Final report - Multi-Agents Systems

Emma Machielse, Federico Tavella, Alessandro Tezza (group 59)

April 2, 2018

1 Introduction

Public transport is an universal subject of travelers' complaints, targeted at the waiting and travel time. Attempts of transportation companies to reduce costs often have a deleterious effect on traveling time. A network of intelligent autonomous transportation units could optimize the desired result, lowering costs and traveling time. To illustrate this, we created a multi-agents system for an intelligent bus transportation mechanism in Amsterdam. The goal is to transfer passengers as efficiently as possible. To create this system, we implement several methods stated in multi-agents theory [1] like communication between buses, coordination and negotiation.

2 Framework

We build our system using a Beliefs-Intentions architecture, similar to the Beliefs-Desires-Intentions one [2]. Intentions are future directed and trigger certain actions [3]. Agents can switch between intentions, that each imply a different assortment of tasks to complete. Based on the state of the environment at a specific time, stored in the beliefs and some local variables, the agent sets the most appropriate intention. This framework enables the agents to behave flexible and responsive to their environment.

The buses share the same interests: every bus wants to minimize the costs, the average traveling time of the passengers and the number of messages send. They cooperate to reach these goals, but also compete to obtain resources. Cooperation and competition are generally achieved through communication. Buses can send exchange messages that contain information or propositions.

3 Conceptual description and strategy

In this section, we explain the various agent-related concepts that we use, how we design and implement them, and along which strategies. Four types of costs shape our strategies: (i) the amount of money we spend; (ii) the amount of messages buses send; (iii) the amount of people waiting for a bus; (iv) the average amount of time a person spends waiting for a bus.

Costs arise by traveling and buying more buses. To reduce costs, the buses need to transport travelers efficiently. The more passengers they can transport in the shortest amount of time, the lower the costs are.

3.1 Transportation and Coordination

Buses travel according to a specific schedule. These different schedules describe the connections between various bus stops. In the following paragraphs, we focus on how we chose the different schedules and how the buses travel through them.

Different fixed schedules To create more easily managed transportation areas, we split the map into four different *areas*. These areas can be classified under the labels *"West", "North", "Center" and "East"*. Each area is defined by a schedule of bus stops, and each bus is assigned to a specific area. Each schedule has at least one bus stop in common with another schedule.

Splitting the bus stops into sets has two advantageous effects. Firstly, buses focus on the specific area they are assigned to. This means that it has to complete a shorter distance before passing the same stop again. Thus the waiting time for travelers is reduced. Moreover, the time that a traveler spends in the bus will be shorter, which will reduce chances of buses being full and unable to pick up waiting travelers.

This solution occasionally requires transferring passengers from one area to another. This reduces the advantageous effects somewhat. For transportation of people between bus stops in different schedules, we define the concept of a *joint position*. A joint position is a bus stop that is shared between different areas. Bus stop 4 for example is in the schedule of area *"West"* and in the schedule of area *"Centre"*. So we say that *bus stop 4* is a joint position between those areas. Stop 3 (`Centraal`) is a joint position between all four areas. The joint positions make it possible to transfer passengers from one area to another. When a bus picks up a passenger with a destination in a different area, it will check for the nearest joint position with that other area. After the passenger is transported and dropped off at the joint stop, a bus from the destination area picks up the passenger from the joint position. This second bus can bring the passenger to the final destination. In order to transfer passengers efficiently, we define some conditions to pick up a passenger from a bus stop:

- the passenger destination is in the bus area, **or**
- the passenger destination is not in the bus area **and** the passenger is not already in a joint position with the destination area.

Direction-based pick up Within an area, we define two different schedules. The first one is a route through all the bus stops of the area, the second one is the reverse of the first one. This bidirectional transportation helps to lower travel time. Passengers are picked up by buses that will reach their destination faster, due to the direction they are traveling in. To profit from bidirectional transportation, we add a third condition for pick-up to the list in the previous paragraph. If the bus has to travel fewer stops to reach the destination of a passenger, than a bus traveling in the opposite direction, a passenger is picked up. The destination can be the final destination of the passenger, or the joint position. If this condition does not hold, the passenger is not picked up. As soon as a bus arrives that travels in the opposite direction, the passenger will be picked up. This increases transportation efficiency; buses carry passengers for a shorter amount of time and they are able to carry more passengers.

