




Mango.jl: A Julia-Based Multi-Agent Simulation Framework

Jens Sager ^{1,2*} and Rico Schrage ^{1,2*}

¹ Digitalized Energy Systems Group, Carl von Ossietzky Universität Oldenburg, 26129 Oldenburg, Germany ² Energy Division, OFFIS Institute for Information Technology, 26121 Oldenburg, Germany * These authors contributed equally.

DOI: [10.21105/joss.07098](https://doi.org/10.21105/joss.07098)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Mehmet Hakan Satman](#) 

Reviewers:

- [@aurorarossi](#)
- [@Tortar](#)

Submitted: 07 August 2024

Published: 01 October 2024

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

Multi-agent simulations are inherently complex, making them difficult to implement, maintain, and optimize. An agent, as defined by ([Russell & Norvig, 2010](#)), is software that perceives its environment through sensors and acts upon it using actuators. Mango.jl is a simulation framework for multi-agent systems implemented in Julia ([Bezanson et al., 2017](#)). It enables quick implementations of multiple communicating agents, either spanning multiple devices or in a single local environment.

For the design process, Mango.jl provides a general structure and a role concept to help develop modular and loosely coupled agents. This is aided by the built-in task scheduler, with convenience methods to easily schedule timed and repeated tasks that are executed asynchronously.

Agents communicate with each other via message exchange. Each agent is associated with a container that handles network operations for one or more agents. Messages may be sent directly via TCP connections or indirectly using an MQTT broker. This way, Mango.jl makes it easy to set up multi-agent simulations on multiple hardware devices.

Mango agents can run in real-time or using simulated time, with either discrete event or stepped time versions. This is useful for simulations, where simulated time should run much faster than real-time. These non-real-time simulation modes also enable the user to simulate the communication to validate the robustness of multi-agent systems against communication issues.

Statement of need

Applications of multi-agent systems can be found in various fields, such as in distributed optimization ([Yang et al., 2019](#)), reinforcement learning ([Gronauer & Diepold, 2022](#)), robotics ([Chen et al., 2019](#)) and more. Many of these systems are highly complex and feature heterogeneous and interacting actors. This makes them inherently difficult to model and develop. Therefore, a structured development framework to support this process is a valuable asset.

While Mango.jl is a general purpose multi-agent framework, we will focus on energy systems in the following as this is the domain the authors are most familiar with.

Many of the ideas for Mango.jl are based on the existing Python framework mango ([Schrage et al., 2024](#)). The main reason for this Julia-based version is to allow better focus on simulation performance, enabling larger scales of multi-agent simulations. This is especially relevant in the energy domain, where an increasing amount of energy resources (e.g. batteries and PV-generators) have distributed ownership, competing goals and contribute to the same power

grid. Large scale multi-agent simulations allow researchers to study the behavior of these participants in energy markets and grid simulations.

The Python version of mango has already been successfully applied to various research areas in the energy domain, including coalition formation in multi-energy networks (Schrage & Nieße, 2023), distributed market participation of battery storage units (Tiemann et al., 2022), distributed black start (Stark et al., 2021), and investigating the impact of communication topologies on distributed optimization heuristics (Holly & Nieße, 2021). New Julia-based projects using `Mango.jl` are in active development.

Related Frameworks

To our knowledge, there is no Julia-based multi-agent framework with a focus on agent communication and distributed operation like `Mango.jl`.

`Agents.jl` (Datseris et al., 2022) is a multi-agent framework for modeling agent interactions in a defined space to observe emergent properties like in animal flocking behavior or the spreading of diseases. This puts it in line with frameworks like `mesa` (Kazil et al., 2020) or `NetLogo` (Tisue & Wilensky, 2004). These have a different scope than `Mango.jl` which is more focused on agent communication and internal agent logic for software applications.

`JADE` (Bellifemine et al., 2001) and `JIAC` (Lützenberger et al., 2013) are Java frameworks of similar scope but are not actively developed anymore. `JACK` (Winikoff, 2005) provides a language and tools to implement communicating agents but is discontinued and proprietary. The `agentframework` (Zhang, 2022) is based on JavaScript and has less focus on communication than `Mango.jl`. Lastly, the original Python version of mango (Schrage et al., 2024) is of course most similar in scope, but makes it more difficult to write high performance simulations, due to the use of `asyncio` and the lack of native multi-threading in Python.

