








A Comparative Analysis of Multi-agent Simulation Platforms for Energy and Mobility Management

Aliyu Tanko Ali¹(✉) , Martin Leucker¹ , Andreas Schuldei¹ ,
Leonard Stellbrink² , and Martin Sachenbacher¹ 

¹ Institute for Software Engineering and Programming Languages,
University of Lübeck, Ratzeburger Allee 160, Lübeck, Germany
{aliyu.ali, leucker, andreas.schuldei, sachenbacher}@isp.uni-luebeck.de

² Institute for Multimedia and Interactive Systems, University of Lübeck,
Ratzeburger Allee 160, Lübeck, Germany
leonard.stellbrink@uni-luebeck.de

Abstract. Effective energy and mobility management benefits from multi-agent simulations (MAS) to model complex interactions among various agent types. Selecting the optimal MAS platform to implement and simulate these interactions is vital for achieving accurate results, scalability to realistic problem sizes, and efficient computational performance. This paper investigates the energy and mobility domain and identifies key parameters such as the number and complexity of agents, parallel computing power, CPU requirements etc., for developing MAS in the context of this domain. It then presents a comprehensive evaluation of various MAS development platforms. Using a multi-level selection and elimination approach, we narrowed down our evaluation to two final candidates. We then implemented key aspects of our model in both platforms to compare them in terms of practical relevance. Our findings reveal that the Agents.jl platform outperforms the Mesa platform in terms of runtime performance, has a smaller memory footprint for large numbers of agents, and offers scalability, making it the most suitable choice for developing MAS for integrated energy and mobility models.

Keywords: Multi-agent simulation · Agent-based models · Mobility transition · Car-sharing

1 Introduction

The shift from internal combustion engine vehicles to battery-powered electric vehicles (EVs) is driving a mobility transition aimed at promoting public transport, car-sharing, and sustainability [13, 14]. Car-sharing users play a crucial role in this transition, as they need to adapt their behavior to new circumstances like

This project was funded by the state of Schleswig-Holstein, Germany.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
V. Malvone and A. Murano (Eds.): EUMAS 2023, LNAI 14282, pp. 295–311, 2023.
https://doi.org/10.1007/978-3-031-43264-4_19

limited range and necessary charging stops, while also the limitations of the electric power grid must be considered. Electric car-sharing providers should learn when to charge cars to maximize renewable energy use and ensure car's availability. This situation creates a need for AI-supported forecasting, peak shaving, and recommendation to dynamically adapt car-sharing behavior [27].

Our ongoing project, "Multi-Agent Simulation of Intelligent Resource Regulation in Integrated Energy and Mobility" (MASIRI)¹, aims to create a multi-agent simulation using intelligent agent modeling based on the psychological behavior of electric car-sharing users. Particularly, we investigate the influence of human experience and behavior on mobility and energy use in vehicle-to-grid (V2G) systems, and how this knowledge can optimize system design.

Multi-agent simulation is a computational technique that simulates the behavior and interactions of multiple agents within an ecosystem [4]. These simulations are built on the principles of agent-based models (ABM) [6], which focuses on representing individual agents and their behaviors to study the emergent properties of a system. The simulation environment provides a platform for agents to interact, communicate, and potentially collaborate or compete with each other. Agents can exchange information, share resources, coordinate their actions, and influence the behavior of other agents.

Various platforms have emerged that aid in the development of MAS in research and industrial contexts [23, 25]. However, selecting a MAS development platform suitable for a given scenario is a difficult task, as there is no universally agreed-upon set of criteria for ranking and evaluating these platforms. Researchers rely on using semi-structured techniques, including questionnaires, to compare platforms [3, 9, 29]. Some studies compare different platforms [10, 18, 22] and evaluate specific requirements like strengths, performance, and code complexity [7, 25]. However, many of these studies are outdated, some platforms are not maintained, and they do not cover our use case area. Several works in the literature, including [8, 11, 21, 31], and [24], have employed agent-based models to explore different scenarios within the realm of electromobility. However, the majority of these studies primarily focus on analyzing EV users and their charging behavior, as well as the role of EVs as a new form of urban mobility. The development of a comprehensive MAS of electromobility in the context of car-sharing and V2G has received comparatively less attention. In this study, we carefully curated a collection of platforms from multiple sources, including diverse projects, comparative studies, and online searches. The primary objective of this extensive curation process was to identify the platform that best aligns with the specific requirements of our project.

The contributions of this paper are as follows: First, it delineates the requirements of MAS for the energy and mobility domain, with a specific focus on a particular use case. Next, it then presents an updated and comprehensive overview of various MAS platforms. Employing a multi-level selection and elimination approach, we narrow down the options from multiple platforms to two final candidates. Subsequently, we implement key aspects of our model in both platforms

¹ <https://www.imis.uni-luebeck.de/en/forschung/projekte/masiri>.

to enable a practical and relevant comparison. Based on the evaluation and comparison, we identify and select the most suitable MAS platform for our specific research purposes.

