PAMELA : a Generic and Light Multi-Agent Platform

Baudouin Dafflon, Maxime Guériau, Yacine Ouzrout, Sylvain Touchard

PAMELA: a generic and light multi-agent platform

Baudouin Dafflon*, Maxime Guériau†, Yacine Ouzrout‡, Sylvain Touchard‡
*DISP-LAB, Université Claude Bernard Lyon 1, Lyon, France, baudouin.dafflon@univ-lyon1.fr
†LITIS, INSA Rouen, Normandie Univ., Rouen, France, maxime.gueriau@insa-rouen.fr
‡DISP-LAB, Lyon 2, Lyon, France, {firstname.lastname}@univ-lyon2.fr

***Abstract*—Multi-agent frameworks are gaining popularity among the research community as they provide efficient and scalable tools for modelling distributed and/or social systems, enabling to simulate and investigate the behaviour of complex systems in a wide range of applications (such as road traffic, crowd evacuation, disease spreading, etc.). Existing MAS frameworks (such as GAMA, MADkit, JADE, etc.) offer stable and widely used solutions for users that are already experienced with distributed systems simulation or the MAS paradigm. They are also often dedicated (or primarily designed) for a specific application (*e.g.*, MATSim for road traffic). New users can see the learning curve associated with each framework as an obstacle, especially when they lack the theoretical knowledge about computer science or agents and they seek to build and run a very first proof-of-concept simulation. The work presented in this paper results from an effort towards providing more accessible MAS simulation tools, possibly further popularizing their use across different research fields. This paper introduces PAMELA: a novel generic collaborative open-source MAS framework that aims at being light, beginner-friendly, and that allows for fast prototyping through assisted scenario generation and powerful configuration. The tool can work with or without (for faster simulations) the integrated graphical user interface (designed for both testing and visualization). To make is more attractive to new programmers and to enable an easier interfacing with trending machine learning frameworks, PAMELA is entirely written in Python and only relies on standard libraries. This choice makes it possible to make it a multi-platform tool that is easy to deploy and maintain in industry and in the laboratory. After a discussion of existing MAS platform capabilities and limitations, we will present the general design of PAMELA and then we will describe its core components. Through two applications areas, we will show the potential of PAMELA to quickly and easily provide running prototypes that could be used as proof-of-concept simulations before building more complex use cases in the same or a more specific MAS framework.**

***Index Terms*—multi-agent systems, agent-based modelling, open-source, framework, simulation**

I. Introduction

Multi-agent systems (MAS) offer a specific way of simulating the interaction between individuals and with their environment. They are used efficiently to model and investigate phenomena in various areas such as economy, commerce, transport, health, urban zones and, in a more general context, can act as decision support tools. However, although software agents have been part of artificial intelligence techniques for years, their implementation remains based on abstract concepts. The research community has developed a variety of agent-based (or MAS-based) platforms over the past two decades, either for general use or oriented towards a specific domain of use. Since the end of the 90s, a lot of software for agent simulation has been developed. Some frameworks have already been dropped while others continue to be updated. This great variety of platform options leads to a high degree of heterogeneity and to limited compatibility. Finding and choosing the right platform that meets the developer needs and matches the application or the investigated problem features is thus challenging.

Finding an agent platform is generally done on the basis of reputation, recommendation, or based on past experience. To date in the literature, articles on multi-agent systems describe only the basic characteristics of existing platforms without even providing a benchmark of the systems themselves. Our review of the literature on MAS frameworks led to the following observation: the more generic a platform is, the more complicated it is to take in hand for new developers or researchers that are new to MAS programming.Maintainability of algorithms and their evolution becomes a real problem while the amount of prior knowledge required to start working with a framework is increasing. This complexity is an obstacle to the use of MAS and to the adoption for more industrial applications.

The work presented in this paper attempts to mitigate this issue by proposing and sharing a new framework called PAMELA: Python reActive Multi agEnt pLAtform, written in python, free and open-source, which provides a light and robust implementation of agent concepts. Getting started with PAMELA only takes a few minutes, making the framework beginner-friendly and helping to demystify and popularise the use of MAS for research and industrial projects. Before going into the details of PAMELA, this paper offers a review and a classification of existing MAS frameworks from the literature. This classification is based on a selection of objective criteria that helped to discuss the relevance of a new platform.After presenting the implementation details and design choices of PAMELA, we will illustrate possible applications of the framework within two separate use cases that showcase its capabilities and highlight the simplicity of using it.

II. Review of existing Multi-agent frameworks

Among the existing MAS frameworks, only a few appear to be used extensively and supported by the research community. This section focuses on the platforms that are the most represented in the literature, by excluding frameworks that are still in an experimental state, abandoned, or distributed confidentially.