3.2 Roles

To ensure that the buses cooperate efficiently, they are assigned roles. These roles differ in their ancillary responsibilities, which resulted in a bus hierarchy.

Besides responsability, roles can define the *behaviour* of a bus. The usage of roles makes it easy to change the behaviour of a specific group of buses.

In addition, the roles define the creation time of a bus. At the beginning of the simulation, there is only one bus, with ID 24. However, as soon as it is able to create new buses (i.e., after few ticks), this bus creates another seven buses. These are three local coordinators and four scouts, their roles will be explained in this section.

Global coordinator A major responsibility is buying new buses in case of shortage. If all buses would be able to do that, it would happen inefficiently and increase costs unnecessarily. Therefore we assign this responsibility only to the first bus created, the one with ID 24. This bus is on top of the hierarchy, and its role is defined as *global coordinator*.

The *global coordinator* adds buses based on shortage. Shortage is defined as the difference between the total number of people waiting and the total capacity of the bus fleet. It adds the cheapest bus type that is able to manage the amount of waiting people. Thus, when a shortage is experienced, the global coordinator creates new buses: as we explain later on, the newly created buses are going to be items of auctions in order to be assigned to a specific schedule.

Local coordinators One step down in the hierarchy, we have the *local coordinators*. Each of them is responsible for one of the four areas in Amsterdam. Note that the *global coordinator* is also a **local coordinator**. The *global coordinator* is assigned to the "Centre", whereas the buses with ID 25, 26 and 27 are responsible for "West", "East" and "North" respectively. The glocal coordinator creates all of them at the beginning of the simulation. The local coordinators keep track of their area by listing the buses assigned to their area, their fleet capacity and the number of people waiting. When a new bus is created, all the local coordinators compete between each other to assign the new bus in their area, as described in Section 3.5.

Scouts Less important than the local coordinators are the *scouts*. Note that the global coordinator and the local coordinators are also scouts. They are created at the beginning of the simulation: each schedule has one scout. Their aim is to have a bus that is always moving in the schedule. They do not have specific responsibility.

Simple Finally, at the bottom of the hierarchy we have the simple buses. Every bus is also a simple bus. However, the bus created after the beginning of the simulation are *exclusively* simple buses. This means that they do not have specific things to do, but they only have to pick, move and drop passengers.

3.3 Communication

As previously mentioned, to enable coorporation and competition, we implemented a form of communication. This happens through the exchange of messages. A bus sends a message with a specific goal, and the receiver performs actions based on the content.

Ontology To take into account the possibility of messages with different purposes, we define the following ontology for communication: `"message_type content"`, where `message_type` is the header of the message, or the performative [4]. This is a string that describes the goal of the message. `content` contains the effective content of the message. In this way, buses know how to use the content of the message based on the performative. The perfomatives of the messages can be related to: promotions, assigning buses to schedules, requesting/offering/accepting help, requesting a bid, bidding and reallocating a bus.

The reception of the messages goes as follows. At every tick, a bus checks its incoming messages. To check whether a message is new, its tick is compared to the latest tick in which the agent checked its inbox. This last tick is represented by a locally stored variable. The performative of the message implies how to interpret the content. Based on the content, the bus might changes its intentions. For example, when a local coordinator gets a message of type "*bid-request*", the new intention of performing a bid is added.

3.4 Group decisions

The division of the route into four areas can lead to certain scenarios that involve unnecessary costs. We take steps to deal with a situation, where one area has a shortage of buses whereas the other area has a remnant.

Local coordinators watch out for shortages, by comparing their fleet capacity to the amount of people waiting. Whenever a bus is created and allocated to a schedule, the corresponding local coordinator receives a message. With this information, the local coordinators keep track of all the buses that are assigned to their area. This is how they can keep track of their fleet capacity. Based on the comparison of fleet capcity and amount of people waiting in their area, they decide whether they need more buses. If this is the case, they collaborate with the other coordinators.

