Factory Simulation

Marco Fontana

June 30, 2025

This project consists of a simulation of a multi agent system where multiple robot BDI agents inside a factory pick up packages from a specific location and deliver those to another, while avoiding obstacles such as walls, other robots and humans in the environment. Each robot has a battery and consumes it at each action it performs to step towards its objective, upon reaching a threshold, beliefs about their inner status makes them desire to reach the nearest charging station and plan new intentions accordingly. Meanwhile, robots acting in the environment may randomly malfunction with a given probability or reach a battery level of 0 and ask for help to other robots, that dinamically change their plan to postpone objectives they were intending to reach and prioritize reparation and battery recharge of the malfunctioning robots before continuing with their previous task.

Contents

1 Goals/Requirements **4**
- 1.1 Scenarios 4
- 1.2 Self-assessment policy 4

2 Background **4**

3 Requirement Analysis **5**

4 Design **6**
- 4.1 System structure 6
- 4.2 Domain entities 6
- 4.3 Behaviour 10

5 Technologies Used **12**

6 Validation **13**

7 Run project instructions **13**

8 Usage example **13**

9 Future works **15**

List of Figures

1 Robot agent activity diagram 7
2 Human agent activity diagram 11
3 Simulation GUI example 14

1 Goals/Requirements

The goal of this project is to:

- create a simulation of a multi agent system where robots in a factory pick up and deliver packages while exibiting an intelligent behaviour that makes them collaborate with other robots to achieve their goals and make the system as a whole able to self-sustain, making it able to autonomously recover from critical situations.
- implementing the simulation using Jason and showing it in a GUI.
- testing the system to make sure it works as expected.

1.1 Scenarios

Since this is a simulation about autonomous intelligent agents, the user is not required to do anything specific rather than monitoring the situation to always be able to know the location and task that each robot is doing at any given time.

1.2 Self-assessment policy

The quality of the produced software may be assessed by the numerous automatic tests that were made to make sure it works correctly in each scenario.

2 Background

Many aspects of intelligent systems were taken into account in order to produce the software for the project or were shown in the behaviour of the agents, some of the aspects that are required in order to understand the overall project are the ones discussed as follow:

The concept of autonomy has different aspects, it is about the encapsulation of control that cannot be directly controlled from the outside, the ability to achieve goals independently and the capacity to generate and pursue self-determined goals rather than externally imposed ones.

Agents are goal-oriented entities, whose behaviour is not casual and acts to reach it. Each agent perceives the environment to first survive and then behave in an intelligent way, so it is aware of its situation and the context it is immersed in. It can exploit its own knowledge about the world to reason about what to do, through beliefs, and possibly plan its course of actions accordingly. They can be defined as autonomous computational entities characterized by:

- **Autonomy**: agents encapsulate control along with criteria to govern it, making them self-regulating entities;

- **Pro-activity**: agents are active entities, actively taking actions to achieve their goals;
- **Situatedness**: agents are intrinsically coupled with the environment where they live and interact, their actions change the environment, and their knowledge about it influence their decisions;
- **Interactivity**: agents exchange informations and knowledge with other agents.

Autonomy has no meaning in isolation and only becomes meaningful when agents are part of a society of them, so an agent is autonomous in relation to other agents. Autonomous agents live and interact within agent societies and multi agent systems, systems where multiple agents are acting in the environment to achieve their goals, and other entities cannot be ignored while doing it. Also, some goals can only be achieved with the cooperation of others: so social abilities are required by agents to interact via some kind of communication, like signals and messages or environment traces that make the other agents choose actions accordingly.

Agents taken into consideration for this project where the BDI agents, which have 'beliefs' about how the world currently is, that represents the agent's knowledge about the world at any given time, including informations about the environment obtained from perception devices, messages from other agents and internal informations; 'desires' that represent the state of the world that the agent is trying to achieve, and 'intentions' that are the chosen means to achieve the agent's desires, generally implemented as plans and post-conditions.

Nature inspired concepts like autopoiesis, that refers to the abilty of complex systems to maintain they overall coherence about their structure and organisation, through mutual interactions of its components.