Performance

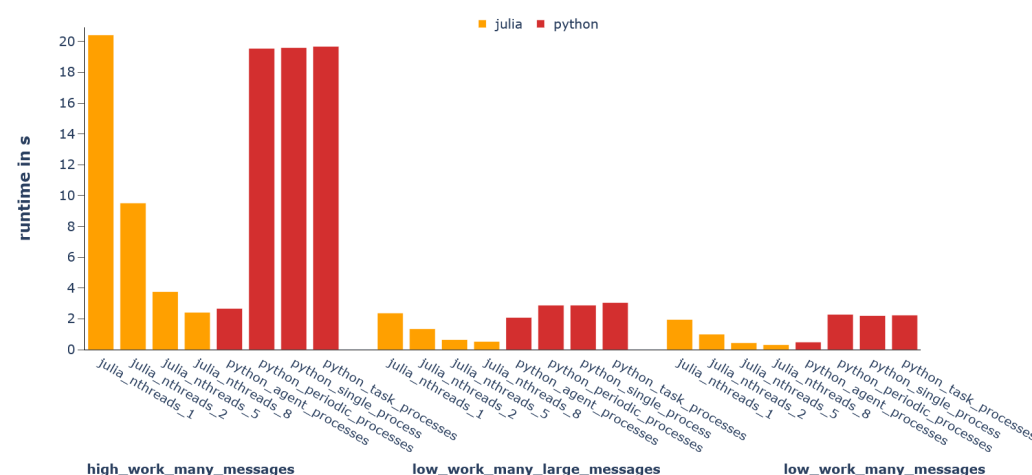


Figure 1: Performance comparison of Python and Julia mango.

The performance of the Python and Julia versions of mango were benchmarked against each other. The results are shown in Figure 1 and the relevant code is available at [mango_benchmark](https://github.com/JuliaMango/mango_benchmark).

The aim of these scenarios is to measure the performance of the frameworks' core features. This mainly means it measures how efficiently tasks are scheduled and messages are sent and

handled. To achieve this, benchmark scenarios have agents set up in a small world topology communicating a fixed number of messages between each other while performing simulated workloads. All workloads in the agents are entirely simulated by static delays. Thus, the benchmarks assume that workloads in Python and Julia are identical.

The main advantage of `Mango.jl` is in the ease of parallelization. Python can in some cases reach similar performance using subprocesses for parallel execution to circumvent the limitations of the Python global interpreter lock. Compared to native threads in Julia, however, this is more prone to issues with the operating system, because it requires large amounts of file handles to operate the subprocesses. Overall, it is easier to get high performance from `Mango.jl`.

Basic Example

In this example, we define two agents in two containers (i.e. at different addresses) that pass messages to each other directly via TCP. Containers can be set up and equipped with the necessary TCP protocol.

using Mango

```
# Create the container instances with TCP protocol
container = create_tcp_container("127.0.0.1", 5555)
container2 = create_tcp_container("127.0.0.1", 5556)
```

Now, we need to define the agents. An agent in `Mango.jl` is a struct defined with the `@agent` macro. We define a `TCPPingPongAgent` that has an internal counter for incoming messages.

```
@agent struct TCPPingPongAgent
    counter::Int
end
```

After creation, agents have to be registered to their respective container.

```
# Create instances of ping pong agents
ping_agent = TCPPingPongAgent(0)
pong_agent = TCPPingPongAgent(0)

# register each agent to a container and give them a name
register(container, ping_agent, "Agent_1")
register(container2, pong_agent, "Agent_2")
```

When an incoming message is addressed at an agent, its container will call the `handle_message` function for it. Using Julia's multiple dispatch, we can define a new `handle_message` method for our agent.

```
# Override the default handle_message function for ping pong agents
function Mango.handle_message(agent::TCPPingPongAgent, message::Any, meta::Any)
    agent.counter += 1

    println(
        "$(agent.aid) got a message: $message." *
        "This is message number: $(agent.counter) for me!"
    )

    # doing very important work
    sleep(0.5)

    if message == "Ping"
        reply_to(agent, "Pong", meta)
    end
end
```

```
elseif message == "Pong"
    reply_to(agent, "Ping", meta)
end
end
```

With all this in place, we can send a message to the first agent to start the repeated message exchange. To do this, we need to start the containers so that they listen for incoming messages and send the initiating message. The best way to start the container message loops and ensure they are correctly shut down in the end is the `activate(containers)` function.

```
activate([container, container2]) do
    send_message(ping_agent, "Ping", address(pong_agent))

    # wait for 5 messages to have been sent
    while ping_agent.counter < 5
        sleep(1)
    end
end
```

Acknowledgements

This work has been partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 359941476.

