot is able to drive with
  - $Robot\_Radius$: minimal radius of a circle centered in the middle of the robot, such that the circle covers the robot extent

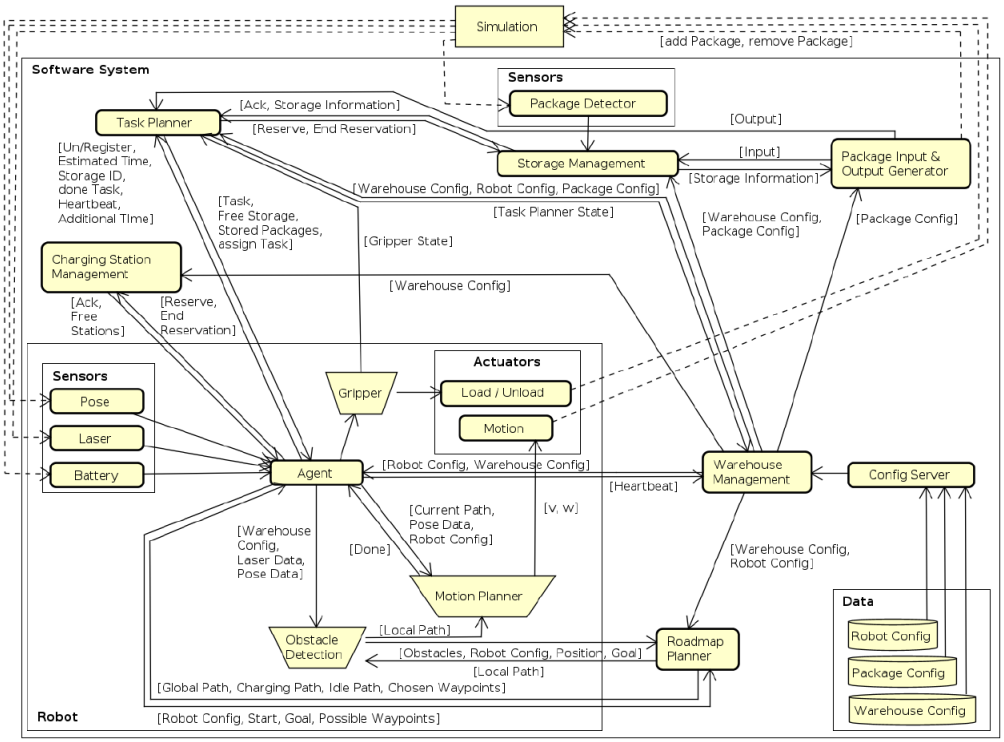

Figure 1: The overall System Architecture

  - Max_Carry_Weight: the maximal weight the robot is able to carry
  - $Carry\_Geometries$: a list of all the geometries the robot is able to carry
  - $Discharging\_Rate$: battery discharging rate in percent per second
- Charging Config: contains one charging configuration per charging zone
  - Max_Number_Robots: The maximum number of robots that can be charged simultaneously.
- Roadmap: one roadmap per robot configuration
  - is generated while running time by the Roadmap Generator
  - a static set of roads the robot is able to drive o
  - is represented as a graph: the nodes are associated with the different waypoint positions on the map, the edges specify which waypoints are directly reachable (in a straight line) from a certain waypoint and the edge weights contain the estimated time a robot would take to drive from one of this edge defining nodes to the other.

ii) Service Requests

- Roadmap Generator
  - generateRoadmap($Map, Robot\_Config$) : Roadmap
    - at the beginning of the simulation the Warehouse Manager requests the Roadmap Generator to create one Roadmap per robot configuration
    - Roadmaps are required to start the agents because they contain the data to plan the path for the robot
- Package Input & Output Generator
  - start($Package\_Config$) : boolean

        * after all other components have been launched, finally the Package Input & Output Generator has to be started to simulate requests to store or retrieve packages
    - stop() : boolean
- Storage Management
    - start($Map, Package\_Config$) : boolean
- Task Planner
    - start($Map, Robot\_Config, Package\_Config$) : boolean
- Agent
    - start($Agent\_ID, Map, Robot\_Config, Roadmap$) : boolean
- Charging Station Management
    - start($Map, Charging\_Config$) : boolean