The remainder of the paper is organized as follows: Sect. 2 introduces MAS for integrated energy and mobility and identifies the resulting requirements for MAS development. Section 3 studies various MAS development platforms, Sect. 4 explains the results from implementing our use case scenario and their evaluation, and finally, Sect. 5 presents the conclusion.

2 MAS for Integrated Energy and Mobility

In our project MASIRI, the model aims to simulate the current and future energy usage and the mobility behavior of the inhabitants of Lübeck, a city in the state of Schleswig-Holstein, Germany, with a population of approximately 220,000 and a car ownership rate of 464 per 1,000 inhabitants. We take into account the growing popularity of private and electric car-sharing, as well as the mobility needs of the residents within their living spaces [12].

Developing MAS in this domain requires various agent types and groups to reflect the different actors involved, including electric cars, their users, and a micro-grid energy system. In our setting, we consider electric cars and charging stations with bi-directional capabilities, meaning that the cars, when not driving, can also serve as energy buffers and feed back energy into the grid (V2G). The car users encompass individuals from diverse age groups, with a variety of car models and sizes to cater to their needs. The micro-grid energy system harnesses renewable energy from different sources, such as wind and solar power, with the latter generated by photovoltaic (PV) panels installed on residential rooftops.

Our model and simulation will be developed by domain experts rather than versatile programmers. As a result, we seek to find a MAS development platform that provides a modeling language based on or similar to popular programming languages like Python or Java. Additionally, we will incorporate historical records of the grid, weather conditions, and car-sharing bookings into the model. This means that the platform should also support the import of data from external sources, such as CSV files. Other requirements regarding MAS development platforms will be identified in Sect. 3.

2.1 Model Description

To scale our model, we utilized the latest available data from the city's statistics department [12]. Based on this data, we identified a total of 150,000 car users, all of whom we assumed to be car-sharing users. In addition, we considered the number of cars in the city, which was recorded as 103,000. For the purpose of our simulation, we assumed that all of these cars were electric, used for car-sharing. Furthermore, we took into account the 50,000 residential buildings in Lübeck and assumed that they are either partially or fully equipped with PV panels, heating systems, and air conditioning units. These buildings, along with the

cars and their users, will collectively represent the agents in our simulation. To capture the dynamics and patterns over time, we will simulate a complete year of data records. By incorporating these real-world numbers and characteristics, our simulation aims to accurately represent the scale and size of the agents and their interactions within the modeled system.

Dynamical generation and optimization are necessary for several components of the agents' modules. These include a booking reservation logic (booking algorithm) for users to reserve cars and bi-directional charging of electric car batteries. Optimizing the booking algorithm is crucial to avoid high car unavailability to satisfy users' needs or to serve as an energy buffer. We have already developed a booking reservation algorithm using the Python programming language, and by leveraging linear programming techniques [30], we have successfully optimized both the booking algorithm and the bi-directional charging system. In addition to this, we also plan to use reinforcement learning to train agents on adaptive booking strategies without harming the energy grid.

2.2 Platform Evaluation Scenario

In order to assess the MAS development platforms, we develop a minimum viable product (MVP) that simulates a use case closely resembling our intended final product. For this purpose, we establish the following requirement in the use case:

1. **Users:** Agents who book electric cars for random short-distance trips, following a normal distribution throughout the day.
2. **Cars:** These are electric cars utilized for a car-sharing service, which have a certain driving range depending on their state of charge (SoC). They recharge upon return from a trip and are capable of buffering excess energy when connected to the grid.
3. **Houses:** Residential units with optional features like solar power roofs, heat pumps, air conditioning systems. Private cars contribute to the energy consumption of the houses when they are being recharged.
4. **Random Weather:** The use case involves random sunshine and temperature data at a 5-min resolution, incorporating yearly and daily cycles, seasonal variations, and daytime fluctuations. These patterns include noise to capture realistic yearly and daily variations. By incorporating this data, we have abstracted the energy model (micro-grid) aspect that will be included in the final product. However, it is important to note that in our final product, we intend to conduct a comprehensive study of a micro-grid connection within the given settings.

Figure 1 provides an overview of the simulation use case. The simulation does not consider the positions of houses and users, types of cars, users' age groups, and other details that will be included in our final model. We also do not consider modeling charging stations as we would have in the final model.

Use Case Agent Actions and Interactions: User agents book cars at random times, with more daytime activity and a normal distribution peaking at 14:00

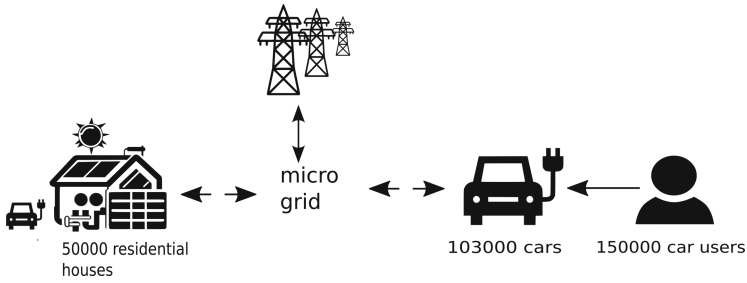


Fig. 1. Use case overview. Arrows indicate temporary or permanent connectivity or availability.

and a 3-h standard deviation. The cars are recharged after each trip, with trip duration determining the required charging. Power consumption and generation data are collected in 5-min intervals, using realistic charging times and battery sizes. Charging times contribute to overall power consumption data.