References

- Bellifemine, F., Poggi, A., & Rimassa, G. (2001). JADE: A FIPA2000 compliant agent development environment. *Proceedings of the Fifth International Conference on Autonomous Agents*, 216–217. <https://doi.org/10.1145/375735.376120>
- Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1), 65–98. <https://doi.org/10.1137/141000671>
- Chen, F., Ren, W., & others. (2019). On the control of multi-agent systems: A survey. *Foundations and Trends® in Systems and Control*, 6(4), 339–499. <https://doi.org/10.1561/26000000019>
- Datseris, G., Vahdati, A. R., & DuBois, T. C. (2022). Agents.jl: A performant and feature-full agent-based modeling software of minimal code complexity. *SIMULATION*, 0(0), 003754972110688. <https://doi.org/10.1177/00375497211068820>
- Gronauer, S., & Diepold, K. (2022). Multi-agent deep reinforcement learning: A survey. *Artificial Intelligence Review*, 55(2), 895–943. <https://doi.org/10.1007/s10462-021-09996-w>
- Holly, S., & Nieße, A. (2021). Dynamic communication topologies for distributed heuristics in energy system optimization algorithms. *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*, 191–200. <https://doi.org/10.15439/2021F60>
- Kazil, J., Masad, D., & Crooks, A. (2020). Utilizing Python for agent-based modeling: The mesa framework. In R. Thomson, H. Bisgin, C. Dancy, A. Hyder, & M. Hussain (Eds.), *Social, cultural, and behavioral modeling* (pp. 308–317). Springer International Publishing. https://doi.org/10.1007/978-3-030-61255-9_30
- Lützenberger, M., Küster, T., Konnerth, T., Thiele, A., Masuch, N., Heßler, A., Keiser, J., Burkhardt, M., Kaiser, S., & Albayrak, S. (2013). JIAC V: A MAS framework for industrial applications. *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, 1189–1190.

- Russell, S. J., & Norvig, P. (2010). *Artificial intelligence a modern approach*. Prentice Hall.
- Schrage, R., & Nieße, A. (2023). Influence of adaptive coupling points on coalition formation in multi-energy systems. *Applied Network Science*, 8(1). <https://doi.org/10.1007/s41109-023-00553-8>
- Schrage, R., Sager, J., Hörding, J. P., & Holly, S. (2024). Mango: A modular Python-based agent simulation framework. *SoftwareX*, 27, 101791. <https://doi.org/10.1016/j.softx.2024.101791>
- Stark, S., Volkova, A., Lehnhoff, S., & Meer, H. de. (2021). Why your power system restoration does not work and what the ICT system can do about it. *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, 269–273. <https://doi.org/10.1145/3447555.3465415>
- Tiemann, P. H., Nebel-Wenner, M., Holly, S., Frost, E., Jimenez Martinez, A., & Nieße, A. (2022). Operational flexibility for multi-purpose usage of pooled battery storage systems. *Energy Informatics*, 5(1), 1–13. <https://doi.org/10.1186/s42162-022-00209-4>
- Tisue, S., & Wilensky, U. (2004). Netlogo: A simple environment for modeling complexity. *International Conference on Complex Systems*, 21, 16–21.
- Winikoff, M. (2005). JACK™ intelligent agents: An industrial strength platform. *Multi-Agent Programming: Languages, Platforms and Applications*, 175–193. https://doi.org/10.1007/0-387-26350-0_7
- Yang, T., Yi, X., Wu, J., Yuan, Y., Wu, D., Meng, Z., Hong, Y., Wang, H., Lin, Z., & Johansson, K. H. (2019). A survey of distributed optimization. *Annual Reviews in Control*, 47, 278–305. <https://doi.org/10.1016/j.arcontrol.2019.05.006>
- Zhang, L. (2022). *Agentframework (2.0.1)* [last access 07-08-2024]. <https://github.com/agentframework/agentframework>