Multi-Agents Systems
Warehouse Simulation

Di Via Roberto 4486648
4486648@studenti.unige.it

15 June 2023

Contents

1 Introduction

A Multi-Agent System (MAS) is a computational system composed of multiple autonomous agents that interact with each other and their environment to achieve common or individual goals [1]. Agents are autonomous, proactive, and reactive software/hardware entities that act upon an environment using sensors and actuators to achieve their goals. They are autonomous in that they have their own goals and objectives. They are proactive in that they take the initiative to achieve their goals. They are reactive in that they respond to changes in their environment. Agents can communicate, collaborate, and coordinate their actions to accomplish tasks, solve problems, or perform complex behaviours that would be challenging or impossible for a single agent alone.

MASs are widely used in various domains such as robotics, logistics, finance, traffic management, and social simulations. They offer a decentralized approach to problem-solving, where agents can act independently and adapt their behaviours based on their perception of the environment and their communication with other agents.

1.1 Overview

In this report is described the development of a Multi-Agent System (MAS) that simulates a warehouse environment. The purpose of this MAS is to efficiently manage the movement and coordination of multiple agents to fill racks with items taken from the stock, pick all the items from the last filled racks, and deliver them to a courier upon request. The system aims to optimize the overall operation of the warehouse by leveraging collaboration and communication between the agents.

The next chapters will present an overview of the technologies used, a description of the MAS architecture, and a description of the agents' design.

2 Technologies Used

The MAS is developed using Java[4] for handling the environment and the Jason (v3.1)[3] platform, based on AgentSpeak(L)[2] language, for programming the agents. Combining these two languages a robust and efficient MAS can be created that effectively simulates warehouse operations.

2.1 AgentSpeak(L)

AgentSpeak(L) is a logic-based agent-oriented programming language that is based on the Belief-Desire-Intention (BDI) architecture[6]. This architecture is based on the idea that agents have beliefs about the world, desires about what they want to achieve, and intentions about what they plan to do to achieve their desires. The beliefs of an agent represent its knowledge about the world. The desires of an agent represent its goals. The intentions of an agent represent its plans for achieving its goals. So, in summary, AgentSpeak(L) allows agents to reason about their beliefs, desires, and intentions enabling them to make decisions and perform actions based on this reasoning.

2.2 Jason

Jason is an open-source interpreter for an extended version of AgentSpeak. It is specifically designed for developing Multi-Agent Systems and it provides a platform that facilitates the implementation, execution, and coordination of agents within a MAS.

The Jason platform enables the creation of multi-agent environments and enhances the basic AgentSpeak language by introducing additional features and constructs, making it more suitable for real-world applications. It offers a comprehensive range of built-in functions, communication primitives, and execution control mechanisms, which enhance the language's expressiveness and flexibility.

To gain a deeper understanding of the AgentSpeak language and its application in developing multi-agent systems using Jason, I referred to the book "Programming Multi-Agent Systems in AgentSpeak using Jason."[5]

2.3 Java

Java is used within the MAS to represent the system using a Graphical User Interface (GUI), to handle environmental percepts and execution of agent actions. While Jason primarily focuses on programming agents' behaviours and decision-making, Java allows for the definition of the environment's state and the agents' actions through classes and methods.

2.4 Execution of the Multi-Agent System

To execute the implemented MAS follow these steps:

1. Ensure to have a Java Development Kit (JDK), higher than the 1.8 version, installed on the system.

2. Download and install the Jason platform.
3. Configure the Java development environment with the necessary libraries and dependencies, including the classpath to the Jason jar file.
4. Start the Jason platform and load the configuration file ”progetto_mas.mas2j”.
5. Click on the execute button to run it.

3 System Architecture

In this chapter, is described the architecture of the implemented MAS, which simulates a warehouse environment. The system consists of multiple agents, an environment, and their interactions presented below.