The house agents generate and consume electricity based on generated weather data. House agents have randomized features, such as air conditioning, solar power, and heat pumps with varying capacities. Larger houses have greater solar power capacity. Power consumption and generation data are collected for each 5-min slot, with car charging times included in the overall data. Electric car batteries buffer excess solar energy, feeding it back to the house when needed, if the car is idle.

In this paper, our primary objective is to evaluate different platforms and identify the most suitable one for our project. Consequently, the simulation use case described herein only considers certain details (of the intended final product) to assess the platforms.

2.3 Expected Features from MAS Platform

As our project is expected to span over multiple years, it is essential for us to have an active and well-maintained platform that can effectively support our evolving needs. The complexity of the project will undoubtedly grow over time, requiring a simulation platform that can keep up with these changes. In this paper and the MASIRI project at large, we identify some general as well as domain-specific features that are crucial. The general features include:

- G1 Language familiarity:** This is important because of the expertise of the developers. The platform must be one that does not require much time to learn its syntax and ABM implementation.
- G2 Scalability:** Currently, our plan is to simulate the inhabitants of Lübeck. However, it is possible that the population size may change, leading to a larger population, or that the model might be adapted or modified to analyze another city. Therefore, it is crucial to have a platform that can accommodate scalability when necessary.

- G3 Parallelism & distributed computing:** Based on our preliminary investigation, we have found that most platforms offer some level of parallel computing features. However, it is important to note that the emphasis on ease of use rather than performance is evident in some platforms [28]. Given that our model comprises various types of agents and is of large size, efficient simulation running time becomes a crucial factor, and therefore, performance is a significant consideration.
- G4 Community support:** A platform with an active and strong community fosters knowledge sharing and facilitates the exchange of experiences. This community support will be invaluable during the development phase, particularly when it comes to debugging and troubleshooting. The collective expertise and insights of community members can provide valuable guidance and solutions, enabling us to address challenges more efficiently.
- G5 Interoperability:** The platform should facilitate interface with external libraries, tools, and data sources for domain-specific functionality and streamlined data processing. This feature will enable us to incorporate our already developed booking algorithm, optimization tools, and the historical record.
- G6 Visualization & analysis tools:** The inclusion of built-in tools dedicated to visualizing and analyzing simulation outputs plays a crucial role in facilitating a deeper understanding of the simulation results and enabling comprehensive performance evaluation. These integrated tools will provide us with intuitive and interactive interfaces to explore, interpret, and visualize complex simulation data in a meaningful way.
- G7 Documentation:** A wealth of comprehensive resources, including platform documentation, YouTube videos, tutorials, and working examples, will greatly assist us in gaining a thorough understanding of how to effectively utilize the platform. These resources will serve as invaluable tools during the initial stages, providing step-by-step guidance, practical demonstrations, and real-world examples that will aid in our learning process and enable us to make the most of the platform's capabilities.

For our agent-based learning, mobility, and energy use case, the following additional features are crucial:

- D1 Learning capabilities:** For an ideal platform, it is imperative to encompass support for a diverse range of learning algorithms that enable the training of agents to adapt strategies and optimize energy usage.
- D2 OpenStreetMap space:** The ability to incorporate geo-spatial data, including the positions of users and cars in relation to each other, house locations, road networks, points of interest, and more, is crucial for accurately simulating real-world mobility and energy systems. It should provide a rich and detailed spatial dataset that enhances the realism and accuracy of our model.
- D3 Data integration:** The integration of external data plays a pivotal role in the effectiveness and significance of an ABM platform. By seamlessly incorporating real-time or historical data sources, the platform becomes

capable of facilitating realistic and dynamic simulations that closely mirror the complexities of real-world situations.

3 Multi-level Selection

To compile a comprehensive list of platforms, we conducted thorough searches across multiple sources, including diverse projects, comparative studies, and online resources. In particular, notable survey articles such as [15] and review articles like [16, 23, 26], and [1] provided valuable insights and compilations of platforms in the field. Leveraging these sources, we identified platforms based on their specific areas of application, existing projects utilizing the platforms, and studies that conducted comparisons among different platforms. Given the diverse modeling approaches offered by these platforms, we categorized them into the following groups:

- **Language or Environment for MAS (LEM):** Refers to programming languages, frameworks, and software environments that are used to create, simulate, and deploy ABMs.
- **Support Software (SS):** Refers to a software tool, package, or platform that provides specific functionalities and capabilities to facilitate the development, deployment, and management of ABMs.
- **MAS-based Modeling Platform (MMP):** Refers to a software application or platform that specifically focuses on modeling and simulating ABMs. These platforms provide an environment where developers can design and simulate agents, their behaviors, interactions, and the dynamic environment in which they operate.