3 Requirement Analysis

- **Functional requirements**
  - robots have to take packages from a truck and deliver them in the delivery location, then repeat.
  - robots have to find and compute the nearest charging station after reaching a battery level lower than a given threshold, then move towards it, recharge, and restore the previous task.
  - robots have a random chance of stopping or reach a battery level of 0 that makes them enter the malfunctioning state, where they notify other robots about their status and ask them for reparations.
  - robots that receive a repair request from other malfunctioning robots have to reach them, repair them, and share a portion of their battery

to them, in order to restore their normal statuses, before continuing with their previous task.

  - robots have to avoid obstacles while executing their movement actions and never get stuck due to static environmental conditions.

- **Non functional requirements**

  - robustness, considering a swarm of robots, if some of them malfunction, robots collaborate to restore autonomously normal working behaviours.
  - scalability, if required, many other robots of the same kind can be easily added to the system.
  - parallelism, considering multiple robots working together, it is possible to deliver more packages (considering the same time interval), compared to a limited amount of robots.

4 Design

4.1 System structure

The simulation was created following the MVC pattern, where the simulation of the factory has a model that contains the items of the simulation, a controller to handle the communication between model and view, and a view that updates the GUI accordingly.

Many agent's internal actions were defined in Java classes exploiting the compatibility between Java and Jason.

4.2 Domain entities

The environment where the agents are immersed in, is a grid world, where each cell contains agents, obstacles and other key locations.

Several entities were created to simulate this MAS, each one with a specific meaning:

- **Robot**: the main agent is the robot, it is responsible for picking up items from a truck and deliver them to the delivery location (its behaviour is mostly shown in Figure 1).