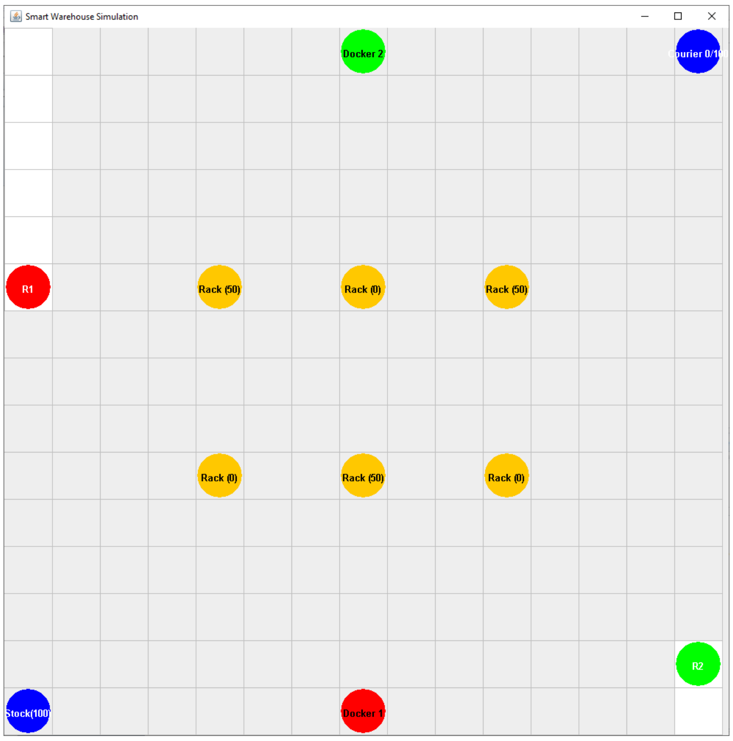
3.1 Agents

The agents are the software entities that interact with each other to achieve the goals of the MAS. The MAS comprises two mobile agents, Robot 1 and Robot 2 and two stationary agents, Stock and Courier. Each agent has specific roles and responsibilities within the warehouse environment.

- Robot 1: This agent is responsible for filling the racks with items. It does this by repeatedly checking the state of the environment to see if there are any empty racks. If there are empty racks, the Robot 1 agent will pick items from the Stock agent and place them in the empty racks.
- Robot 2: This agent is responsible for handling the Courier agent's request, picking the items from the last filled racks and bringing them to him. It does this by repeatedly checking the state of the environment to see if there are any filled racks at full size. If there are, Robot 2 will go to the last filled rack to pick the items and it will deliver them to the Courier agent until the request isn't satisfied (using a recursive plan).
- Stock: This agent is responsible for managing the availability of items in the stock. It does this by refilling the items in the stock when requested by the Robot 1 agent.
- Courier: This agent is responsible for handling deliveries to buyers. It does this by requesting (every 10 seconds) items from the Robot 2 agent in order to deliver them to the customers.

3.2 Environment

The environment continuously updates and provides percepts to the agents, allowing them to perceive the state of the world. In the developed MAS, both the Courier agent and Robot agents receive percepts from the environment. The percepts include information such as the object and agents' locations, the availability of items in the Stock, and the state of the Racks.



The environment is represented as a grid of cells of dimension 15x15. Each cell can contain a component or be empty, and the agents can move from one cell to another by following a path. There are different approaches to path search, in this case, a basic path search algorithm was used. It calculates the necessary movement directions to bring the agent closer to the destination, incrementing or decrementing the coordinates along the x-axis and y-axis accordingly.

The warehouse consists of several components, including 6 racks (each with its own capacity and location) for filling or emptying, a Stock location where Robot 1 (R1) retrieves items, a Courier location where Robot 2 (R2) delivers items, and two Docker stations where mobile agents can recharge their batteries.

3.3 Communication

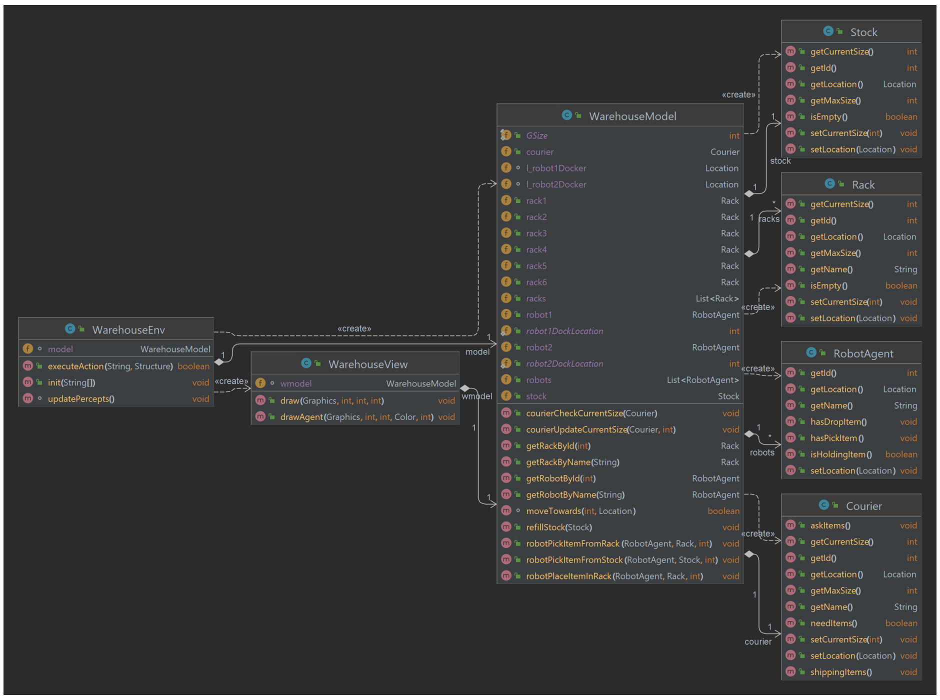
In a MAS, agents interact with each other and the environment to achieve their respective goals efficiently. The agents can use a different number of communication protocols to interact; they can send direct messages to each other, they can broadcast messages to all agents in the system, they can subscribe to events that other agents generate, and they can also observe the state of the environment by reading data from the environment.