2.2 Roadmap Generator

The Roadmap Generator offers a service to create roads on a static map with respect to the physical constraints of the vehicles that are about to use this roads.

i) Service Offers

- generateRoadmap(Map, $Robot\_Config$) : Roadmap
    - generates a Roadmap with respect to the static obstacles the Map contains as well as the $Robot\_Config$ which determines which movement behaviour the specific kind of robot has.
    - the Map contains important locations the roads have to cover e.g. input / output & storage locations, charging zones, start / rest locations, . . .

2.3 Package Input and Output Generator

The Package Input & Output Generator simulates a source such that packages arrive at the input for storage in the warehouse as well as it simulates a sink where a specific type of package is demanded at some given output and the system shall provide that package there as soon as possible. The generation of requests shall be random.

i) Service Offers

- start($Package\_Config$) : boolean
    - hands over the package configurations
    - starts to simulate package sources as well as package sinks
    - requests storage information from the Storage Management
- stop() : boolean
    - stops to create new packages at the inputs as well as it stops to demand stored packages at the outputs (delivery points for logistics).

ii) Service Requests

- Storage Management
    - getStorageInformation() : $Storage\_Information$
        * directly after being started, the Package Input & Output Generator asks for the storage informations, to get the required informations for generating inputs & outputs.
    - newInput($Input\_ID, Package\_ID, Package\_Type$) : boolean

  - newOutput($Output\_ID, Package\_Type$) : boolean

iii) Topics Subscribed

- Storage Management
  - Storage Information
    * used to stay up to date with the storage informations
    * informations are published when the state of the warehouse storage changed
    * required to know where to add input and which output to request

2.4 Storage Management

The Storage Management has an overview of the overall storage of the warehouse and provides those information, performs administrative tasks like reservation of storage and informs the Taskplanner about incoming Tasks. Furthermore the Storage Management is able to detect if a package is stored at a certain storage location so that loading as well as unloading packages is doublechecked.

i) Service Offers

- start($Map, Package\_Config$) : boolean
  - hands over the map as well as the package configurations.
  - starts all other services and publishes the storage informations topic.
- getStorageInformation() : $Storage\_Information$
  - provides the storage informations of the whole warehouse.
  - useful at start because the storage information topic only publishes if the package detectors recognized some change.
- newInput($Input\_ID, Package\_ID, Package\_Type$) : boolean
  - a new package with a given id and type has been arrived at the input container with the given id.
  - saves the id and type of the package corresponding to the input container with the given id.
  - expects the package detection laser to recognize an incoming package at the input with the given id and marks that input as reserved.
- newOutput($Output\_ID, Package\_Type$) : boolean
  - a package of the given type is demanded at the output container with the given id.
  - saves the type of the package corresponding to the output container with the given id.
  - expects the package detection laser to recognize an incoming package at the output with the given id and marks that output as reserved.
- reserveStorage($Storage\_ID$) : boolean
  - questions to reserve the storage with the given id.
  - reserved storage is not marked as free storage.
  - returns if the storage is reserved.
- endReservation($Storage\_ID$) : boolean
  - questions to end reservation of the storage with the given id.
  - the storage is marked as free again.
  - returns if the reservation has been ended.

ii) Topics Published

- Storage Information Every time a package has been placed into or has been retrieved from a storage container and every time a storage container is reserved or not any longer reserved the Storage Informations are published.
  - $Free\_Input$: $Input\_ID$'s of the input containers that are empty
  - $Free\_Output$: $Output\_ID$'s of the output containers that are empty
  - $Free\_Storage$: $Storage\_ID$'s of the storage containers that are empty
  - $Stored\_Packages$: information about where and which packages are stored in the warehouse e.g. $Storage\_ID, Package\_ID, Package\_Type$,. . .
  - $Full\_Output$: $Output\_ID$'s of the output containers that are full

iii) Service Requests

- Task Planner
  - storePackage($Input\_ID, Package\_ID, Package\_Type$) : boolean
    - after getting the new input request and detecting some package at the corresponding input the task is sent to the planner.
  - retrievePackage($Output\_ID, Package\_Type$) : boolean
    - after getting the new output request and making sure such a package type is stored in the warehouse already the task is sent to the planner.

iv) Topics Subscribed

- Package Detector Sensor: one per input tray, output tray & storage containers on the map
  - $Package\_Present$
    - detects whether a package is present at the container or not.
    - could be handled by collision detection or laser detection.