A. Comparison criteria

Our review of existing MAS frameworks gives a particular focus on beginner-friendly features, as reflected by the following list of criteria, used for comparison and discussion:

- **Usability**: describes the type of users for which the platform is primarily designed for.
- **Operational capability**: refers to the suitability of the platform to easily implement and use various MAS properties (such as self-organization, communication/coordination, adaptation/learning, etc.).
- **Configurability**: the quality allowing the system behaviour to be varied by a small amount of user input.
- **Stability**: refers to issues of stability during simulation and the ability of the tool to maintain the same behaviour under high load or during scaling.
- **Versatility**: the fact a given framework is generic enough in its design to allow for a wide scope of applications and scenarios.
- **Extensibility**: the quality for a MAS plateform of being designed to enable the addition of new functionality.

B. Comparison of existing frameworks

1) JADE [2]: (Java Agent DEvelopment Framework[1]) is a software framework implemented in Java. Though the user requires prior knowledge on multi-agent systems and agent-based simulation, it simplifies the implementation of MAS through a middle-ware that complies with the FIPA specifications [13] and through a set of graphical tools that support the debugging and deployment phases. A JADE-based system can be distributed across machines (which not even need to share the same OS) and the configuration can be easily controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another, as and when required. JADE is completely implemented in Java language and the minimal system requirement is the version 5 of JAVA. JADE is free software and is distributed by Telecom Italia, the copyright holder, in open source under the terms and conditions of the LGPL license, allowing it to be extended by expert users. The project however does not seem to be updated since 2017[1].

2) JADEX [3]: Jadex is an agent system that follows the Belief Desire Intention (BDI) model [5] and facilitates easy intelligent agent construction due to software engineering foundations. It allows for programming intelligent software agents in XML and Java. The platform design was initiated by the Distributed Systems and Information Systems Group of the Hamburg University, and the tool is available under GPLv3 license. Although there is documentation and examples, the handling can be complex for novel users as JADEX is presented as an extension for agent middleware and knowledge about an other agent platform (*e.g.*, JADE) is required. The BDI paradigm also limits the scope of agent application that can be specifically designed using JADEX.

3) Janus [8] and SARL [14]: Janus is an open-source multi-agent platform fully implemented in SARL[2] (relying on Java for its versions 1 and 2). Janus enables developers to quickly create web, enterprise and desktop multiagent-based applications. It provides a comprehensive set of features to develop, run, display and monitor multiagent-based applications. Janus-based applications can be distributed across a network. Janus could be used as an agent-oriented platform, an organizational platform, and/or an holonic platform. It also natively manages the concept of recursive agents and holons.

Janus platform was initially published during the 2007-2008 period as a pure Java framework. Since 2014, Janus is fully reimplemented to support the SARL Agent-Oriented Programming Language. And since 2020, it is fully re-implemented using the SARL language. It is Licensed under the Apache License. The extended capabilities of Janus makes it a powerful and stable tool for expert users but could drastically discourage users that are new to agents or multi-agent programming.

4) MADKIT [10]: The MadKit platform was developed in 1998 at the LIRMM (Montpellier) by Olivier Gutchneck during his thesis and Jacques Ferber his supervisor. Thereafter, under the supervision of Jacques Ferber, Fabien Michel took over the development of the platform and integrated, in particular, TurtleKit a StarLogo-like. MaDKit agents interpret one or more roles. They belong to groups and create artificial communities. MaDKit provides general functionality for agents, such as lifecycle management, message distribution, and allows great flexibility in agent architectures. Although there is a documentation, the first steps with MaDKit do not seem to be accessible to a beginner. A computer scientist should be able to take it in hand easily. MaDKit-5 is free software licensed under CeCILL-C, a french licence close to LGPL, unknown in software industry.

5) Gama [17]: GAMA is an agent platform which aims at providing a complete modeling and simulation development environment for building located multi-agent simulations. It provides capabilities concerning data visualization, GIS data management, and multi-level modeling. GAMA programming is based on GAML, agent-oriented language, with an optional graphic user interface to support rapid prototyping. It has been first developed by a Vietnamese-French research team until 2010. Now, it is developed by a academic consortium. This is a free software (distributed under the GNU GPL v3 license). This platform is very easy to learn thanks to many tutorials and videos. However, it is much more complicated to understand its architecture to make it fit your specific need such as non localized simulation.