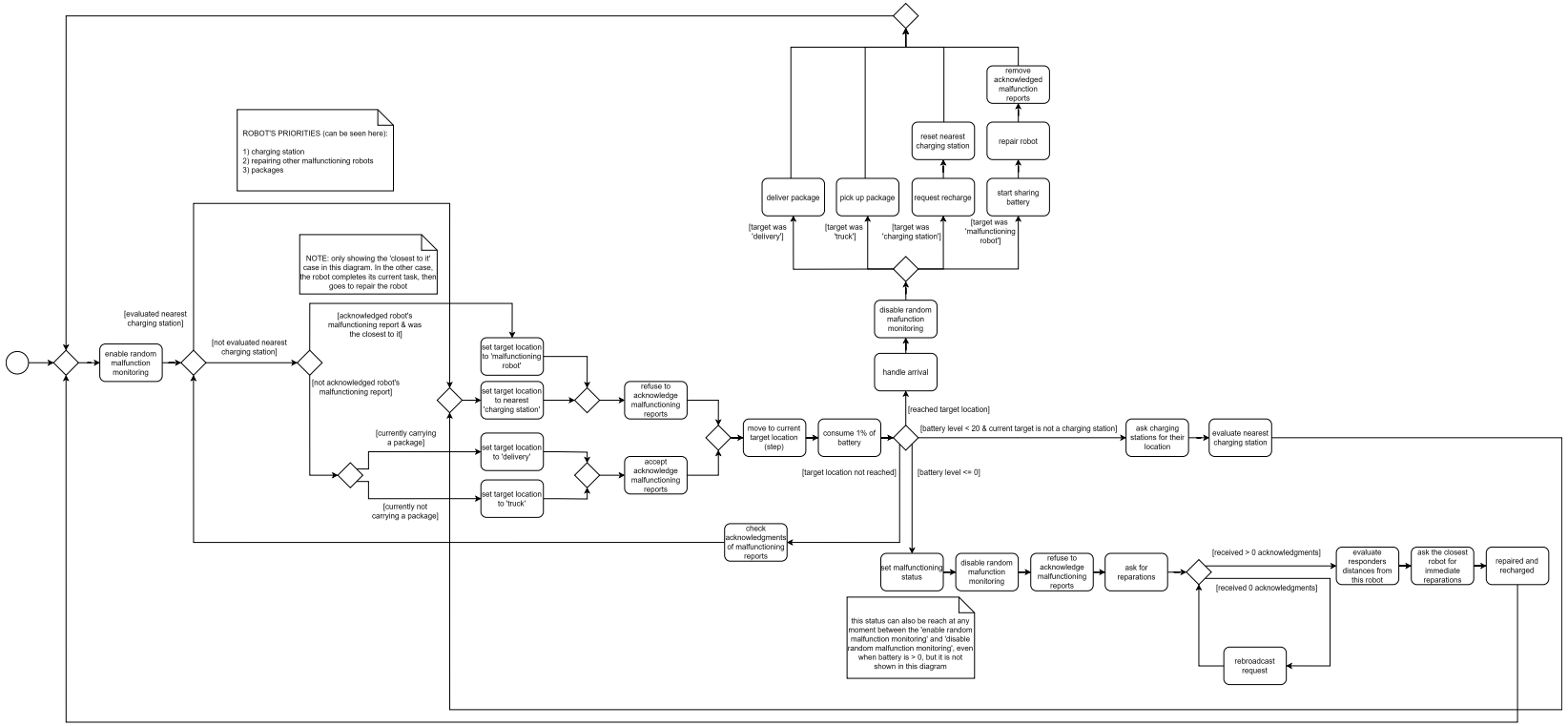

Figure 1: Robot agent activity diagram

Once initialized, they are randomly positioned inside the environment, not carrying any packages, and have to reach the truck.
In order to achieve this, they move one step at the time towards their objective, with a higher chance of moving in its direction and a lower chance of moving away from it, sensing and avoiding any obstacle in between, that are represented by other robots, humans and static walls; this was done in order to prevent robots from getting stuck in a situation where, due to environmental conditions such as an unlucky wall disposition, robots would keep moving in a loop making the same actions over and over never being able to move around walls to reach their target.

Each time they try to move one step towards their target, they use 1% of their battery. They're initialized with a random battery value between 80 and 100, and when reaching a battery less than or equal to 20, they change their current goal and ask the charging stations for their positions; once obtained each response, they compute the closest one by evaluating their euclidean distance (not considering static obstacles in between them, neither other robots that may be currently using it) and start moving towards it.

In the simulation environment provided, the number of robots is purposedly higher than the number of charging stations, to make some of them unable to reach it and recharge correctly.

If robots reach the charging station before the battery gets to 0%, they get recharged to 100%, otherwise they enter the 'malfunctioning' state, where they stop moving and ask for reparations; once in the malfunctioning status, robots send a broadcast message to each other one, and based on their current statuses (here referring to the 'helpers' robots), they can decide whether to repond with and acknowledge or not (i.e., if a robot is currently malfunctioning, it won't respond to the other malfunctioning robot; the same happens if a robot is already recharging and reparing another robot; or is already moving towards a charging station because its battery is too low to be able to help other robots), after collecting all the acknoledgments, the malfunctioning robot chooses immediately the closest one and asks for its help (it will immediately update its plan to help that robot before continuing with its tasks), while the others can still arrive but only after they have achieved their current goal. This is obstained through the malfunctioning robot that gives the others its position, and they start moving towards it (meanwhile, if it takes too much time to get reparations, robots start a timer of 1 minute to rebroadcast again their reparation request).

After reaching the malfunctioning robot, they start sharing their battery with him up to a maximum amount of 25 units of battery, this can stop

previously, if the helper agent reaches a battery level of 30 or below while giving its battery to the other robot; if the helper robot, after the battery sharing, has reached an amount of battery higher than 0%, then it resumes working restoring its previous task.

In certain circumstances the robots may not share at all any battery with the malfunctioning robot, due to the fact that while moving towards it, their battery dropped to a value between 20 and 30, because when they started moving towards it they had a battery higher than 30, but reached it before reaching the threshold of 20 that makes them move towards a charging station first (a battery higher than 30 is required to share battery with another robot).

The malfunctioning state can also be reached at any moment, due to a random chance of happening, each robot concurrently monitors its state at any time, to check for malfunctions, sometimes it will detect a random malfunction and enter the malfunctioning state; the malfunction monitor stops only when the robot is already malfunctioning, charging (included battery sharing), and completing a delivery, and resumes immediately after (while the monitoring has stopped, no random malfunctions may occur, but malfunctions due to reaching a battery level of 0% may still occur).