2.5 Task Planner

The Task Planner receives tasks, decides which kind of robot is capable to fulfill those, requests how much time it would take for the suitable robots to accomplish the task, decides which robot is the most available one and assigns the task to it.

i) Service Offers

- start($Robot\_Config, Package\_Config$) : boolean
  - hands over the map, the robot configurations and the package configurations.
  - starts all other services.
- storePackage($Input\_ID, Package\_ID, Package\_Type$) : boolean
  - a package with the given id and type arrived at the input with the given id and is ready to be stored in the warehouse.
  - chooses which of the registered robots are capable of fulfilling the task and sends the store request to them.
- retrievePackage($Output\_ID, Package\_Type$) : boolean
  - a package with a given type is demanded at the input with the given id.
  - chooses which of the registered robots are capable of fulfilling the task and sends the retrieve request to them.
- register($Agent\_ID$) : boolean
  - registers the agent with the given id.
  - registered agents receive task requests.

- unregister($Agent\_ID$) : boolean
  - unregisters an agent to avoid receiving task requests.
  - reserved storage is not marked as free storage.
  - returns if the storage is reserved.
- endReservation($Storage\_ID$) : boolean
  - questions to end reservation of the storage with the given id.
  - will be called if the battery of an Agent is below some minimum and it's not clear if the Charging Station is going to be reached before it's empty.
- completedTask($Task\_ID$) : boolean
  - called by an agent if the task with the given id has been fulfilled.
  - when the Storage Management provides the expected Storage Information the task is completed.

ii) Service Requests

- Agent
  - store($Task\_ID, Input\_ID, Free\_Storage$) : $Storage\_ID, Estimated\_Time$
  - retrieve($Task\_ID, Output\_ID, Stored\_Packages$) : $Storage\_ID, Estimated\_Time$
  - assignTask($Task\_ID$) : boolean
    - choose agent with the lowest estimated time for that task
    - before calling this tell the Storage Management must have reserved the corresponding storage.
- Storage Management
  - reserveStorage($Storage\_ID$) : boolean
    - requests to reserve storage if an agent asked to do so.
  - endReservation($Storage\_ID$) : boolean
    - requests to end reservation if an agent asked to do so or if the associated task is complete.

iii) Topics Subscribed

- Storage Management
  - Storage Information
    - needs storage informations to tell the agents which possibilities they have to store the package at or retrieve the package from.

2.6 Agent

The Agents are the executive components of the whole system. They receive requests to calculate how much time they would require to fulfill a specific task and have to take care of those tasks. Doing this the battery level is considered to include visiting charging zones which should be integrated into the calculation of the duration the task could be fulfilled in. The Agents have access to a pose, laser and battery sensor to get information about their location and their local environment as well as they have access to load / unload and motion actuators to interact with their environment.

i) Operations

- detectObstacles($Laser\_Data, Pose\_Data, Map$) : Obstacle
  - if the laser data doesn't match the data of the static obstacles on the map the unexpected obstacle is returned
  - an obstacle consists of a set of points in the x-y coordinate plane