6) NetLogo [16]: NetLogo was designed by Uri Wilensky, in the spirit of the programming language Logo, to be ”low threshold and no ceiling”. It teaches programming concepts using agents in the form of turtles, patches, links and the observer. NetLogo was designed for multiple audiences in mind, in particular: teaching children in the education community, and for domain experts without a programming background to

[1]https://jade.tilab.com/

[2]SARL website: http://www.sarl.io/

TABLE I
SUMMARY AND COMPARISON OF EXISTING MAS FRAMEWORKS

Framework	Usability	Operational capability	Configurability	Stability	Versatility	Extensibility
JADE [2]	MAS experts, expert programmers (Java)	FIPA compliant, state-of-the-art agent technologies	Remote GUI, runtime changes	Scalable and distributed	Generic middle-ware	Requires a solid experience with the framework
JADEX [3]	MAS experts, expert programmers (Java)	BDI paradigm	Same as JADE	Same as JADE	Limited to BDI agents	Already based on JADE
Janus [8] and SARL [14]	MAS users and advanced programmers	state-of-the-art agent technologies with holonic and recursive agent representations	Agent-Oriented Programming Language	Actively maintained and improved[3]	Various applications[4]: transportation, games, etc.	Extensible agent behaviours(using Capacities/Skills)
MADKIT [10]	Beginners in MAS/agents, advanced programmers	Cellular automata and grid-like environments	Basic GUI	Not suited for large-scale simulation	Limited by the environment	Requires in-depth core changes
Gama [17]	Advanced knowledge of MAS/agents, advanced/expert programmers	Specialized in located agents and simulations	GAML agent-oriented language and declarative 3D GUI	Suited for large scale agent-based simulation, large base of tutorials and demo videos	Compatible with GIS, has several plug-in available (traffic, pedestrians, etc.)	Integrated plug-in development pipeline
NetLogo [16]	Beginners in MAS/agents and programming	Limited to educational or fast prototyping	Basic GUI	Popular in education and research	Limited to small-scale or simplified models	Requires in-depth core changes
MATSim [11]	Advanced knowledge of MAS/agents, advanced/expert programmers	Large-scale agent-based transportation simulation	Advanced demand modelling	Suited for large scale agent-based simulation	Dedicated to transportation applications	Modular architecture
PAMELA (this paper)	Beginners in MAS/agents, intermediate programmers	state-of-the-art agent technologies	Interactive GUI, Json configuration files, scenario generator	Actively maintained and improved[5]	Various applications: transportation, epidemiology	Modular framework based on the REST API

model related phenomena. Many scientific articles have been published using NetLogo [1], [15].

The NetLogo environment enables exploration of emergent phenomena. It comes with an extensive models library including models in a variety of domains, such as economics, biology, physics, chemistry, psychology, system dynamics. NetLogo allows exploration by modifying switches, sliders, choosers, inputs, and other interface elements. Beyond exploring, NetLogo allows authoring new models and modifying extant models. NetLogo is very popular in the education and research community. NetLogo is also free software licensed under GPL Licence.

7) MATSim [11]: MATSim provides a framework to implement large-scale agent-based transport simulations. The framework consists of several modules which can be combined or used stand-alone. Modules can be replaced by custom implementations to test single aspects of your own work.

Currently, MATSim offers a framework for demand-modeling, agent-based mobility-simulation (traffic flow simulation), re-planning, a controller to iteratively run simulations as well as methods to analyze the output generated by the modules. MATSim is developed in Java under GPL v2 licence.

C. Summary and discussion

We reviewed existing MAS frameworks that appeared to be representative of the state of the art.NetLogo seems to be the perfect start for a user, programmer or researcher who wants to learn about the multi-agent systems paradigm by designing and prototyping his very first application. However, moving to more complex applications can be quickly limited and more powerful/extensive tool are then required. Expert users can legitimately go for JADE or Janus, although the learning curve could be seen as an important obstacle. Finally, a proportion of users will naturally be driven towards GAMA and MATSim, mostly depending on the initial application they are working on. MATSim will particularly suit for traffic and transportation-related applications while GAMA can be more versatile while interesting programmers who work with GIS data.

[3]SARL GitHub repository: https://github.com/sarl/sarl

[4]Example of projects built using SARL: http://www.sarl.io/community

[5]PAMELA GitHub repository: https://github.com/BDafflon/PAMELA-DEV

After testing and evaluating these platforms on several types of projects: Transportation Simulation, Product Design Optimization and Human Society Simulation, we noticed that each of them has shown advantages and disadvantages, as highlighted in Table I. None of them allowed to realize the projects without a huge technical and technological investment. We have noticed several obstacles to the adoption of these existing platforms by the general public and to their introduction in the industrial sector. Among these obstacles, we can cite the great complexity of these projects from a software and architecture point of view. Adapting a tool to customized problems is not accessible to a beginner or a technician. In the operational aspect, these platforms are very robust and offer plug-ins to help design simulation and agent applications. We have encountered few stability or scalability problems. Nevertheless, their deployment on exotic architectures (RPi for example) is complicated or impossible.