The coordinators collaborates by distributing buses between them, based on who needs them the most. This collaboration happens through exchanging messages. When a local coordinator faces a shortage, it sends a message to the other coordinators. The message has performative `help-request` and the content is a number. This number represents the difference between the local fleet capacity, and the amount of people waiting in their area. The other coordinators ask to the buses in their fleet if they are empty (i.e., performative `empty-check`). If a bus is empty, it can be transferred to the area that needs support. The empty bus sends a message with performative `reallocable` and their ID as content, to the coordinator that requested help. Upon receiving this offer, the requester evaluates how many buses it needs to resolve the shortage in the area. If the offer is welcome, the coordinator adds the bus to its fleet. It does so by sending a message with performative `reallocated` and its coordinator ID to the offered bus. The offered bus changes coordinator and will travel according to the different schedule. The old coordinator will remove this bus from its fleet, after it receives a message with performative `bye-bye`. Buses whose offer is rejected, receive a message with performative `disengage`. That tells them that they do not need to change schedule.

3.5 Negotiation

The local coordinators compete between each others in order to obtain the newly created buses.

When the global coordinator creates a new bus, it also sends a message `bid-request` to every local coordinator, that starts the auction for the new bus. Every local coordinator sends a message `bid` to the global coordinator specifying the need for the two schedules in its area. The need for a specific schedule is computed counting the number of passengers in the area that should be picked up by a bus following the schedule (remember that, based on the schedule, the direction of the bus changes and passengers are picked up based on the traveling direction). After receiving the bids from the local coordinators, the global coordinator chooses the schedule for which the need is higher. It specifies to the newly created bus that it should drive according to that schedule, sending it a message of type `schedule` containing the ID of the schedule.

3.6 Stopping buses

We allow a bus to stop when it matches some conditions. In this way, we can reduce the traveling costs whenever it is possible. In particular, a bus can stop when:

- the bus is empty **and**
- there are no passengers waiting in its schedule; here, we consider the number of passengers waiting in the bus stops composing the schedule of the bus that have to be picked up (considering the direction of the bus).

4 Code

For clarity and readability, we divided the code into different files: (i) the file `agent.nls` implements the decision process of one agent: the management of beliefs and intentions; (ii) the file `init.nls` is used to initialize all the local variables of the agents; (iii) the file `execute-intentions.nls` implements the actions related to every intention; (iv) the file `messages.nls` implements some functions to interact with a message, for example the retrieval of the content of the message; (v) the file `utils.nls` implements all the utility functions used by the other files.

5 Tuning

The agent logic consists of a set of parameters that need to be tuned for performance purposes. We try different combinations of these parameters in the training set and in the two test sets, in order to find the best parameter setting. In particular, these parameters are:

- **Stop**: whether we allow the bus to stop or not, as explained in Section 3.6. The tuning showed better results when we do not allow the buses to stop;
- **Initial bus type**: the type of the buses created at the beginning of the simulation (i.e., the scout buses). We set this parameter to 3, consequently the firstly created buses are red;

- **Threshold for creating a new bus**: as we described in Section 3.2, the global coordinator creates a new bus based on the shortage - i.e., the difference between the number of people waiting and the fleet capacity. The shortage is compared with this threshold: when the shortage is higher, new buses are created. We set this parameter to 0, thus as soon as we have a shortage, we need to create new buses;
- **Threshold for the bus types**: when the shortage is higher than the previous threshold, new buses are created. The type of the newly created buses is determined by how the size of the shortage. Thus, we have three different thresholds for the three bus types. We set the threshold for creating a red bus to 60, the threshold for creating a yellow bus to 12 and the threshold for creating a green bus to 0. This means that we create the cheapest bus that can "handle" the shortage situation.

6 Performance

6.1 Improvements

To establish the performance of the final code, we compare the results to a baseline code. This baseline code is our solution developed for the first assignment. In this first version, buses travel according to one fixed schedule, picking up and dropping off passengers. In the final code, we implemented all the improvements explained in Section 3. Table 6.1 shows the comparison between the baseline code and the final code.

Measurement	Baseline	Final Code
Avg travel time	150.01	72.53
Buses' expenses	1039682	711251
Messages sent	0	285
Final amount waiting	232	25
Avg travel time remaining	100.25	78.12
Final avg travel time	156.34	75.48

Table 1: Performance results

Table 6.1 shows that there is a significant decrease of the costs. Not only does the travelling time and the amount of people waiting go down, the financial costs also lower. Solely the number of messages increase, because no communication system was implemented in the baseline version.

These results highlights the advantages of an intelligent coordination and competition between buses. Because of transportation efficiency, travel time and passengers waiting are reduced, without increasing financial costs.