  - no complex algorithm needed, just match the laser data with the static obstacles
  - returns NULL if no Obstacle detected
- planPathToStorage($Input\_ID, Free\_Storage, Pose\_Data, Roadmap, Battery\_Level) : Path, Storage\_ID, E$
  - calculates the shortest path on the roadmap that is required to store the package at the input with the given id.
  - considering the battery level it takes the visit of charging stations into account in the computation
  - chooses one of the empty storage containers to store the package at start point of the path is the destination of the last task in the taskqueue of the robot and the battery level is the estimated battery level at this location
  - shortest path finding with dijkstra
- planPathToDeliver($Output\_ID, Package\_Type, Stored\_Packages, Pose\_Data, Roadmap$) : $Path, Storage\_ID, Estimated\_Time$)
  - calculates the shortest path on the roadmap that is required to deliver a stored package of the given type to the output with the given id.
  - considering the battery level it takes the visit of charging stations into account in the computation.
  - chooses one of the containers that contain the given package type to retrieve the package from start point of the path is the destination of the last task in the taskqueue of the robot and the battery level is the estimated battery level at this location.
- planTrajectory($Map, Obstacle, Pose\_Data, Current\_Path, Robot\_Config$) : $Path, Estimated\_Time$)
  - is triggered when an obstacle is detected.
  - calculates the shortest path back to the current path considering the given obstacle, the map and the robot configuration.
  - may skip some of the waypoints of the current path but may not skip subgoals e.g. storage location to load / unload a package
  - start point of the returned path is the current location of the robot and the destination point of the returned path is some point of the current path
  - use the D* algorithm here
- planMotion($Current\_Path, Pose\_Data, Robot\_Configuration$) : boolean
  - adjusts the parameters of the motion actuator to drive along the given current path and takes into account the current pose data as well as the robot configuration.
  - tells motion actuator which linear and angular velocity to set
  - returns true if already at destination

ii) Service Offers

- start($Agent\_ID, Map, Robot\_Config, Roadmap$) : boolean
  - hands over the agent id, the map, the robot configuration and the roadmap associated to the robot configuration.
  - starts all other services.
  - calls the register service at the task planner.
- store($Task\_ID, Input\_ID, Free\_Storage$) : $Storage\_ID, Estimated\_Time$
  - request to plan the storing of a package which is located at the input with the given id.
  - returns the estimated time for this task in addition to the time for completing all other tasks in the taskqueue and returns the id of the storage the package should be stored at.
  - calls planPathToDeliver with the associated parameters.

  - saves task as well as calculation results temporarily.
- assignTask($Task\_ID$) : boolean
  - assigns the task with the given id to this agent
  - the task is placed at the last position of the taskqueue of the agent.
  - we know which task to fulfill because we saved it temporarily before.
  - calls reserveChargingZone

iii) Service Requests

- Charging Station Management
  - reserveChargingZone($Agent\_ID, ChargingZone\_ID$) : boolean
    - will be called after a task is assigned and it requires charging to fulfill the task
  - chargeRequest($Agent\_ID, ChargingZone\_ID$) : boolean
    - triggered a short time before the robot wants to charge
  - endReservation($Agent\_ID, ChargingZone\_ID$) : boolean
    - triggered after succesful charging or if charging plans change
- Motion Actuator - one per agent
  - stop() : boolean
    - stops the robot from moving
    - sets linear and angular velocity to 0
  - setSpeed($Linear\_Velo, Angular\_Velo$) : boolean
    - sets the linear and angular velocity of the motion actuator allowing the robot to move in the x-y coordinate plane
- Load / Unload Actuator - one per agent
  - load() : boolean
    - grips a package in front of the robot and attach it to itself
    - no need to arrange the arm posture, just take it
  - unload() : boolean
    - detaches the attached packet and puts it in front of the robot
    - no need to arrange the arm posture, just release it

iv) Topics Subscribed

- Pose Sensor - one per agent
  - offers the current Position of the robot as well as the direction the robot is heading to
- Laser Sensor - one per agent
  - offers the list of positions of the points the lasers are pointing to
  - positions are returned as triple (x,y,z) if there is no abstacle in front of a laser
  - it returns (0,0,0) for that laser
- Battery Sensor - one per agent
  - offers the battery level in percent