In this context, we started the reflection and design of an ultra light MAS or agent-based platform that provides the main functionalities of MAS while being accessible to beginners. For reasons of practicality and maintenance, we have opted for the python language. Indeed, since a few years, python has become the most used language on GitHub and presenting the most documentation on public search engines. In the following section, we present the architecture and functionality of PAMELA: Python reActive Multi agEnt pLAtform. PAMELA intends to offer an alternative MAS-based simulation framework to intermediate programmers and users new to MAS that can be used for fast prototyping, through an integrated GUI and a simplified scenario definition, while showing advanced extensibility capabilities thanks to its modular architecture. The capabilities of PAMELA are illustrated when presenting some of the existing modules that are dedicated to non-computer scientists users.

III. PAMELA FRAMEWORK

A. Global overview

In the previous section, we observed a great heterogeneity between existing MAS frameworks. It is the same for agent formalism, communication models and architectures. In PAMELA, we have left this choice to the user by providing the basic concepts. Each agent evolves in an environment, perceives it and is able to act on it. Interfaces can be used as is or overloaded to adapt to the user's level or the targeted application.

B. Structure

The PAMELA platform is built around this concept. As shown with Figure 1, four components are required:

- an environment manager
- an entity collection (Agent, item, object, etc.)
- a GUI
- a scenario manager

We will see in a following part that agents also provide web-services that help them to be easily inter-operable.

PAMELA is a set of Python classes that implements these components, libraries, messaging function, etc. The platform also offers a scenario generator and a graphical editor that allow to configure a simulation without writing code. The basic philosophy of the architecture is to use the platform as much as possible to add new features. The small size of the environment manager, combined with the principle of modular services, enable advanced users to add new functionalities and to adapt or propose custom agent models.

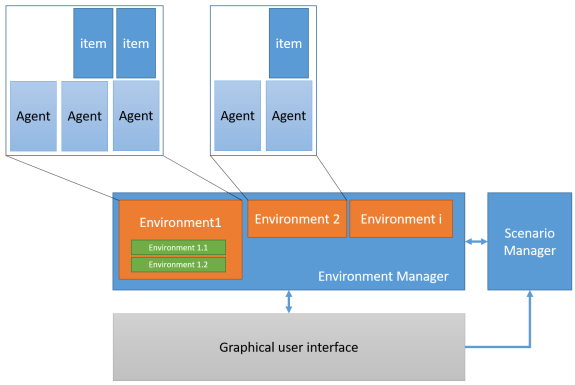

Fig. 1. Architecture diagram of PAMELA

C. Execution of agents

The execution cycle of an agent (Figure 2) is based on the principles defined by Ferber [7] and Wooldrige [18].

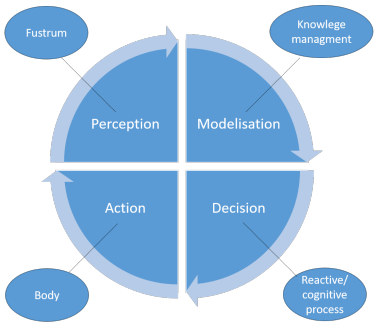

Fig. 2. Agent life cycle

The environment manager executes this cycle at each time step. It calculates perceptions according to the defined agent's frustrum (*i.e.*, its field of view), then collects all agents' decisions and finally applies them taking in account physical constraints of each agent body (*i.e.*, the agent physical representation).

The implementation of a standard agent is done by defining the following functions:

- Creation function (init): this function allows the user to define a frustrum, a body, and the parameters required for the agent's decision making. The agent's body and frustrum can be customized classes or instances of default classes (*e.g.*, circular, conical frustrum, etc.).

- Display function (getEditable): it summarizes the information to be displayed for the log or the GUI.
- Decision function (update): the result of the decision taken by the agent given its current perception, under the form of an Action (or None).

D. Agents configuration

The agents configuration can be done by an overload of the ”Agent” class. It can also be done through a configuration file. The platform can also be controlled by defining all necessary parameters in a scenario file (or by using the editor), for faster prototyping or for beginners. In both cases, a JSON configuration file is required.

```
def __init__(self, f=1):
    Agent.__init__(self)
    self.body = StadardAgentBody()
    self.body.frustrum = Circularfrustrum(5)

def getEditable(self):
    return ['id']

def update(self):
    action = AnimateAction()
    action.move = self.moveRandom()
    return action
```

Fig. 3. Example of a random agent configuration

The structure of this file is described in Figure 3. The first part (Start) describes the state of the simulation before the launching the simulation. The second part (Simulation) rules the execution of the simulation at each time step. Each element defined in this block will be created in the simulation according to a timestamp that takes the beginning of the simulation as a reference.