In this specific case, Robot 1 and Robot 2 interact with the environment to perceive the state of the racks for filling or emptying. Robot 1 further interacts with the Stock agent to request a refill when the stock is empty and wait for the event. Similarly, the Courier agent periodically interacts with Robot 2 to request the items needed for customers and Robot 2 interacts with the Courier agent to inform about the items delivered.

These interactions allow the agents to collaborate and manage warehouse operations effectively, ensuring that the stock is never empty, racks are always filled, and the courier's requests are satisfied.

3.4 System Architecture Diagram

To visualize the system architecture, a UML[7] class diagram that illustrates the main components and their relationships is provided.



4 Agent Design

In this chapter, it's explained the design of the agents in the implemented MAS, providing insights into their beliefs, intentions, and plans to demonstrate how the AgentSpeak language is utilized.

4.1 Robot 1 Agent

The Robot 1 agent has the role of filling the empty racks with items taken from the stock. Let's explore the beliefs, rules, and plans of the Robot 1 agent.
Beliefs:

- empty(Rack): This belief represents the list of racks that are currently empty and need to be filled.
- picked_items(Items): This belief indicates that the agent has picked N items. This is used as a condition to execute some plans and to update, with some action, the size value of the racks and stock.
- empty_stock: This belief is perceived by the environment and indicates that the stock is empty and needs to be refilled.
- stock_full: This belief is added by the Stock agent and indicates that the stock is being refilled.
- at(robot_1, Location): This belief is perceived by the environment and indicates that the agent is at a certain location, for example, the Stock area or the Rack to fill.
- battery_level(Level): This belief represents the battery status of the mobile agent. When is under a certain level the agent must go to the corresponding Docker location to recharge itself.

Goals:

- fill_rack: This is the main agent's goal, fill the empty racks with items from the stock.

Plans:

- move_to(Location): This plan is executed when Robot 1 has enough battery to reach a certain location, like the stock or the rack to fill. If the agent hasn't enough battery, it goes to recharge at the Docker location.

- pick_items_from_stock(N): This plan is executed when the Robot 1 agent is at the Stock location and perceives that is not empty. The stock size is then updated according to the N items picked and the picked_items belief updated. If the stock is empty it sends a message to the stock agent saying to refill it, waits for the event, and then continues to pick items from the stock.
- bring_items_to_rack(N): This plan is triggered when Robot 1 has picked a specific number of items (N) and needs to deliver them to the rack. It checks for an empty rack, moves to the rack's location, and puts the items in the rack. If Robot 1 hasn't picked up the desired number of items it initiates the process of filling the rack again.
- check_for_empty_rack: This plan checks if there is an empty rack available. If an empty rack is not perceived it waits for a certain period and checks again.
- put_items_to_rack(N, Rack): This plan is executed when Robot 1 is at the location of an empty rack. It updates the rack's size through action and resets the picked_items belief.

4.2 Stock Agent

The Stock agent ensures that the stock is refilled when required. Let's explore the beliefs and plans of the Stock agent.
Beliefs:

- refill_stock: This belief is added by Robot 1 when it requires the stock to be refilled.

Goals:

- The agent has not an individual goal. He ensures that the Stock is refilled when the refill_stock belief is added from the Robot 1 agent.

Plans:

- refill_stock: This plan is triggered when Robot 1 adds the refill_stock belief. It simulates the process of refilling the stock by waiting for a certain period of time (10 seconds in this case), performing an action, and then notifying Robot 1 that the stock is now full sending the stock_full belief.

4.3 Robot 2 Agent

The Robot 2 agent is responsible for handling the courier's requests for items picking up all the items from the last filled racks. Let's explore the beliefs, rules, and plans of the Robot 2 agent.
Beliefs:

- full(Rack): This belief represents the list of racks that are currently full and so they can be empty.
- picked_items(Items): This belief indicates that the agent has picked N items. This is used as a condition to execute some plans and to update the size value of the racks.
- at(robot_2, Location): This belief indicates that the agent is at a certain location, for example, the Courier area or the Rack to empty.
- battery_level(Level): This belief represents the battery status of the mobile agent. When is under a certain level the agent must go to the corresponding Docker location to recharge itself.