We then gathered information (summarized in Table 1) on each platform’s modeling language, licence, and activity status². The latter was checked through various means, including visiting the platform’s website (in search of recent updates, news, and announcements), engaging with the community (such as discussion groups), and examining the GitHub or source code repository.

3.1 First Round of Selection

After careful evaluation, we eliminated platforms for which we could not find essential information, such as licensing details or recent activities. Some platforms, including FAME, SWARM, JACK, Junus, GOAL, Cougaar, and StarLogo (TNG and Nova versions), have not been regularly updated or maintained. This lack of maintenance raises concerns about their reliability, potential bugs, and compatibility with modern operating systems. To mitigate the risk of selecting an inactive platform (or platform that might become inactive) for our project, we decided to also exclude platforms with no activity for more than 2 years. Consequently, FLAME, 2APL, ZEUS, and ActressMAS were also removed from consideration.

² This information was checked on May 25th, 2023.

Table 1. A table showing different MAS development platforms. The platforms are listed in alphabetical order and not ranked. GPL stands for General Public Licence, AFL for Academic Free License, EPL for Eclipse Public License, and COSL for Cougaar Open Source License.

Name	Modeling language	Licence	Category	Last activity
ActressMAS	C#	Open source	LEM	2021.06.15
Agents.jl	Julia	MIT	MMP	2023.05.19
AgentScript	JavaScript	Various	LEM	2023.01.23
Cougaar	Java	COSL	LEM	2013.10.22
FAME	Jave	Apache v2.0	SS	2021.01.20
FLAME	C/C++	Open source	LEM	2017.05.30
GAMA	GAML	GNU GPL v3	LEM	2021.10.15
GOAL	GOAL	unknown	LEM	2021.11.08
JACK	Java	Commercial license	LEM	2015.07.01
JADE	Java	Open source (Java)	LEM	2022.12.19
Jadex	Java	GNU GPL v2.0	LEM	2022.10.08
Janus	SARL	Apache v2.0	LEM	unknown
Jason	AgentSpeak	GNU GPL v3	LEM	2023.04.02
Mason	Java	AFL	LEM	2022.09.07
MATSim	XML	GNU GPL	MMP	2023.04.01
Mesa	Python 3+	Apache v2.0	MMP	2023.03.08
NetLogo	NetLogo	GNU GPL v2.0	MMP	2023.05.11
Repast4Py	Python	Various	MMP	2023.03.02
SPADE	Python	Open source	SS	2023.12.13
SpaDES	R	GNU GPL v3	SS	2022.02.16
SUMO	Python, Java, C++	EPL v2.0	SS	2023.06.29
StarLogo	Objective C	Various	LEM	2018.11.24
SWARM	Java C	GNU GPL	LEM	2013.08.01
ZEUS	Java	Unknown	LEM	2021.06.20
2APL	2APL	GNU GPL v3.0	LEM	2021.12.01

Platforms such as AgentScript, SPADE, Jason, Jadex, and JADE are actively maintained and continue to receive updates. Unfortunately, we were unable to find information regarding the compatibility of these platforms with the general requirements G4 and G7. SpaDES is based on the R language, while Jason is based on AgentSpeak. Moreover, we found limited documentation and community support forums for these platforms. Consequently, their capacity to fulfill requirements G1, G4, and G7 is further hindered.

We examined different implementations of Repast, specifically focusing on Repast Symphony, a Java-based modeling toolkit, and Repast4Py, a Python-based distributed agent-based modeling toolkit. Since our booking algorithm

was already developed in Python, we found the Repast4Py version more appealing. We further evaluated it alongside another Python-based platform, Mesa. Through our analysis, we discovered that Repast offers certain advantages over Mesa in terms of documentation and flexibility in programming languages [5]. However, Mesa provides advantages in terms of simplicity, user-friendliness, and seamless integration with Python libraries and frameworks [20]. The simulation software SUMO [17] was also evaluated as part of our study. It offers a user-friendly graphical user interface (GUI) called “sumo-gui,” which simplifies the process of adding road layouts, intersections, vehicles, and users through drag and drop functionality. Additionally, SUMO provides an interface Python library called “TraCI,” allowing users to develop Python scripts that can connect to a running SUMO simulation, retrieve information, and control various aspects of the simulation. We encountered challenges when attempting to incorporate the energy model component of our model in a seamless and straightforward manner.

Although some of the remaining platforms are not based on Java or Python, they have extensive documentation and a wealth of working examples available, making them popular within the ABM modeling community. Moving forward, we will provide a brief overview of these platforms.

