Multi-Agent Distribution Centre Coordination

MEIC - 2021/22 - Agents and Multi-Agent Systems - 1st Assignment

João Carlos Machado Rocha Pires (UP201806079)
Rafael Valente Cristino (UP201806680)

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Problem description

In an Amazon facility, multiple robots work together to carry and deliver packages throughout the warehouse with maximum efficiency. Those robots are limited and have a maximum capacity. Based on the daily needs, robots should consider the prioritisation of the packages and the expected time for them to be delivered.

Considering the above problem description, our MAS-based solution should behave in a way that robots (agents) are allocated an item to move and items are scheduled so that they arrive where they are needed at the time they are needed. The allocation of packages to robots should be dynamic, through negotiation between the agents involved.

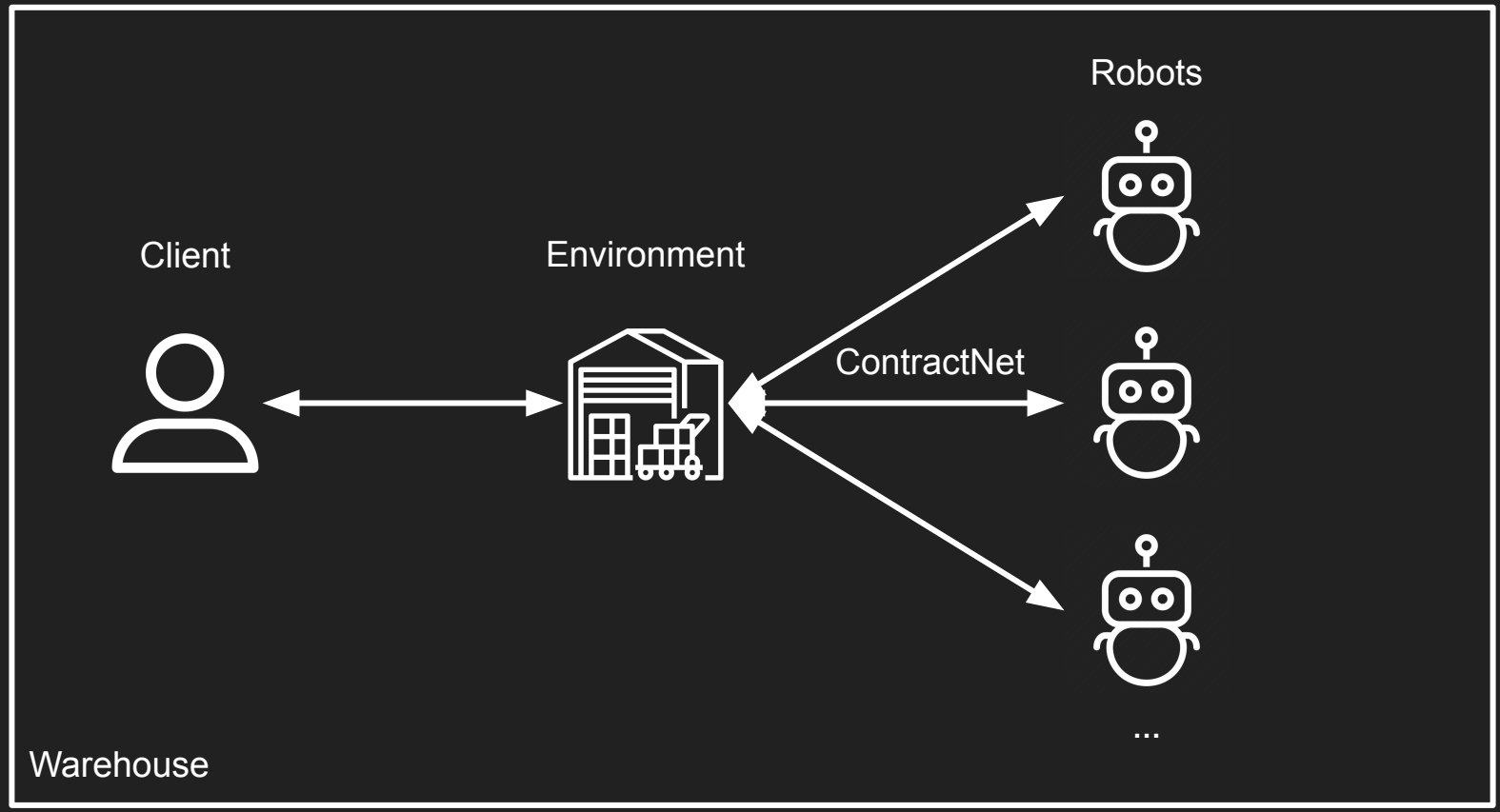
The environment contains infrastructure to hold the items that can be arranged in multiple ways, in order to evaluate its influence on the efficiency of the centre.

