Università di Genova
DIBRIS DIPARTIMENTO DI INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

S&DAI

P2P electric prosumers and MAS

Lorenzo Foschi

Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Creative

Hard

Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

The proposal seeks to **integrate MAS (Multi-Agent System) knowledge into an existing scholarship research project focused on an energy prosumers** network tool, namely STONKSpp. It aims to achieve a real BDI (Belief-Desire-Intention) approach in this context through three key directions:

- **Theoretical Exploration of BDI Architecture for Prosumers**: develop a theoretical model, potentially supported by code mockups, for extending prosumer agent design towards a real BDI architecture.

- **Implementation of BDI Prosumers:**
Implement BDI prosumers based on the model from point 1, using necessary simplifications to meet the project's 7-10 day timeline, utilize the **Python-Agentspeak** framework and compare implementations with the Jason framework.

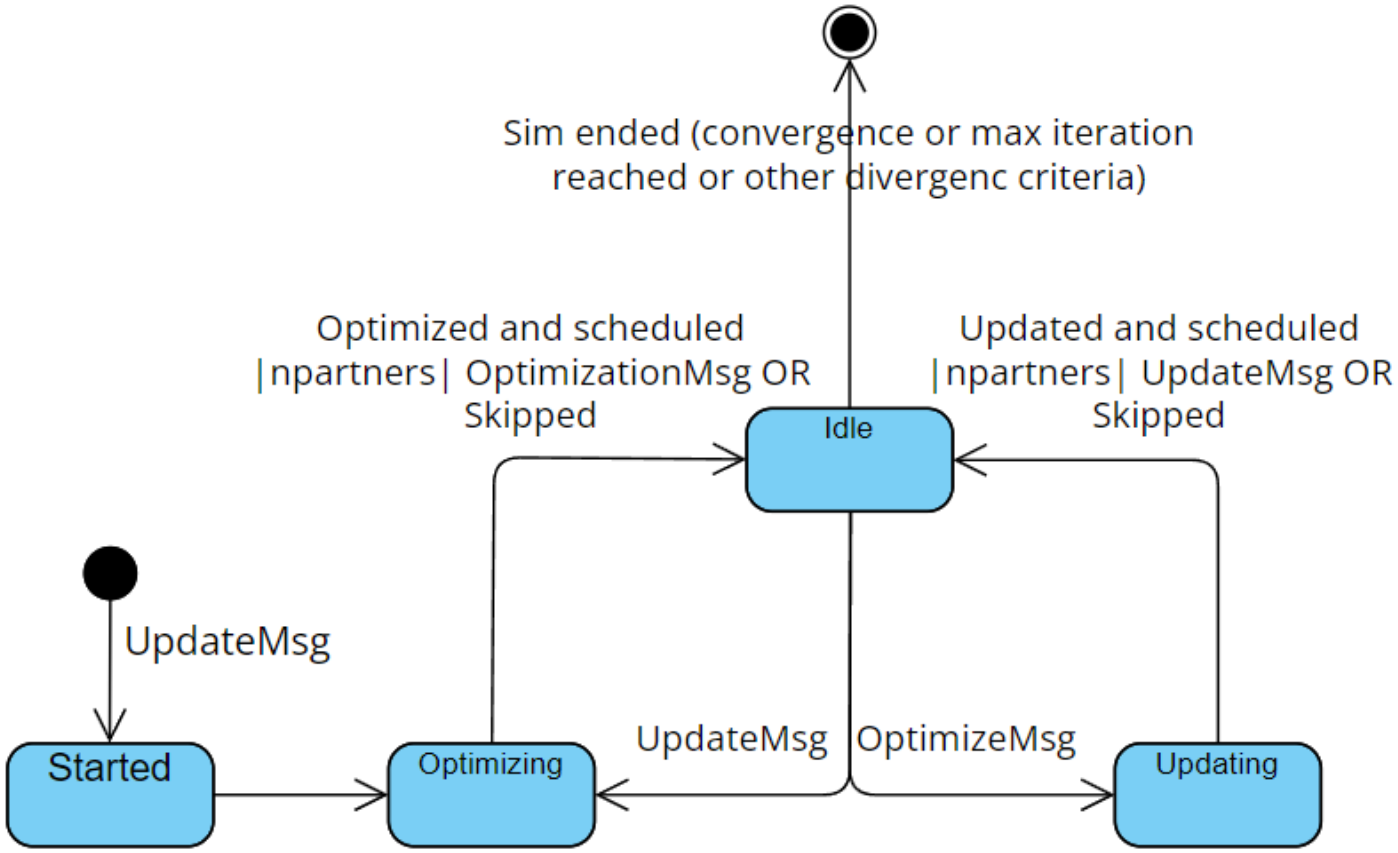
- Integration of **Mango** agents into the paper (trial)