6.2 Performance on test data

Three data sets (*day 1*, *day 2* and *day 4*) are available for testing our transportation system. Each data set contains data for a time span of 24 hours. Each 15 minutes, for each bus stop, the amount of travelers arriving and their destination is displayed. The day 1 data set is used to create our solution, whereas day 4 and day 5 are to test the proposal. In this section, we show the results of our solution for the three different data sets.

The final values of the different metrics for day 1, are shown in the right column of Table 6.1. Figure 1 shows the plot of the expenses, figure 2 the messages send, figure 3 the passengers waiting and figure 4 the travel time.

Day 1 has two rush hours, in the morning and in the evening. The figures show how our solution deals with these rush hourse. First there is a peek in the number of people waiting, followed by a peek in expenses. This is due to the new buses added to manage the rush hour. During the second rush hour, the peek of people waiting is smaller. Because the fleet is larger, the tranportation system can efficiently deal with the rush hour.

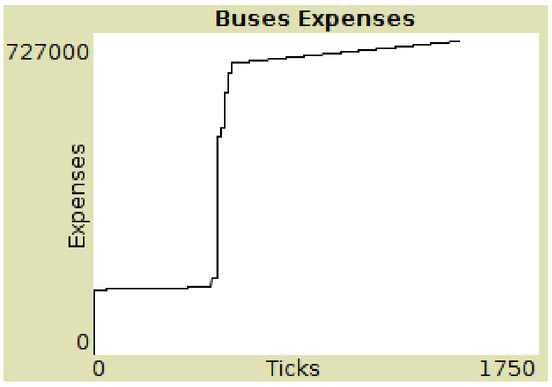

Figure 1: Expenses of the buses in Day 1.

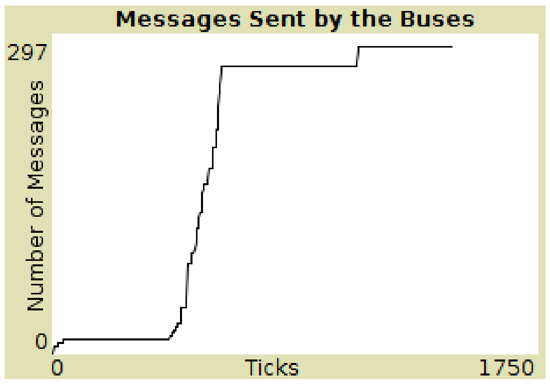

Figure 2: Number of exchanged messages in Day 1.

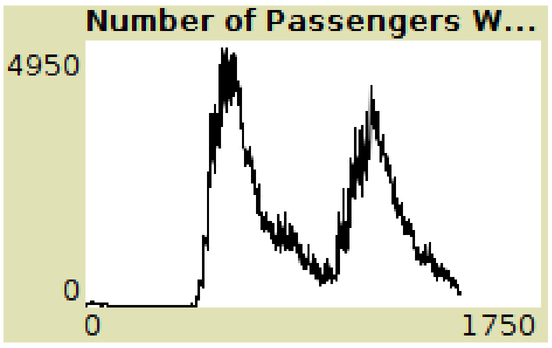

Figure 3: Final number of passengers waiting in Day 1.

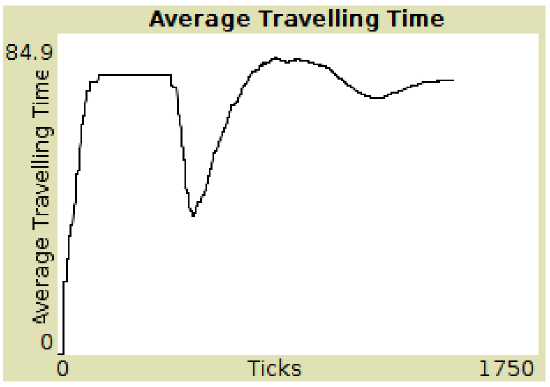

Figure 4: Average traveling time in Day 1.

To measure performance, we check if the system is capable of handling different scenarios. The results of running the system on the training data, is compared to the results for the test data sets. Table 6.2 shows the results available for comparison.

Measurement	Traning Data (Day 1)	Test Data (Day 3)	Test Data (Day 4)
Avg travelling time	72.53	92.36	86.72
Buses' expenses	711251	2393732	2346943.5
Messages sent	285	950	992
Final amount waiting	25	2603	2513
Avg travel time remaining	78.12	182.97	206.91
Final avg travel time	75.48	101.42	95.97

Table 2: Performance results

The system does not perform as well on the test data, as on the training data. The average travelling time only rises a little. But the expenses and the amount of messages are tripled, because much more new buses are added. Even though the fleet was larger, the amount of people waiting in the end is larger.