In each of these two blocks, agents and objects can be described according to the structure illustrated in the example given in Figure 4.

```
{
        "entity":"object",
        "name":"Grass",
        "type":"Grass",
        "aabb":[
           0,
           0,
           200,
           200
        ],
        "customArgs":{
           "orientation":180
        }
}
```

Fig. 4. Example of an entity description

In this example, an object of type ”Grass” will be added at the beginning of the simulation at position [0,0] and with an orientation of 180 degrees. The particularity of this file is that all the elements present in the ”customArgs” block will become attributes of the Grass class. The description of an agent follows the same rule.

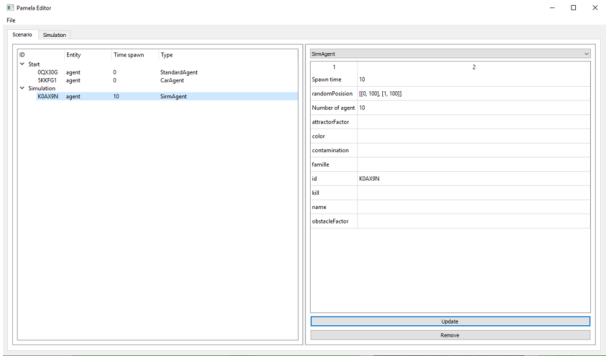

Fig. 5. Configuration GUI available in PAMELA

Creating a file like this can be complicated, especially if the simulation becomes very large. An embedded tool (Figure 5 allows to do it in a graphical way.

E. Servisization / Observer

As mentioned previously, PAMELA relies on a set of classes that can be directly used and/or overloaded. Among these classes, the RESTEnvironment class proposes a REST API to interact with PAMELA as a server. It is possible to retrieve the list of agents, their states or to relocate the decision. REST endpoints are available after a JWT identification. The endpoints allow to build and update the environment (creation, modification, deletion) and the items (object and agent). Thanks to this system, it becomes easy to build a webview or a custom GUI. RESTEnvironment allow to use heavy cognitive decision processes such as neural networks, ontologies, etc., by decentralizing the calculation. The PAMELA platform is modular, so that the user has the choice to use the classical environment or a REST environment. The classical environment is included in the REST environment which allows to deploy a hybrid approach.

F. Conclusion

In this part, we have presented the functioning of PAMELA from a user point of view, whether he is a computer scientist or a simple user. The configuration, simplified by the editor, makes it a first gateway to agents and MAS simulation. We also saw that it was possible to customize the different components of PAMELA using simple inheritance. In the same way, servitization allows to add a layer of complexity in the decision making mechanisms.

Like all modern platforms, the functions of messaging, metrics, etc., are available in PAMELA. A number of modules have been developed around the framework, such as:

- city and urban road network environment generator,
- environment import system from an image,
- waypoint and pheromone mechanism,
- webview based on webGL, etc.

Next sections illustrate the capabilities of PAMELA in two different applications.

IV. Use case 1: COVID SIR simulation

A. Global overview

The global health crisis of the Covid-19 Coronavirus has demonstrated the role of modelling in political and health decision making. A classic model in the literature for decision support is the "Compartmental models" such as the SIR model [4] (Susceptible, Infectious, or Recovered). For a given population, we study the size of three sub-populations over time: $S(t)$ represents the susceptible individuals at time t, $I(t)$ the infected individuals, and $R(t)$ the removed individuals; $N = S(t) + I(t) + R(t)$ then represents the total constant population over time. It is important to differentiate between healthy and removed individuals: healthy individuals have not been affected by the virus, whereas removed individuals are cured and therefore considered immune. In other words, withdrawn persons are no longer taken into account. This model can be studied by solving the following system:

$$\begin{cases} \frac{dS(t)}{dt} = & -\beta S(t)I(t) \\ \frac{dI(t)}{dt} = & \beta S(t)I(t) - \gamma I(t) \\ \frac{dR(t)}{dt} = & \gamma I(t) \end{cases}$$

The advantage of this type of model is to be able to calculate and study the population dynamics very quickly. However it is very difficult to simulate complex behaviours. One possibility is to multiply the number of classes. The following simulation, relying on the underlying MAS provided by PAMELA, has common points with the SIR model (3 sub-populations: healthy, infected, withdrawn) but also major differences: it is discrete (each point represents one person) and random (in order to model the movements of people and their potential contacts). It allows to see more precisely the impact of the modification of the parameters on the evolution of the virus thanks to the definition of different types of (agent) behaviours and interaction between populations.

B. Environment

This simulation can be done using a reactive multi-agent system. In this type of simulation, agents environment is the corner stone of the approach. It links the population with the decision process mechanism.