Mason [19] is a fast, discrete-event, multi-agent simulation library core in Java. It serves as a robust foundation for developing large-scale, custom-purpose simulations in Java, while also catering to the requirements of lightweight simulation applications. It has a comprehensive model library accompanied by an optional suite of visualization tools, catering to both 2D and 3D simulations.

Mesa [20] is a versatile and open-source Python library specifically designed for agent-based modeling (ABM). It offers users a streamlined approach to developing agent-based models by providing built-in core components like spatial grids and agent schedulers. Additionally, Mesa allows for flexible customization through the implementation of personalized components.

NetLogo [28] is an integrated development environment and programming language designed for modeling and simulating complex systems. It features a custom scripting language, NetLogo, and built-in visualization tools. Users can export data to external visualization tools for advanced analysis.

GAMA [2] is an open-source simulation platform and modeling language offering various features for agent creation, communication, and decision-making. GAMA provides visualization and analysis tools and a custom language, GAML, for composing complex models with spatial dimensions. The platform supports running simulations on multiple machines for increased performance.

MATSim [32] is an open-source framework for simulating large-scale transportation systems, modeling individual travelers and vehicles within a network. It is Java-based and features tools for different transport modes, routing algorithms, and activity-based travel demand modeling. MATSim also offers visualization and analysis tools for exploring simulation dynamics and is ideal for predicting policy impacts on transportation systems.

Agents.jl [7] is a Julia library for agent-based modeling within the JuliaDynamics ecosystem. Julia is a high-performance language suitable for computational and numerical science applications. Agents.jl manages and creates spaces, simplifies data collection, and offers visualization options, including OpenStreetMaps and 3D visualizations, through related JuliaDynamics libraries. Julia also enables developers to call methods, functions, or scripts from other languages such as Python or R.

A summary of the pros and cons of the platforms following the initial round of selection is presented in Table 2. This summary highlights the notable advantages and drawbacks of each platform, providing valuable insights to inform the subsequent stages of the selection process.

3.2 Second Round of Selection

In order to evaluate the remaining platforms based on the specified requirements outlined in Subject 2.3, and taking into account the pros and cons summarized in Table 2, we employ a rating scale. This rating scale assigns values of *high* = 3, *medium* = 2, and *low* = 1 to each platform, indicating the level of satisfaction for each requirement.

For each platform, we assess its performance against each requirement and assign a corresponding rating. A rating of “*high*” is assigned when a platform fully satisfies a requirement, “*medium*” when it partially satisfies the requirement, and “*low*” when it does not meet the requirement. By applying this rating scale, we calculate a score for each platform, considering the cumulative ratings for all the evaluated requirements as follows:

$$score = 3 \times (highs) + 2 \times (mediums) + 1 \times (lows). \quad (1)$$

The platform with the highest cumulative score signifies that it fulfills the majority of the requirements. The scores for each platform are calculated based on the assigned ratings, and Table 5 showcases the platforms alongside their corresponding scores.

In the end, we observed that Mason (scored 18) and GAMA (scored 18), have relatively smaller user communities compared to other platforms, resulting in limited availability of tutorials and documentation. We encountered challenges in finding comprehensive resources such as kickstart examples, troubleshooting guides, and interactive forums for engaging with developers and users of these platforms. MATSim (scored 19) and NetLogo (scored 18), although they have larger user communities and tutorials, lack community support at a similar level as Mason and GAMA. Additionally, we found limited examples or resources showcasing the implementation of non-transport simulations using MATSim. NetLogo, being a Logo-based language, requires developers to familiarize themselves with its specific syntax. Furthermore, information on integrating external modules or expanding the platform’s functionality was scarce during our evaluation process.

Table 2. Pros and Cons.

Platform	Pros	Cons
Mason	Highly customizable and flexible for creating multi-agent simulations. Supports both discrete and continuous modeling	Steep learning curve. Lacks a user-friendly interface. Limited visualization options
Mesa	Easy to use and well-documented. Supports both discrete and continuous modeling. Has a built-in visualization tool	Limited support for advanced features like parallel computing and large-scale simulations
NetLogo	User-friendly and intuitive interface. Supports both discrete and continuous modeling. Has a large library of pre-built models. Good visualization options	Limited support for large-scale simulations. Limited customization options
GAMA	Highly customizable and flexible. Supports both discrete and continuous modeling. Good visualization options	Steep learning curve. Limited community support
MATSim	Strong support for agent-based transportation modeling. Supports large-scale simulations. Good visualization options	Limited support for other types of multi-agent simulations. Steep learning curve
Agents.jl	Highly customizable and flexible. Supports both discrete and continuous modeling. Good support for scientific computing	Recurring issue of packages being redefined constantly. Steep learning curve. Various visualization options

Mesa (scored 23), benefiting from Python’s intuitiveness, community support, and familiarity, was more appealing, especially since we already have a module (booking algorithm) that was developed in Python. In addition to this, we found it to be more easy to learn compared to RepastPy. Similarly, Agents.jl (scored 26) is based on Julia, a language that has some similarities in syntax with Python, and the ability to call Python’s methods and scripts through packages such as PyCall.jl. Agents.jl is appealing as well due to multiple reasons. Among these reasons is the fact that it benefits from Julia’s active user base and the availability of Julia-specific libraries and resources that are interoperable with Python, agents’ learning capabilities, speed, scalability [7] etc. Agents.jl and Mesa have the highest and second highest score. As such, we pre-selected Agents.jl and Mesa as the final two platforms for further evaluation.