Goals:

- handle_courier_request: This is the main agent's goal, bring items from racks to courier when requested.

Plans:

- go_to_courier: This plan is triggered when Robot 2 receives a request to go to the courier location.
- move_to(Location): This plan is executed when Robot 2 has enough battery to reach a certain location, like the Courier or the rack to empty. If the agent hasn't enough battery, it goes to recharge at the Docker location.
- need_items: This plan is executed when Robot 2 receives a request from the Courier for a specific number of items. It handles the courier's request using a different plan.
- handle_courier_request(N): This plan is responsible for handling the courier's request for items. It checks for full racks, moves to the full rack's location, picks a specific number of items (50 in this case) from the rack, delivers them to the courier, and updates the picked_items belief. It then recursively handles the remaining items requested by the courier. When all items requested by the courier have been delivered it moves to the docker location waiting for the next request.

- check_for_full_rack: This plan checks if there are any full racks available. If a full rack is not perceived the agent waits for a certain period and checks again.
- pick_item_from_rack: This plan is triggered when Robot 2 is at the location of a full rack. It picks a specific number of items (N) from the rack and updates the rack's size and picked_items belief.
- deliver_items_to_courier: This plan is executed when Robot 2 has picked a specific number of items (N) and needs to deliver them to the courier. It moves to the courier's location, delivers the items and resets the picked_items belief.

4.4 Courier Agent

The Courier agent is responsible for requesting items from the Robot 2 agent in order to ship them to the customers. Let's explore the beliefs, goals, and plans of the Courier agent.
Beliefs:

- received_items: This belief represents the number of items that the Courier agent has received, it is reset for every new request.

Goals:

- requestDelivery: This is the main agent's goal and consists to request N items from the Robot 2 agent.

Plans:

- call_robot: This plan asks the Robot 2 agent to reach the Courier location and waits for it. It checks the location of Robot 2 and waits for a certain period before checking again.
- request_items: This plan is triggered when the desired number of items has been received. When this happens, the received_items belief is set to 0 and new items are requested after 10 seconds. If the received_items belief doesn't correspond to the number of items needed it sends a message to Robot 2 requesting them.
- delivered_items: This plan handles the case when items are delivered by Robot 2 updating the received_items belief with the newly delivered items.

5 Conclusion

In conclusion, this report has provided a comprehensive overview of developing a Multi-Agent System, with Jason, for simulating an intelligent and autonomous warehouse. Fundamental concepts are explored, such as agent architecture, environment modeling, perception and action handling, coordination mechanisms, and communication protocols.

The field of MAS holds immense potential for addressing complex real-world problems and improving efficiency in various domains. As technology continues to advance and new challenges emerge, there are numerous opportunities for future work and enhancements in MAS development to further augment its capabilities.

5.1 Future Work and Enhancements

While this report has provided a comprehensive understanding of MAS development for the warehouse domain, there are several promising directions for future work and enhancements:

- Dynamic Request Quantity: Enhancing the courier agent's capability to request a variable quantity of items based on user demands can significantly improve the system's flexibility and responsiveness. Implementing intelligent algorithms that adaptively adjust the requested quantity can optimize resource allocation and improve customer satisfaction.

- Advanced Pathfinding Algorithms: Integrating advanced pathfinding algorithms, such as the AlphaStar algorithm, can enhance the robots' navigation efficiency by preventing overlaps and minimizing travel distances. By intelligently planning paths, the system can optimize resource utilization and reduce delivery times.

- Goal Suspension and Task Reassignment: Introducing mechanisms for goal suspension and task reassignment during recharging periods can enhance the overall efficiency of the MAS. While one robot agent is recharging, another robot can temporarily take over its suspended goals to ensure continuous progress and optimize task completion.

References

[1] AI for Anyone. *Multi-agent system.* URL: `https://www.aiforanyone.org/glossary/multi-agent-system`.

[2] Università di Bologna. *AgentSpeak(L) and Jason*. URL: `https://core.ac.uk/download/pdf/17198968.pdf`.

[3] Jason. *Java-based interpreter*. URL: `https://jason.sourceforge.net/wp/`.

[4] Oracle. *Java Technical Details*. URL: `https://www.oracle.com/java/technologies/`.

[5] Michael Wooldridge Rafael H. Bordini Jomi Fred Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason*. 2007.

[6] Wikipedia. *Belief-Desider-Intention architecture*. URL: `https://en.wikipedia.org/wiki/Belief%E2%80%93desire%E2%80%93intention_software_model`.

[7] Wikipedia. *Unified Modeling Language*. URL: `https://en.wikipedia.org/wiki/Unified_Modeling_Language`.