Motivation of the project

Prosumer’s smart grids are one of the most prominent multi-agent-related fields:

- As cited in CETINA’s seminars on Research in Artificial Intelligence for the Energy and Infrastructure Sector (held by S&DAI prof. V.Mascardi), **intelligent agents have demonstrated their value in peer-to-peer energy trading** and managing dynamic scenarios, highlighting their ability to adapt to rapidly changing environments.
- In my research paper I already implicitly made some steps towards MAS, **moving to a decentralized approach with real-message passing among multiple independent entities**.

Message-passing paradigm



First step made in STONKSpp paper for transitioning from a fully centralized approach to a discrete event simulation: the initial implementation of Baroche's prosumer markets assumed a perfect network with no message loss or delays, where nodes received all messages synchronously and processed them in nested loops.

Extensibility towards BDI approach (1)

Enviromental definition - Given that the environment is currently scattered across multiple entities we should first identify these and consequently study an ideal extraction of the environment with the following capabilities:

- Storing global states (e.g. network topology, global time, and external constraints).
- Providing environment-related **services** for agents (e.g., query/obtain the neighbor set, get network latencies, or check resource constraints).

Extensibility towards BDI approach (2)

Prosumer def	Currently	With BDI
Beliefs	Trades, partner list, a/b cost function parameters, rho penalty factor, estimated electricity demand, observed electricity usage, local/global prices	Set of beliefs that must be true
Desires	Minimizing costs, maximizing social welfare, achieving local constraints (e.g local conspumption <= production + battery), cooperate with neighbors	Body of the plan to execute
Intentions	Performing local optimization, sending trade proposals to neighbor X with a certain quantity, receiving a trade proposal	Triggering event due to message received & send message to neighbors

```
1. plan opt:
2.     trigger: message_received("UpdateMsg")
3.     context: believes("require_opt", True)
4.     body:
5.         update_local_model()
6.         run_gurobi()
7.         update_beliefs_with_new_solution()
8.         send_message_to_neighbors("OptimizeMsg")
```

Extensibility towards BDI approach (3)

Neighbors Awareness

- In a BDI setup, an agent's knowledge about its neighbors can definitely be dynamic, as it may discover new neighbors or lose existing ones due to changing environmental conditions or reconfiguration events.
- Additionally, an optional advanced extension involves incorporating **social awareness and trust**, allowing agents to track the reliability or trustworthiness of their neighbors.

Communication Protocols

- **Inform**: "I inform you that my new supply is X kW"
- **Request**: "I request you to send me an updated price"
- **Propose**: "I propose an exchange of Y kW at price P" → Accept/Reject

Python-agentspeak implementation (1)