4 Mesa vs. Agents.jl

We created a minimum viable product (MVP) reflecting our final product to compare the performance and ease of use of Mesa and Agents.jl. This MVP is based on the scenario in Subsect. 2.2. The code is publicly available³. During

³ https://github.com/stockh0lm/masiri_mas_framework_benchmark.

development, we sought help from Python and Julia communities for technical details and bug hunting, using chat platforms and web forums. While we found no dedicated Mesa community, there was an Agents.jl discussion channel in the Julia Slack server⁴. Both communities were approachable, helpful, and competent, with fast response times. Although the Julia community was smaller, it did not negatively impact the support received.

It is worth noting that Mesa is single-threaded; therefore, there is no direct way to utilize multi-threaded runs. The only option for achieving concurrent or parallel computing in Mesa is to run multiple computations in parallel and use the intermediate results. This implies that the implementation discussed in this paper utilizes a single thread for Mesa.

4.1 Implementation - Model Speed and Scalability Evaluation

We implemented identical simulations in Mesa and Agents.jl. Mesa, by default, operates in a single-threaded manner. However, in order to enhance its performance and enable parallel agent processing, we incorporated multi-threading functionality in Julia, utilizing varying numbers of concurrent threads. Anticipating more complex final agents, we added a recursive Fibonacci computation for both frameworks to increase computational load. We ran both simulations with agent numbers as powers of two (1, 2, 4, ..., 2048), at which point the Mesa runs took several days, and clear trends were established. We examined Agents.jl's vertical scalability using 1, 2, 4, and 8 threads for different agent numbers. We did not attempt using the Distributed.jl library for Julia, as it would have exceeded our time scope and been challenging to test at the time. The results of this implementation is shown in Tables 3 and 4.

In order to evaluate the RAM consumption and runtime performance (excluding the compile-time of Agents.jl), we incorporated code to measure these metrics and enabled dynamic configuration of the number of agents. The benchmarks were conducted on a Debian GNU-Linux Server with specifications including 12 cores, 32GB RAM, Julia 1.8.5, Agents.jl 5.12, Python 3.11.1, and Mesa 1.2.1. To minimize system-related variability, we recorded the median runtime of four simulation runs, ensuring that the agent numbers varied identically for each run.

Table 3. Comparison of RAM usage in the benchmark for Mesa and Agents.jl. The runs for 10^6 and 10^7 agents were never completed and only started to measure the trend in memory usage.

	Mesa	Agents.jl
1 agent	155.84 MB	555.39 MB
1000000 agents	0.7 GB	0.7 GB
10000000 agents	5.3 GB	3.3 GB

⁴ <https://julialang.slack.com/archives/CBLNLEU74>.

Table 4. Single threaded speed comparison of Mesa and Agents.jl.

	Mesa	Agents.jl
1 agent	91 s	0,5 s
32 agents	48 min	2,5 s
2048 agents	52 h	155 s

4.2 Benchmark Results

Mesa required about 160 MB for agent counts up to 2048, while Agents.jl used between 524 MB and 608 MB in this range, see Fig. 2. Mesa’s RAM footprint was about four times smaller and more consistent than Agents.jl’s. In both cases, the majority of the memory footprint appeared to be caused more by binaries and libraries than the model and agent data. This manifested only at higher agent counts of one million, as seen in Table 3. Run time for the models increased with the number of agents: Mesa ran between 50 s (1 agent) and 52 h (2048 agents). In Agents.jl the same computations took between 0.2 s and 5 min, as displayed in Table 4. Mesa is about a thousand times slower than Agents.jl when running the simulations.

The multi-threading results (also depicted in Fig. 3) showed that run-time decreased as thread count increased, particularly for higher agent numbers, where eight threads were roughly twice as fast as one thread. However, the scalability was not consistently linear, indicating factors such as resource contention, parallelization overhead, or implementation limitations. For low agent numbers, higher thread counts led to increased run-time, suggesting that the optimal thread count is dependent on the simulation’s complexity and agent count. Overall, Agents.jl demonstrates promising vertical scalability, but further optimization is necessary for efficient resource utilization. Based on our evaluation and experience during the simulation of our use case, we have concluded that Agents.jl is the most suitable platform for implementing our model. One