Dependent and independent variables

Dependent variables	**Independent variables**
Delayed items (average delay time and number of delayed items)	Environment specification (location of structures, …)
Total number of items distributed in a certain time period	Quantities such as the number of available robots, number of items to move, the distribution of the number of items over time (eg calm time periods followed by a window where the robots are overwhelmed by the number of items that need to be transported)
Robots occupation	Strategies such as the robot allocation strategy and item scheduling strategy
Cost to run the distribution centre	Characteristics of the items (size, weight, shape, …)
	Characteristics of the robots (what weight they can carry, how the items fit in their holding space, velocity, reliability, …)

Agents

Agents interaction/communication



System Configuration

- The system can be configured through a JSON file which allows the user to set multiple variables, such the amount of robots, the characteristics of each robot, whether he wants to specify the packages to be generated or for them to be random, the period through which the packages are sent, among others.


```
{
  "basicTest": {
    "robots": [
      {
        "name": "robot1",
        "capacity": 30
      }
    ],
    "environment": {
      "tickingPeriod": 1000
    },
    "client": {
      "initialPackages": [
        {
          "products": [
            {
              "name": "eggs",
              "location": {
                "x": 10,
                "y": 10
              },
              "weight": 10
            }
          ],
          "deadline": 70
        }
      ],
      "random": false,
      "period": 10000
    }
  }
}
```

Interaction and communication protocols

Between the Client and the Environment

- Each *x* seconds, where *x* is equal to the period defined in the configuration file, the client sends a package to the environment. This package can be previously specified (deterministic) or random. When it is random, it generates a random package with 1 to 5 products. Each product's weight and position is randomly generated, with values ranging from 1 to 10. The package expected delivery time (deadline) is also set as a random number between 20 and 50 and this applies to all the products inside the generated package.
- Upon the reception of the package, the Environment adds each of its products to a priority queue which keeps the products in order of deadline. Periodically on each tick it starts the ContractNet protocol for the product at the front of the queue, in order to attribute the pick up of each of them to a specific robot.

Interaction and communication protocols

Between the Environment and the Robots

- Each robot registers itself in the DF agent. This way, the environment can, upon initialization, recover the list of robots to whom he will have to send the CFP (call for proposal) messages of the ContractNet protocol.
- The environment keeps the “time” - a tick - which is propagated to the robots and keeps them synchronized.
- When the Environment receives a new package from the Client, it adds each of its products to a priority queue which keeps the products in order of deadline. Periodically on each tick it starts the ContractNet protocol for the product at the front of the queue. Upon receiving the CFP message, each of the Robots will make a proposal if he is interested. The **goal of each robot is to deliver the maximum amount of products they can, while not letting items arrive late at their destination**.
- Each robot checks whether he has the capacity for the advertised product and going to get it won’t affect the timely delivery of the others he already has. If so, then he makes a proposal with his current position. The environment then gathers all of the proposals and picks the robot that is closest to the product.