The virtual environment is composed of:

- **Obstacles**: when perceived by the agent frustrum, they are transmitted and placed in the virtual environment. Each obstacle is represented by an aggregation of repulsive spots. These entities does not show a specific behaviour. In addition a spot belt mark the environment boundary corresponding to the simulation perception limits.
- **Building** (housing, infrastructure, etc.): They represent points of interest for the simulation such as places of mixing like schools or bottlenecks like hospitals. They influence the decisions of agents according to their class.
- **Agents**: A population of agents dispersed in the environment at the beginning of the simulation. A random draw allows to assign them a class (S or I) according to the parameters of the simulation.

C. Agents

The behaviour of the agents looks like a finite state machine (Figure 6). The change from one class of the SIR model to

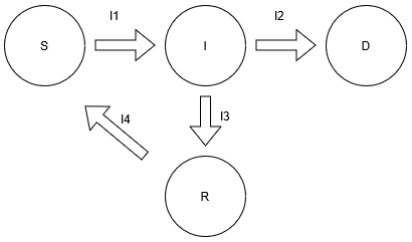

Fig. 6. Agent behaviour

another is done according to the following rules:

- I1: To pass from class "S" to "I", it is necessary to have in the frustrum of the agent "S" an agent belonging to class "I". However, this is not automatic. A probability based on [12] is applied
- I2: For the agent "I" to die, a probability according to the internal states is applied [12]
- I3: The move from "I" to "R" is only based on time. After a certain time, an "I" agent will automatically join the "R" class if he is not dead.
- I4: As immunity is not permanent, after a certain period of time, the agents of the "R" class return to the "S" class.

The interactions between agents are also simplified for the needs of the example: following laws of attraction and repulsion inspired by Newtonian physics, agents "S" and "R" move randomly in the environment seeking to satisfy their desires (go to the supermarket, to work and to their homes). They avoid the "I" agents.

Class "I" agents follow the same desires for the first few days and then confine themselves to their homes when the symptoms appear. Class "D" agents do nothing and are removed from the simulation.

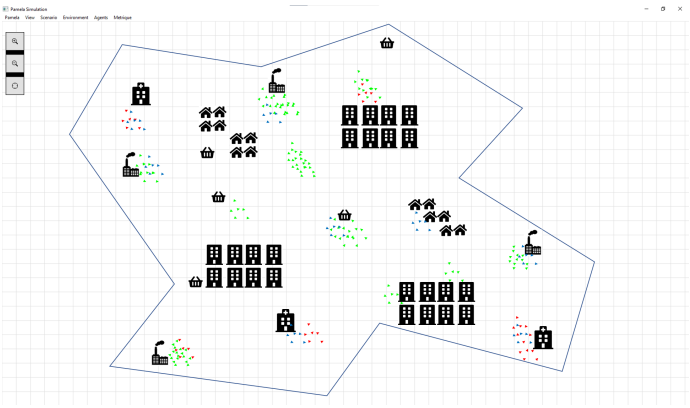

Fig. 7. PAMELA View: SIR Simulation. Green: susceptible, Red: infectious

A screenshot of this simulation is depicted in Figure 7.

D. Results

Two question arose from this very simplified example: can we find the same results as with an analytical solution? and was the implementation easier thanks to the platform?

In the first case, by measuring the population of each class in real time, we obtain results of the same order of magnitude (Figure 8). The implementation was fast and the abstraction

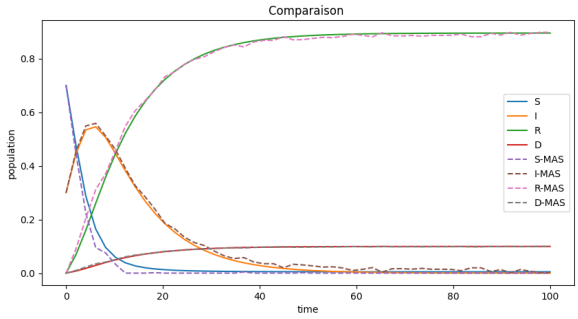

Fig. 8. Comparison

of the platform and the flexibility of the language allowed to have results after a few minutes.

V. Use case 2: Mobility-on-Demand optimisation

A. Global overview

Mobility-on-Demand (MoD) systems offer a flexible mobility alternative to classical public transportation services in urban areas. However, a significant part of MoD vehicles operating time can be spent waiting empty or driving to reach new potential ride requests. Improving vehicle fleet operation is an extremely challenging problem, as the number of vehicles in operation at a time cannot be controlled. To cope with this issue, new forms of mobility are being deployed successfully: for instance, ride-sharing enabled MoD systems can match riders from several requests. MAS have been shown to be particularly suited to model vehicles and riders interaction finely in MoD systems [9].