After reparations from another agent, or being recharged by a charging station, the robot resumes working to reach whatever goal it was trying to achieve before that.

Edge cases: many edge cases can occur while executing the described robot's behaviour, some of them are:

- the distance from the current target (can either be one of the ones mentioned above: other robots, charging stations, etc...), is evaluated not considering obstacles in between, so it may happen that if the robot has to choose between 2 targets, it may choose the nearest one, but that requires more time to reach with respect to the other target, due to environmental conditions (i.e., walls).
- even though it's improbable, there's still a slim chance of each robot entering the malfunction state simultaneously (due to a random chance of happening, reaching 0% battery level, or a combination of both), in this situation the simulation would stop, because no agent is able to resume working. In order to solve this, a human technician agent was added to repair a random robot and recharge it (see human agent below), to keep the system able to self-maintain over time.
- if no robot acknowledges the repair request (due to each one currently reaching a charging station, helping another robot, malfunctioning,

etc...), after a fixed amount of time, a new message is sent as broadcast to all robots, in order to eventually receive an acknoledge when at least one is finally able to repair it.

  - if a robot is delivering a package, then it is required to move towards a malfunctioning robot to help it, but its battery drops below 20%, its plan dinamically adapts and becomes: first move towards the charging station, then move back to the robot, and only then go back to the delivery location where you were previously going towards.
  - if multiple robots are moving towards the same malfunctioning robot to repair it, and once reached its location the robot was already repaired by another one, they ask the robot if it was repaired, and if, so immediately resume their previous task.

- **Charging station**: charging stations are required to charge each robot that gets inside of them, only one robot at the time can access a charging station. Once reached, a message is exchanged between the robot and the station, and the charging process starts, with the charging station charging the robot until 100%, independently of the battery it had when entering it.

- **Human**: a human agent is randomly placed in the environment. Its purpose is to be another obstacle to avoid and it continuously asks robots for their statuses. Robots concurrently share their state with the human, that repairs remotely a random robot and recharges it to 100% only in the edge case where all robots are malfunctioning at the same time (its behaviour of asking robots their statuses and repairing one at random, is described at Figure 2).

  The idea of recharging a single robot was done to make the simulation gradually resume by itself, where the repaired agent autonomously starts repairing another robot, that will start doing the same until eventually all of them are repaired and up and running again. So the system is able to self-sustain over time.

- **Truck and delivery location**: both of them wait for a robot to be at their position, then reply to messages that either ask for a package or confirm for the deliver. No other obstacle or key location can randomly be placed adjacent to those, in order to make it possible to always reach it by robots in adjacent cells.

4.3 Behaviour

The software agents programmed are autonomous and encapsulate their control logic, only by sharing data between them, they are able to cooperate and take actions accordingly.