Continuation of the Robots behaviour

- Each Robot keeps the products that it has to pick up in a list, and as they are picked up they are transferred to another list that keeps the products to be delivered.
- The Robot will start delivering the products to the delivery line once he **has no product in the pick up list**, he **can’t fit any more packages**, or he **needs to do so because of some product’s delivery deadline**. Once the products are delivered, the Robot returns to gathering new ones to collect.

Summary of agents architectures and behaviour

Client

- Responsible for the creation of packages and sending them to the Environment.

Environment

- Responsible for storing the packages received from the Client, the list of active Robots and for negotiating with them, using ContractNet, to attribute products for each of them to pick up.

Robots

- Responsible for picking up the products and communicating the status with the Environment.

Experiences and results

- To experiment with the code we used deterministic and random data, that is, the packages that are sent to the Environment agent. The deterministic data is very useful to be able to reproduce certain things that happen when running the code, whereas with random data we could have a hard time trying to reproduce it again.
- Our experiences with deterministic and random data can be replicated with the use of the *configurations.json* file, by toggling the *random* flag.
- Throughout the experiments it was clear that it isn’t impossible for the products to arrive where they are supposed to be late. This happens especially with random data, where a product can be put so far that it would be impossible to take it to the delivery place, but also in other cases, such as when the robots are very filled with items.

Conclusions

- From our experiences, we've concluded that our system works well, but it could be better.
- One thing that we could improve is further optimize our allocation algorithm, in order to make robots follow more efficient paths throughout the warehouse.
- We could also improve our experimentation methods and implement an analyser agent who constantly monitors the performance of the system in more analytical ways.

Additional Information

Execution examples

```
------------------------------------------------------------

[ENVIRONMENT] tick

[robot3] %%% Current robot products:
[robot3] %%% Current position: Position{x=0, y=0}


[robot1] %%% Current robot products:
[robot1] %%% Current position: Position{x=0, y=0}


[robot2] %%% Current robot products:
[robot2] %%% Current position: Position{x=0, y=0}

[robot1] Received cfp: ProductProposal{product=Product{weight=10, name='eggs', location=Position{x=10, y=10}}, deadline=70}
[robot2] Received cfp: ProductProposal{product=Product{weight=10, name='eggs', location=Position{x=10, y=10}}, deadline=70}
[robot3] Received cfp: ProductProposal{product=Product{weight=10, name='eggs', location=Position{x=10, y=10}}, deadline=70}
[robot1] Replied with PROPOSE
[robot3] Replied with PROPOSE
[robot2] Replied with PROPOSE
[ENVIRONMENT] Robot 'robot1' proposed: RobotResponse{freeCapacity=30, position=Position{x=0, y=0}}
[ENVIRONMENT] Robot 'robot3' proposed: RobotResponse{freeCapacity=40, position=Position{x=0, y=0}}
[ENVIRONMENT] Robot 'robot2' proposed: RobotResponse{freeCapacity=20, position=Position{x=0, y=0}}
[robot3] Proposal was rejected...
[robot2] Proposal was rejected...
[robot1] --> Handling product: eggs
```

Execution examples

```
------------------------------------------------------

[ENVIRONMENT] tick

[robot3] %%% Current robot products:
[robot3] %%% Current position: Position{x=0, y=0}


[robot1] %%% Current robot products:
- eggs, deadline: 70
[robot1] %%% Current position: Position{x=1, y=0}


[robot2] %%% Current robot products:
[robot2] %%% Current position: Position{x=0, y=0}

[robot1] Received cfp: ProductProposal{product=Product{weight=10, name='bacon', location=Position{x=15, y=20}}, deadline=70}
[robot3] Received cfp: ProductProposal{product=Product{weight=10, name='bacon', location=Position{x=15, y=20}}, deadline=70}
[robot1] Replied with PROPOSE
[robot3] Replied with PROPOSE
[robot2] Received cfp: ProductProposal{product=Product{weight=10, name='bacon', location=Position{x=15, y=20}}, deadline=70}
[robot2] Replied with PROPOSE
[ENVIRONMENT] Robot 'robot1' proposed: RobotResponse{freeCapacity=20, position=Position{x=1, y=0}}
[ENVIRONMENT] Robot 'robot3' proposed: RobotResponse{freeCapacity=40, position=Position{x=0, y=0}}
[ENVIRONMENT] Robot 'robot2' proposed: RobotResponse{freeCapacity=20, position=Position{x=0, y=0}}
[robot2] Proposal was rejected...
[robot3] Proposal was rejected...
[robot1] --> Handling product: bacon
```