Agentspeak file of one of the two implemented Prosumer, in its first version (static neighbors and without introduced randomness)

```
/* Beliefs */
available_power(100).
demand(50).
neighbor(prosumer2).

+!start_optimization : true
<- .print("Prosumer1: Starting optimization process...");
   !optimize_energy.

/* Optimize energy (Surplus > 0 => propose a trade) */
+!optimize_energy
:  available_power(Avail) & demand(Dem) & Surplus = (Avail - Dem) & (Surplus > 0)
<- .print("Prosumer1: Surplus of ", Surplus, " => proposing a trade...");
   !propose_trade(prosumer2, Surplus, 10).

/* Optimize energy (Surplus <= 0 => no surplus) */
+!optimize_energy
:  available_power(Avail) & demand(Dem) & Surplus = (Avail - Dem) & (Surplus <= 0)
<- .print("Prosumer1: Not enough surplus (Surplus=", Surplus, "). No trade possible.");
   !no_surplus(Surplus).

/* Propose a trade to another agent */
+!propose_trade(Neighbor, Quantity, Price) : true
<- .print("Prosumer1: Proposing", Quantity, "units at price", Price, "to", Neighbor, ".");
   .send(Neighbor, achieve, propose(Quantity, Price)).

/* If no surplus */
+!no_surplus(Surplus) : true
<- .print("Prosumer1: Surplus=", Surplus, ". No trade possible.").

/* Plans for receiving acceptance/rejection of a trade */
+propose_accept(Quantity, Price) : true
<- .print("Prosumer1: Proposal ACCEPTED for ", Quantity, " units at price ", Price, ".");
   !propose_second_trade.

+propose_reject(Quantity, Price) : true
<- .print("Prosumer1: Proposal REJECTED for ", Quantity, " units at price ", Price, ").

+!propose_second_trade : true
<- .print("Prosumer1: Preparing a second trade proposal...");
   !propose_trade(prosumer2, 90, 15).
!start_optimization.
```

Python-agentspeak implementation (2)

Adding neighbor discovery to the equation

```
/* Beliefs */
registered([]).

/* Register a new list of prosumers */
+!register(Prosumers) : true
<- registered(Current);
   New = .concat(Current, Prosumers);  /* Concatenate the new prosumers */
   -registered(Current);
   +registered(New);
   .print("DF: Registered prosumers: ", New, ").

/* Provide the list of registered prosumers */
+!get_prosumers : true
<- registered(Prosumers);  /* Retrieve the list of registered prosumers */
   .print("DF: Sending list of registered prosumers: ", Prosumers, ");
   .send(prosumer1, tell, prosumer_list(Prosumers)).
```

```
/* Beliefs */
available_power(100).
demand(50).
neighbors([]).

/* Register itself with the DF */
+!start_discovery : true
<- .print("Prosumer1: Registering with DF...");
   .send(df, achieve, register([prosumer1]));
   !retrieve_neighbors.

/* Retrieve neighbors from the DF */
+!retrieve_neighbors : true
<- .print("Prosumer1: Retrieving neighbors from DF...");
   .send(df, achieve, get_prosumers).

+prosumers_list(Prosumers) : true
<- .print("Prosumer1: Received prosumer list: ", Prosumers, ");
   -neighbors(_);
   +neighbors(Prosumers);
   !start_optimization.
```

Python-as and Jason… are equivalent!

Python-agentspeak

Deeply inside, Python-agentspeak is a set of lexer/AST/parser classes which allow to use agentspeak syntax in a python environment. Basically, it's like having Jason but with a Python environment behind instead of a Java one. Apart from...

```
@actions.add(functor="set_belief", arity=2)
def set_belief(agent, term, intention):
    belief_name = str(term.args[0])
    belief_value = int(term.args[1])

    agent.beliefs[belief_name].add((belief_value,))
    yield
with open(os.path.join(os.path.dirname(__file__), "prosumer1.asl")) as f:
    prosumer1 = env.build_agent(f, actions)

p1_power = random.randint(50, 120)
p1_demand = random.randint(0, 80)
prosumer1.call(
    Trigger.addition,
    GoalType.achievement,
    Literal("init_beliefs", (p1_power, p1_demand)),
    Intention()
)
prosumer1.call(
    Trigger.addition,
    GoalType.achievement,
    Literal("start_discovery"),
    Intention()
)
```