2.7 Charging Station Management

i) Service Offers

- start($Map, Charging\_Config$) : boolean
  - hands over the map and the charging configurations.
  - starts all other services.
- reserveChargingZone($Agent\_ID, ChargingZone\_ID$) : boolean
  - if the maximum number of reservations in the charging zone with the given id hasn't been reached yet the agent with the given id gets a reservation
  - otherwise the reservation request is added to a waiting queue
- chargeRequest($Agent\_ID, ChargingZone\_ID$) : boolean
  - returns true if the agent with the given id has a reservation at the charging zone with the given id and the proximity sensors data tells that there are only robots with reservation in the charging zone.
  - returns false if the agent's reservation request is in the waiting queue or if there are are more agents in the charging zone than allowed.
- endReservation($Agent\_ID, ChargingZone\_ID$) : boolean
  - terminates the reservation of the agent with the given id at the charging zone with the given id.
  - removes reservation request from waiting queue.

ii) Service Requests

- Proximity Sensor - one per charging zone
  - getAgents() : [Agent]
    - returns a set of agents located in the radius of the proximity sensor

2.8 Simulation

The simulation done by MORSE [3] simulates a real-world warehouse scenario. All sensors that have been mentioned before get their sensor data from the simulation as well as all actuators have a direct impact to the simulation, so that the data received by the sensors changes over time. In addition to the actuators the Package Input & Output Generator is able to affect the Simulation environment while it creates & places or removes packages while runtime to provide a realistic scenario.

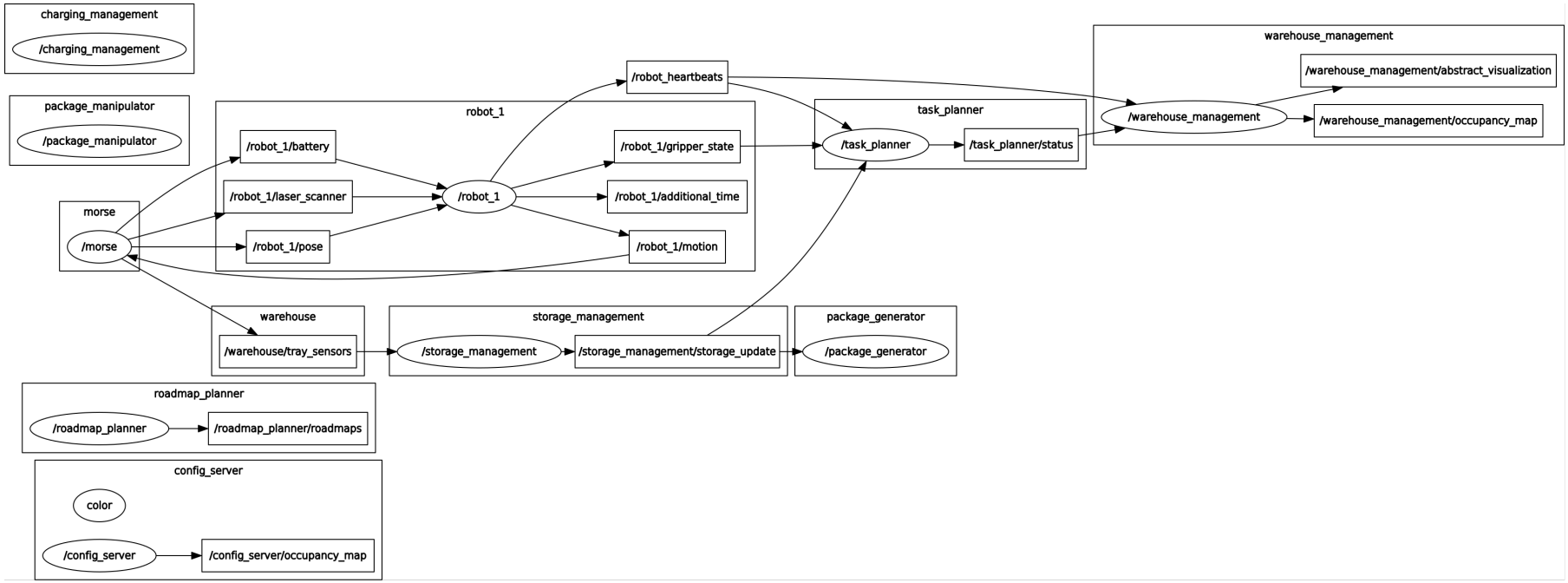

Figure 2: ROS Node graph, rqt_graph, of the integrated system