B. Environment

The problem of rider-driver assignment (or rider-vehicle assignment in the case of Autonomous MoD) can be described in a multi-agent environment. Here agents are vehicles and riders, both trying to fulfil their own goals. The MoD booking platform allows vehicles to access pending requests from riders, and unlike traditional MoD systems, this information is shared with all riders. This framework is described in the UML diagram depicted in Figure 9.

C. Agents

1) Riders: A rider i is defined by the vector $R_i : O_i, D_i, P_i, T_i, J_i, S_i | fc_{it}$ where O_i is its origin (where the request is created), D_i its destination, P_i its field of view, T_i time of the request creation,J_i denotes its impatience, S_i its current state and fc_{it} is an internal function that drives

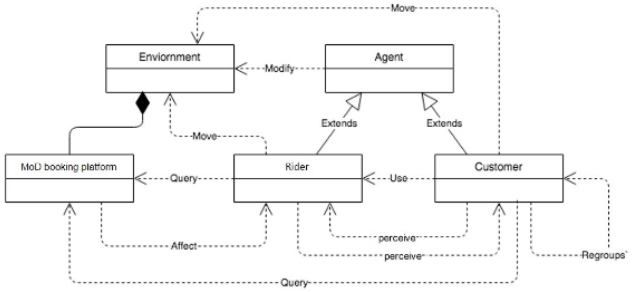

Fig. 9. Environment for MoD

the agent decision. f_{it} uses all the other agent parameters to minimize the waiting time and the cost of the trip. To remain active in the environment, each rider agent must keep an impatience gauge be low 100 %, otherwise, we assume that the client leaves the system (completing its trip using an alternative transportation mode) and his request is recorded as unserved. Riders tend to exhibit a gregarious behaviour: each rider tries to minimize is own waiting time and the cost of the journey by moving (walking) in the environment to form groups of riders.

D. Driver

Each vehicle V_j is defined by a field of view P_j, a working time T_j and the number of free seats C_j. Similarly to riders, each vehicle has an internal function called $f_{d_{jt}}$ that traduces its objective of maximizing profit: $V_j : P_j, T_j, C_j | f_{djt}$. Each vehicle tries to pick-up more passengers and attempts to avoid travelling empty. The behaviour of a vehicle can be divided into two phases: waiting (vehicle is empty waiting for new requests) and traveling (with at least one rider on board).

E. Interactions

Agents interaction are inspired by physics:

- Rider-rider interaction: this interaction is modeled as a simple linear attraction force
- Rider-vehicle interaction: To reduce the waiting time of riders,we enable them to walk to a location where he is more likely to be pick-up by a vehicle, by meeting more riders with a similar destination, hence making the newly formed group more attractive to the MoD system vehicles. When a vehicle passes (or plans to drive) near a potential rider, and goes in the same direction, the MoD system sends a notification containing the coordinates of the meeting point (on the current trajectory of the vehicle, where riders should meet). This meeting point is defined taking into account the capacity of the vehicle and its dynamics.
- Vehicle-vehicle interaction: for empty (waiting) vehicles, this interaction acts as a repulsive force computed through a repulsion force
- Vehicle-rider interaction: vehicles tend to be attracted by riders, and the intensity of this attraction is weighted by the number of riders going to a similar destination

and located close to each other (for instance, forming a micro-community). All riders (even individually) impact vehicles trajectories.

F. Results

This second implementation example is more complex than the COVID SIR simulation (use case 1). The interactions between agents are more complicated, and so is the scenario. Thus, it is difficult to compare it to an analytical model because of the nature of the problem. However, its development with PAMELA was very fast. The technical points that could have been blocking was the link with the database of the requests, which was solved thanks to the library proposed by Python. The interested reader can refer to [6] for a more detailed overview of the scenario, set-up and detailed results of this study.

VI. Discussion and conclusion

Multi-agent systems and intelligent agents are particularly suited to model the behaviour of complex and distributed systems. Researcher and programmers usually rely on existing simulation frameworks (such as GAMA, JADE or MADkit) to leverage MAS properties for tackling problems in various application areas. However, users that are new to these platforms can see the learning curve associated with each framework as an obstacle, especially when they lack the theoretical knowledge about computer science or agents and they seek to build and run a very first proof-of-concept simulation. The work presented in this paper results from an effort towards providing a more accessible MAS simulation tool, possibly further popularizing their use across different research fields. This paper introduced PAMELA: a novel generic collaborative open-source[6] MAS framework that aims at being light, beginner-friendly, and that allows for fast prototyping through assisted scenario generation and powerful configuration. To make it more attractive to new programmers and to enable an easier interfacing with trending machine learning frameworks, PAMELA is entirely written in Python and only relies on standard libraries.