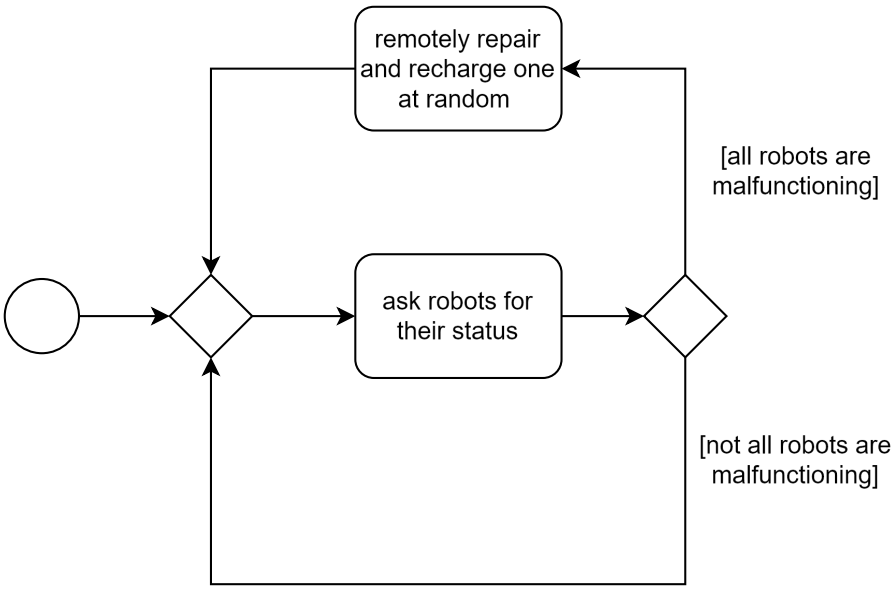

Figure 2: Human agent activity diagram

The system as a whole, shows a typical autopoiesis ability to maintain its overall coherence through mutual interactions of the network of agents, that are able to restore other malfuncioning robots by repairing and sharing their battery to them, in order for the system to be able to self-maintain over time and restore its overall standard system behaviour of queues of robots picking up and delivering packages.

In fact, upon malfuncioning, agents are able to exchange informations to other agents communicating their current state that makes the receivers acknoledge their state and change their own plan in order to be able to restore standard working behaviours; each one has its own control logic that does not pass through agent boundaries, so they cannot be controlled externally, but only take actions based on the informations that are exchanged outside their boundaries.

The robot agents behave intelligently, and in a proactive way, to make sure to firstly be able to maintain an operational behavior, by prioritizing their movements towards charging stations in order to make sure to always have enough battery to sustain their goals (and this means adapting and changing their current objective dinamically to prioritize this one over the others, like helping another robot, and execute the other ones in a different order than what were initially planned), and then going back to execute the previous tasks only once in a condition that can sustain them. So actions are planned and executed in a specific order, and also, based on the environment, the robot can autonomously

adapt to changes and plan to execute actions in a different order with a goal-oriented behaviour.

The environment considered in this simulation can be considered as 'inaccessible', due to the fact that the whole state of the environment cannot be completely observable by the agents (the environment is partially observable), i.e. they do not know the position of all obstacles until adjacent to them; also, robots moving towards a malfunctioning one, to help it, do not know if it has already been repaired, and moved away, while the helper robot is reaching its latest known position, only after reaching it, they check if their knowledge about the malfunctioning robot is still valid and take a decision on helping it or going back to their previous task. Another example is when the agent is seeking a charging station, while doing it, the robot does not know if other agents are currently charging in it or not, and only after they have reached it, they try to enter, but are not permitted access until the agent that is currently inside, exits.

This environment can also be considered 'static', meaning that it remains unchanged except by the effect of agent actions, and, due to the fact that this simulation considers a basic environment, it can be considered as 'deterministic', in fact, any action of the agents have a guaranteed and predictable effect (that does not necessarily imply that agents always achieve their goal: as described earlier, if a robot is not anymore malfunctioning, upon reaching its latest known position, the helper robot will restore its previous task, which makes it behave in a predictable way, even though it may not always imply that it will achieve its goal correctly).

5 Technologies Used

Different tecnologies, like the ones described below, where used to develop this simulation:

- Gradle, to automate the building process of the project, chosen for its ease of use.
- Jason, a robust agent-oriented programming platform that implements the AgentSpeak(L) language for developing intelligent agents in multi-agent systems. Jason provides a comprehensive framework for building BDI (Belief-Desire-Intention) agents with built-in support for agent communication, environment interaction, and distributed system deployment. The platform offers powerful reasoning capabilities through its logic-based approach, allowing agents to maintain beliefs about their environment, pursue goals through intentions, and execute plans dynamically.

6 Validation

In order to validate the correctness of the software produced, many types of automatic tests, focusing on different aspects, were created and can be checked and run in the 'test' folder.

7 Run project instructions

In order to successfully run the project and its tests, make sure to have Gradle installed.

To run the simulation it is sufficient to run gradle's task '`runFactoryMas`'.

8 Usage example

The interface shows the location of each agent at any given time, their ID, battery and current objective using different colours.

An example of agent, that can be monitored in the GUI (see Figure 3), is the one with a round shape. Those ones are 'robot' agents, and the text over them shows their ID and battery; i.e., as shown in the picture, the robot with written over it "R2 56" (in the center, at the right side of the grid), means that the robot has an identifier "R2" that stands for "delivery robot 2" and its battery is currently at 56%, its color changes over time to match the goal it is currently trying to achieve, as shown in the picture, it is currently carrying a package and trying to reach the delivery location "D" (this matches the color 'cyan').