The plots using day 4 for the expenses, the messages, the number of people waiting and the travelling time are shown in Figure 5, 6, 7 and 8 respectively. Figure 7 shows that the final amount of people waiting on day 4 is larger, even though the bus fleet is larger. It also shows that the rush hour is not properly managed. The amount of people waiting does not go down fast enough.

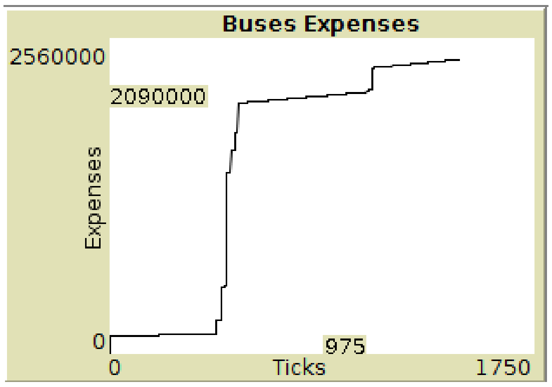

Figure 5: Expenses of the buses in Day 4.

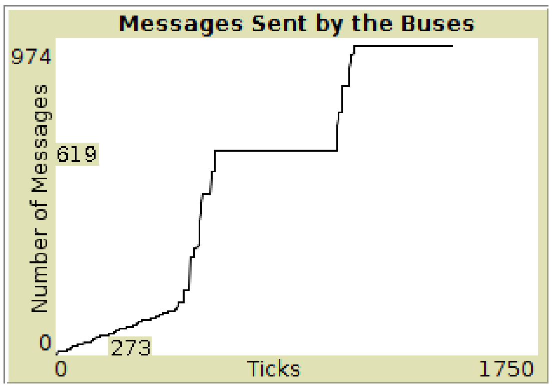

Figure 6: Number of exchanged messages in Day 4.

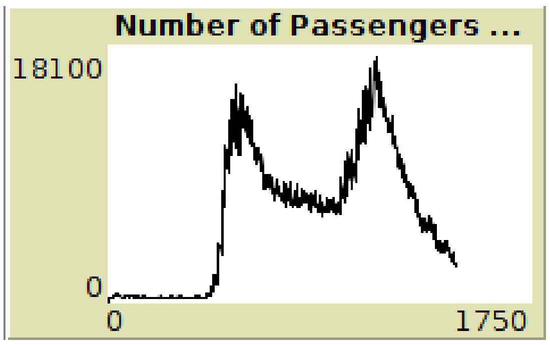

Figure 7: Final number of passengers waiting in Day 4.

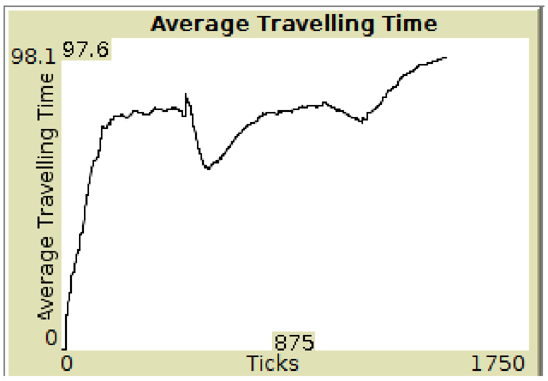

Figure 8: Average travelling time in Day 4.

The plots using Day 5 for the expenses, the messages, the number of people waiting and the travelling time are shown in Figure 9, 10, 11 and 12 respectively.

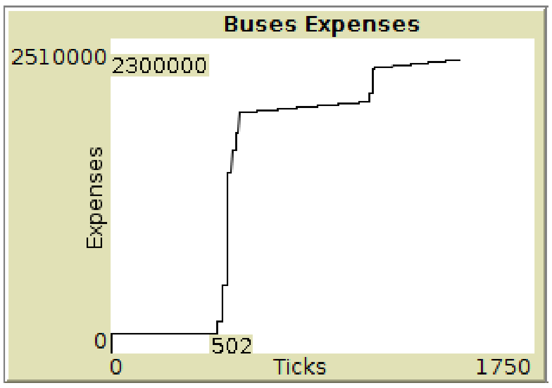

Figure 9: Expenses of the buses in Day 5.