Execution examples

```
------------------------------------------------------

[ENVIRONMENT] tick

[robot3] %%% Current robot products:
[robot3] %%% Current position: Position{x=0, y=0}


[robot1] %%% Current robot products:
- eggs, deadline: 63
- bacon, deadline: 64
[robot1] %%% Current position: Position{x=8, y=0}


[robot2] %%% Current robot products:
[robot2] %%% Current position: Position{x=0, y=0}

[CLIENT] Sending package:
####
Package:
- Product{weight=10, name='flour', location=Position{x=10, y=10}}
- Product{weight=10, name='sugar', location=Position{x=5, y=5}}
####


[ENVIRONMENT] received new package!
####
Package:
- Product{weight=10, name='flour', location=Position{x=10, y=10}}
- Product{weight=10, name='sugar', location=Position{x=5, y=5}}
####
```

Execution examples

```
------------------------------------------------------------

[ENVIRONMENT] tick

[robot3] %%% Current robot products:
[robot3] %%% Current position: Position{x=0, y=0}


[robot2] %%% Current robot products:
[robot2] %%% Current position: Position{x=0, y=0}


[robot1] %%% Current robot products:
- eggs, deadline: 61
- bacon, deadline: 62
- flour, deadline: 80
[robot1] %%% Current position: Position{x=10, y=0}

[robot1] Received cfp: ProductProposal{product=Product{weight=10, name='sugar', location=Position{x=5, y=5}}, deadline=80}
[robot3] Received cfp: ProductProposal{product=Product{weight=10, name='sugar', location=Position{x=5, y=5}}, deadline=80}
[robot1] Replied with IS_FULL
[robot2] Received cfp: ProductProposal{product=Product{weight=10, name='sugar', location=Position{x=5, y=5}}, deadline=80}
[robot3] Replied with PROPOSE
[robot2] Replied with PROPOSE
[ENVIRONMENT] Robot 'robot1' refused.
[ENVIRONMENT] Robot 'robot3' proposed: RobotResponse{freeCapacity=40, position=Position{x=0, y=0}}
[ENVIRONMENT] Robot 'robot2' proposed: RobotResponse{freeCapacity=20, position=Position{x=0, y=0}}
[robot2] Proposal was rejected...
[robot3] --> Handling product: sugar
```

Implemented classes (the most relevant)

- **AgentLauncher** - class that is responsible for initializing a container and all of the agents, including their configuration parameters.
- **Robot, Environment and Client** - the implementation of the agents.
- **RobotBehaviour** - the implementation of the Robot's ContractNet responder, where he decides and makes proposals.
- **RobotTickListeningBehaviour** - a cycling behaviour that listens for the "tick" sign from the environment (passing of time).
- **EnvironmentBehaviour** - A parallel behaviour that has a sub behaviour that listens for packages coming from the client and upon reception starts the dispatcher behaviour.
- **EnvironmentPackageListeningBehaviour** - cyclic behaviour that listens for packages coming from the client.
- **EnvironmentProductDispatcherBehaviour** - the implementation of the environment's ContractNet initiator, where he starts the ContractNet protocol.
- **EnvironmentTickerBehaviour** - the implementation of the environment's "tick", which is a measure of time for the robots.
- **ClientTickerBehaviour** - the client's ticker behaviour that periodically sends new packages to the environment.

Observations

- The source code is split into 4 packages: the **client**, **environment** and **robot** packages include their respective agents and their behaviours; the **utilities** package is more general, it contains things such as the Position class and others.