Another one is the human, that has an identifier "H", a brown color (that is the only one it has) and a square shape.

All the possible colors that each robot may have are shown in the 'legend' next to the grid, and are:

- **dark cyan**: going towards truck to pick up a new package
- **cyan**: carrying a package and going towards delivery location
- **orange**: going to the nearest charging station to recharge
- **yellow**: recharging at a charging station
- **dark red**: currently malfunctioning and waiting for repairs
- **magenta**: moving towards a malfunctioning robot to repair it
- **dark magenta**: currently sharing battery to a malfunctioning robot and/or repairing it

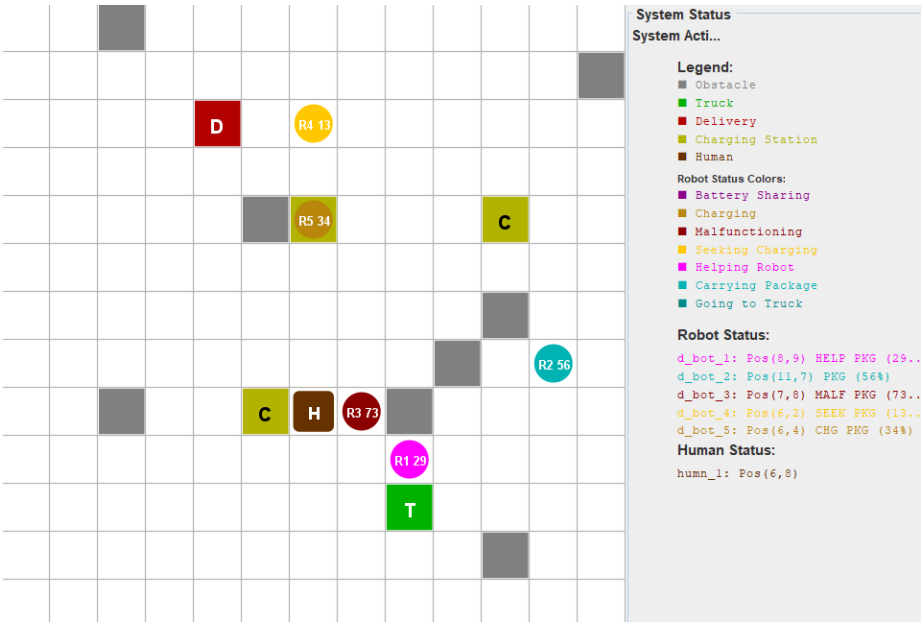

Figure 3: Simulation GUI example

Key locations and other agents are:

- **Gray square**: are obstacles that each robot has to avoid (along with the human and the other robots).
- **H**: is the human.
- **C**: are the charging stations where robots charge after reaching a battery level below 20%.
- **D**: is the delivery place where robots deliver their packages.
- **T**: is the truck where robots pick up packages from.

Each agent's status can also be monitored through the 'robot status' section, that shows, using texts, the robot identifier, its current position, objective, status (also shown using a text color that matches the one described before), and its battery level.

Their current objectives and statuses are described with words:

- **PKG**: currently carrying a package and going towards the delivery location (same as cyan color).
- **HELP**: currently moving towards a malfunctioning robot to repair it (same as magenta).

- **SEEK**: currently moving towards a charging station (same as orange).
- **CHG**: currently charging at a charging station (same as yellow).
- **MALF**: currently malfunctioning and waiting for reparations (same as dark red).
- **BSHARE**: currently sharing battery and/or repairing another malfunctioning robot (same as dark magenta).
- ***blank***: currently moving towards truck to pick up a package (same as dark cyan).

Each robot status text is formatted the same way:

```
agent_id:  Pos(current_x_position, current_y_position) STATUS (battery_level%)
```

i.e. considering again the "R2" robot, its status in the picture, formatted as text, is:

```
d_bot_2:  Pos(11,7) PKG (56%)
```

Meaning that its position in the grid is (11,7), carrying a package and moving towards the delivery location, and currently has a battery level of 56%.

9 Future works

It is possible to extend this simulation by adding multiple trucks and delivery places, and also considering different kinds of robot agents, each with its own characteristics.