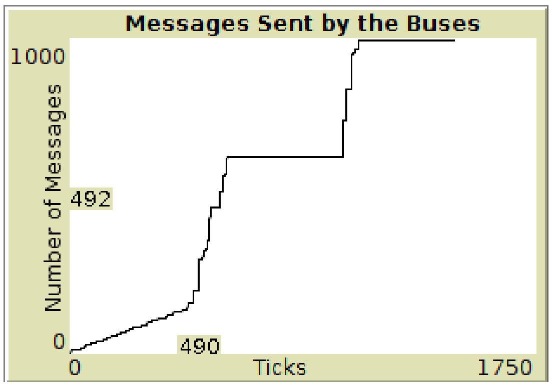

Figure 10: Number of exchanged messages in Day 5.

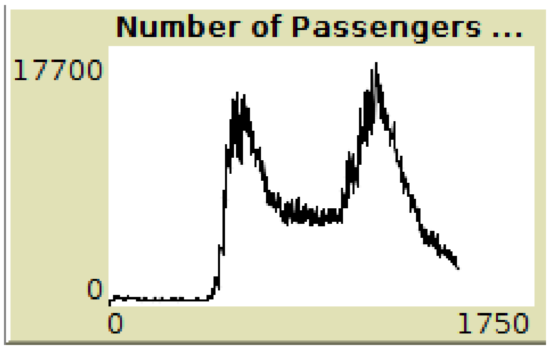

Figure 11: Final number of passengers waiting in Day 5.

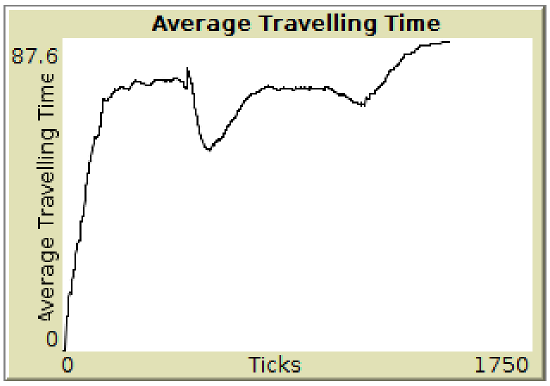

Figure 12: Average travelling time in Day 5.

7 Discussion and conclusion

In this project, we implemented a multi-agents system composed out of a set of buses. The buses interact with each other in order to transport passengers in the city of Amsterdam.

The behaviour of buses is managed through its knowledge about the environment (belief), goals (intentions) and responsibility within the fleet (role). Each bus travels in a specific area of the map, that is associated with a set of bus stops. Furthemore, within the same area buses can move along two different directions. This enables intelligent and efficient management of passengers.

When the number of waiting passengers increase, new buses are created. The local coordinators negotiate the distribution of new buses between areas. When one coordinator faces a shortage of buses, whereas other areas have remnant buses, they cooperate. Local coordinators from different areas support each other by exchanging buses between areas.

This solution improved travel time, expenses and the number of people waiting. It is challenging for transportation companies to reduce expenses, without making worse the travel time and the number of waiting passengers. An intelligent multi-agent bus mechanism could possibly lower both.

Our solution is based on a form of transportation, which entails that buses travel along a fixed route. We suspect that a solution in which travelling is based on the position and destination of the passengers, can lead to improved results. However, such a solution could be difficult to extend and to interpret, whereas a schedule-based solution is particularly easy to improve with new ideas and new behaviours.

For future works, it would be useful to perform a detailed study of the behaviour of our system in different scenarios. This, in order to understand when and why the system is not working adequately. This study could highlight problems in the conceptual model that we designed, and could help to increase the performance of our solution.

References

[1] M. Wooldridge, *An introduction to multiagent systems.* John Wiley & Sons, 2009.

[2] P. Caillou, B. Gaudou, A. Grignard, C. Q. Truong, and P. Taillandier, "A simple-to-use bdi architecture for agent-based modeling and simulation," in *Advances in Social Simulation 2015*, pp. 15–28, Springer, 2017.

[3] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations.* Cambridge University Press, 2008.

[4] P. D. O'Brien and R. C. Nicol, "Fipa—towards a standard for software agents," *BT Technology Journal*, vol. 16, no. 3, pp. 51–59, 1998.