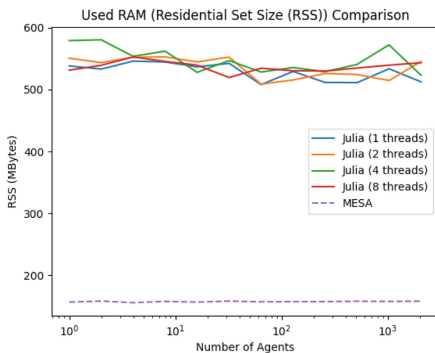
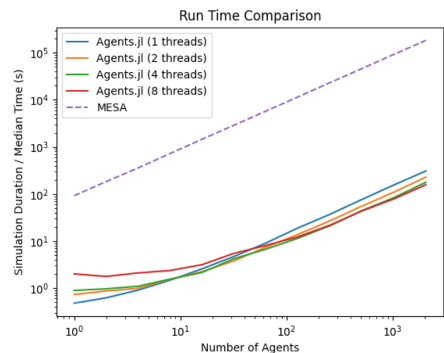
**Fig. 2.** Used RAM Comparison.**Fig. 3.** Median Run Time Comparison.

Table 5. Scoring MAS development platforms.

Tools	Language Familiarity	Scalability	Parallelism & distributed computing	Community support	Interoperability	Visualization & analysis tools	Documentation	Learning capabilities	OpenStreet-Map space	Data integration	Score
Mesa	high	medium	low	medium	high	high	high	medium	low	high	23
Agents.jl	low	High	High	High	High	High	High	medium	high	medium	26
Mason	high	high	medium	low	medium	medium	low	low	low	medium	18
MATSim	high	high	medium	low	medium	medium	medium	low	low	medium	19
NetLogo	low	medium	medium	medium	medium	high	medium	medium	low	low	18
GAMA	low	high	medium	low	medium	medium	low	medium	low	high	18

of the key factors that led us to this conclusion is the impressive runtime speed we observed while using Agents.jl. It has demonstrated superior performance in handling the computational demands of our simulation, making it the preferred choice among the evaluated platforms.

5 Conclusion

In this paper, we conducted a study on various MAS development platforms with the aim of selecting the most suitable candidate for modeling a Multi-Agent Simulation of Intelligent Resource Regulation in the context of Integrated Energy and Mobility. We considered both general (G1-G7) and domain-specific (D1-D3) features during our evaluation process. Through this study, we identified two final candidates, namely Mesa in Python and Agents.jl in Julia. To assess their suitability for our project, we developed a MVP in these two platforms and evaluated their performance, speed, scalability, and memory footprint. Our evaluation criteria also encompassed determining whether the platforms fulfill our requirements for both general and domain-specific features.

While Mesa had a smaller memory footprint and a larger community, Agents.jl offered significantly better runtime performance, which was nearly a thousand times faster in some cases. Considering these factors, we concluded that Agents.jl in Julia was the most suitable framework for our project. Its superior performance, ability to scale with larger models, and integration with Python code make it a solid choice for implementing our intelligent resource regulation model.

References

1. Abar, S., Theodoropoulos, G.K., Lemarinier, P., O'Hare, G.M.: Agent based modelling and simulation tools: a review of the state-of-art software. *Comput. Sci. Rev.* **24**, 13–33 (2017)
2. Amouroux, E., Chu, T.-Q., Boucher, A., Drogoul, A.: GAMA: an environment for implementing and running spatially explicit multi-agent simulations. In: Ghose, A., Governatori, G., Sadananda, R. (eds.) *PRIMA 2007. LNCS (LNAI)*, vol. 5044, pp. 359–371. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01639-4_32
3. Bitting, E., Carter, J., Ghorbani, A.A.: Multiagent systems development kits: an evaluation. In: *Proceedings of the 1st Annual Conference on Communication Networks & Services Research*. Moncton, Canada, pp. 80–92. Citeseer (2003)
4. Bousquet, F., Le Page, C.: Multi-agent simulations and ecosystem management: a review. *Ecol. Model.* **176**(3–4), 313–332 (2004)
5. Collier, N.T., Ozik, J., Tatara, E.R.: Experiences in developing a distributed agent-based modeling toolkit with Python. In: *2020 IEEE/ACM 9th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*, pp. 1–12. IEEE (2020)
6. Crooks, A.T., Heppenstall, A.J.: Introduction to agent-based modelling. In: Heppenstall, A., Crooks, A., See, L., Batty, M. (eds.) *Agent-Based Models of Geographical Systems*, pp. 85–105. Springer, Dordrecht (2012). https://doi.org/10.1007/978-90-481-8927-4_5