We showed in the experiments that PAMELA can be easily and quickly tuned and configured for very different applications: an infection spreading model and a mobility-on-demand system optimization. Different types of agents can be defined and their interaction can be controlled by the system designer, before running a simulation or at runtime. PAMELA also provides different modules (such as metrics computation, scenario generation, environment generation) that enable a fast prototyping of MAS algorithms and models while giving a first overview of simulation results.PAMELA also aims, by being released as an open-source[6] project, to be maintained and extended, thus further benefiting from the experience of the MAS community. We hope that PAMELA will be useful and helpful in an academic and industrial context, providing a framework that allows for a fast prototyping of MAS solutions, that could either be further developed in the same or an other framework for more complex scenarios or more detailed simulations.

[6]PAMELA GitHub repository: https://github.com/BDafflon/PAMELA-DEV

References

[1] José Barbosa and Paulo Leitão. Simulation of multi-agent manufacturing systems using agent-based modelling platforms. In *2011 9th IEEE International Conference on Industrial Informatics*, pages 477–482. IEEE, 2011.

[2] Fabio Bellifemine, Giovanni Caire, Giosuè Vitaglione, Giovanni Rimassa, and Dominic Greenwood. The jade platform and experiences with mobile mas applications. In *Software Agent-Based Applications, Platforms and Development Kits*, pages 1–20. Springer, 2005.

[3] Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. Jadex: A bdi-agent system combining middleware and reasoning. In *Software agent-based applications, platforms and development kits*, pages 143–168. Springer, 2005.

[4] Yi-Cheng Chen, Ping-En Lu, and Cheng-Shang Chang. A time-dependent sir model for covid-19, 2020.

[5] Francisco JP Cunha, Marx L Viana, Tassio Ferenzini Martins Sirqueira, Marcio Rosemberg, and Carlos Lucena. Understanding normative bdi agents behavior. In *SEKE*, pages 221–220, 2018.

[6] Baudouin Dafflon, Maxime Guériau, Yacine Ouzrout, and Ivana Dusparic. Emergent micro-communities for ride-sharing enabled mobility-on-demand systems. In *11th Workshop on Agents in Traffic and Transportation (ATT 2020) held in conjunction with ECAI 2020*, 2020.

[7] Jacques Ferber. *Les systèmes multi-agents: vers une intelligence collective*. InterEditions, 1997.

[8] Nicolas Gaud, Stéphane Galland, Vincent Hilaire, and Abderrafiâa Koukam. An organisational platform for holonic and multiagent systems. In *International Workshop on Programming Multi-Agent Systems*, pages 104–119. Springer, 2008.

[9] Maxime Gueriau, Federico Cugurullo, Ransford A. Acheampong, and Ivana Dusparic. Shared autonomous mobility-on-demand: Learning-based approach and its performance in the presence of traffic congestion. *IEEE Intelligent Transportation Systems Magazine*, 12(4), 2020.

[10] Olivier Gutknecht and Jacques Ferber. Madkit: A generic multi-agent platform. In *Proceedings of the fourth international conference on Autonomous agents*, pages 78–79, 2000.

[11] Andreas Horni, David Charypar, and Kay W Axhausen. Variability in transport microsimulations investigated with the multi-agent transport simulation matsim. *Arbeitsberichte Verkehrs-und Raumplanung*, 692, 2011.

[12] Nancy HL Leung, Daniel KW Chu, Eunice YC Shiu, Kwok-Hung Chan, James J McDevitt, Benien JP Hau, Hui-Ling Yen, Yuguo Li, Dennis KM Ip, JS Malik Peiris, et al. Respiratory virus shedding in exhaled breath and efficacy of face masks. *Nature medicine*, 26(5):676–680, 2020.

[13] Stefan Poslad and Patricia Charlton. Standardizing agent interoperability: The fipa approach. In *ECCAI Advanced Course on Artificial Intelligence*, pages 98–117. Springer, 2001.

[14] Sebastian Rodriguez, Nicolas Gaud, and Stéphane Galland. Sarl: a general-purpose agent-oriented programming language. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 3, pages 103–110. IEEE, 2014.

[15] Zhenjiang Shen, Xiaobai A Yao, Mitsuhiko Kawakami, Ping Chen, and Masahito Koujin. Integration of mas and gis using netlogo. In *Geospatial Techniques in Urban Planning*, pages 369–388. Springer, 2012.

[16] Elizabeth Sklar. Netlogo, a multi-agent simulation environment. *Artificial life*, 13(3):303–311, 2007.

[17] Patrick Taillandier, Duc-An Vo, Edouard Amouroux, and Alexis Drogoul. Gama: a simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 242–258. Springer, 2010.

[18] Michael Woolridge and Michael J Wooldridge. *Introduction to multiagent systems*. John Wiley & Sons, Inc., 2001.