7. Datseris, G., Vahdati, A.R., DuBois, T.C.: Agents.jl: a performant and feature-full agent-based modeling software of minimal code complexity. *Simulation*, p. 00375497211068820 (2022)
8. ElBanhawy, E.Y., Dalton, R., Thompson, E.M., Kottor, R.: Real-time electric mobility simulation in metropolitan areas. A case study: Newcastle-Gateshead, in **1**, 533–546 (2012)
9. Garcia, E., Giret, A., Botti, V.: On the evaluation of MAS development tools. In: Bramer, M. (ed.) *IFIP AI 2008. ITIFIP*, vol. 276, pp. 35–44. Springer, Boston (2008). https://doi.org/10.1007/978-0-387-09695-7_4
10. Garcia, E., Giret, A., Botti, V.: Analysis, comparison and selection of mas software engineering processes and tools. In: Yang, J.-J., Yokoo, M., Ito, T., Jin, Z., Scerri, P. (eds.) *PRIMA 2009. LNCS (LNAI)*, vol. 5925, pp. 361–375. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-11161-7_25
11. Grignard, A., et al.: The impact of new mobility modes on a city: a generic approach using ABM. In: Morales, A.J., Gershenson, C., Braha, D., Minai, A.A., Bar-Yam, Y. (eds.) *ICCS 2018. SPC*, pp. 272–280. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96661-8_29
12. Lübeck, H., Bürgermeister, D., et al.: *Statistisches Jahrbuch 2019–2022: Lübeck in Zahlen 2019–2022*. Hansestadt Lübeck, Fackenburger Allee 29, 23539 Lübeck (2022). <https://bekanntmachungen.luebeck.de/dokumente/d/1720/inline>
13. Hertzke, P., Müller, N., Schenk, S., Wu, T.: The global electric-vehicle market is amped up and on the rise. McKinsey Center for Future Mobility, pp. 1–8 (2018)
14. Jittrapirom, P., Caiati, V., Feneri, A.M., Ebrahimigharehbaghi, S., Alonso González, M.J., Narayan, J.: Mobility as a service: a critical review of definitions, assessments of schemes, and key challenges (2017)
15. Kravari, K., Bassiliades, N.: A survey of agent platforms. *J. Artif. Soc. Soc. Simul.* **18**(1), 11 (2015)
16. Leon, F., Paprzycki, M., Ganzha, M.: A review of agent platforms. *Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS), ICT COST Action IC1404*, pp. 1–15 (2015)
17. Lopez, P.A., et al.: Microscopic traffic simulation using sumo. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pp. 2575–2582. IEEE (2018)
18. López, T.S., Brintrup, A., McFarlane, D., Dwyer, D.: Selecting a multi-agent system development tool for industrial applications: a case study of self-serving aircraft assets. In: 4th IEEE International Conference on Digital Ecosystems and Technologies, pp. 400–405. IEEE (2010)
19. Luke, S.: Multiagent simulation and the mason library. George Mason University 1 (2011)
20. Masad, D., Kazil, J.: MESA: an agent-based modeling framework. In: 14th PYTHON in Science Conference, vol. 2015, pp. 53–60. Citeseer (2015)
21. Nijenhuis, B., Doumen, S.C., Hönen, J., Hoogsteen, G.: Using mobility data and agent-based models to generate future e-mobility charging demand patterns (2022)
22. Owen, C., Love, D., Albores, P.: Selection of simulation tools for improving supply chain performance. In: *Proceedings of 2008 OR Society Simulation Workshop* (2008)
23. Pal, C.V., Leon, F., Paprzycki, M., Ganzha, M.: A review of platforms for the development of agent systems. arXiv preprint [arXiv:2007.08961](https://arxiv.org/abs/2007.08961) (2020)
24. Querini, F., Benetto, E.: Agent-based modelling for assessing hybrid and electric cars deployment policies in Luxembourg and Lorraine. *Transp. Res. Part A: Policy Pract.* **70**, 149–161 (2014)

25. Railsback, S.F., Lytinen, S.L., Jackson, S.K.: Agent-based simulation platforms: review and development recommendations. *Simulation* **82**(9), 609–623 (2006)
26. Rendón Sallard, T., Sánchez-Marrè, M.: A review on multi-agent platforms and environmental decision support systems simulation tools (2006)
27. Thoma, D., Sachenbacher, M., Leucker, M., Ali, A.T.: A digital twin for coupling mobility and energy optimization: the ReNuBiL living lab. In: FM2023 Workshop on Applications of Formal Methods and Digital Twins (2023, to appear)
28. Tisue, S., Wilensky, U.: NetLogo: a simple environment for modeling complexity. In: International Conference on Complex Systems, vol. 21, pp. 16–21. Citeseer (2004)
29. Tran, Q.-N.N., Low, G., Williams, M.-A.: A feature analysis framework for evaluating multi-agent system development methodologies. In: Zhong, N., Raś, Z.W., Tsumoto, S., Suzuki, E. (eds.) ISMIS 2003. LNCS (LNAI), vol. 2871, pp. 613–617. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39592-8_87
30. Vanderbei, R.J., et al.: Linear Programming. Springer, Cham (2020)
31. Vuthi, P., Peters, I., Sudeikat, J.: Agent-based modeling (ABM) for urban neighborhood energy systems: literature review and proposal for an all integrative ABM approach. *Energy Inform.* **5**(4), 1–23 (2022)
32. Axhausen, K.W., Horni, A., Nagel, K.: The Multi-agent Transport Simulation MATSim. Ubiquity Press (2016)