Jason

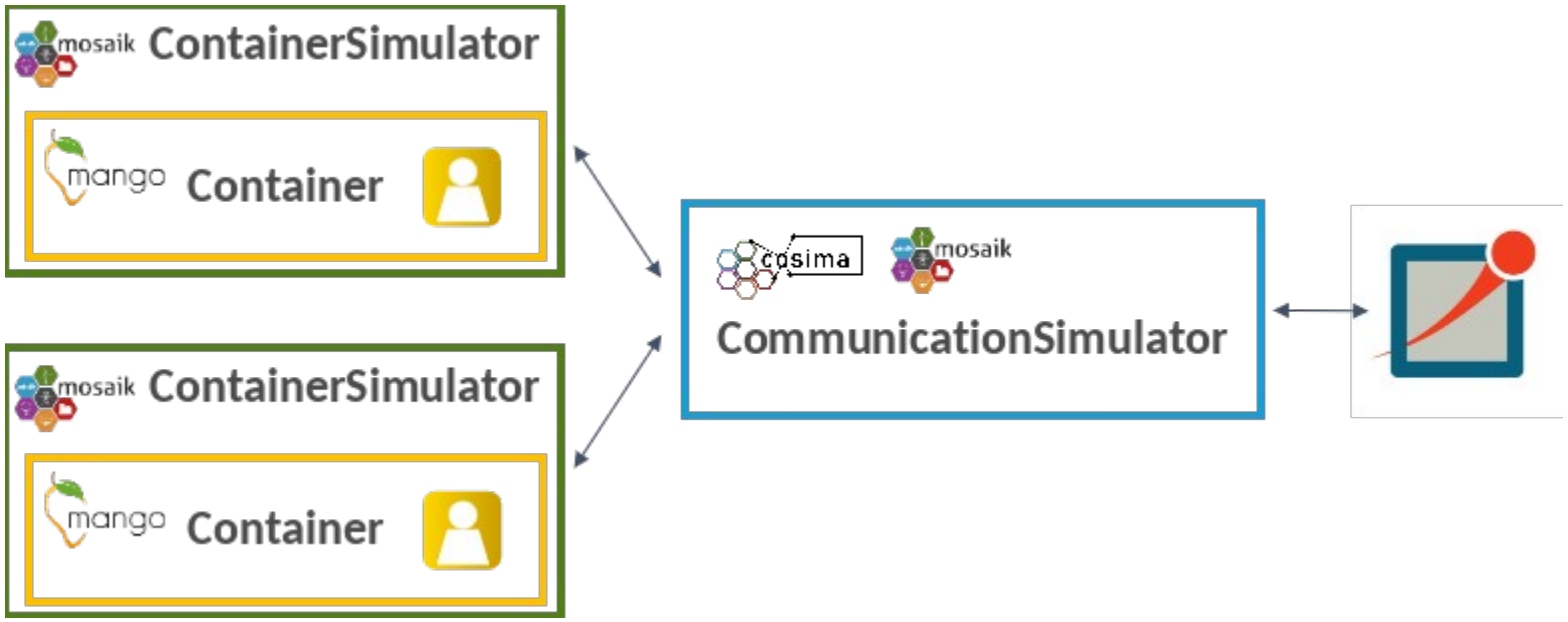
```
public void init(String[] args) {
        super.init(args);
        Random rand = new Random();
        String[] agents = {"prosumer1", "prosumer2"};

        for (String agent : agents) {
            int randPower = 50 + rand.nextInt(101);
            String beliefStr = "available_power(" + randPower + ")";
            Literal belief = Literal.parseLiteral(beliefStr);
            addPercept(agent, belief);
            logger.info("Added belief to " + agent + ": " + beliefStr);
        }

        int demand = 50 + rand.nextInt(51);
        String demandStr = "demand(" + demand + ")";
        Literal demandBelief = Literal.parseLiteral(demandStr);
        addPercept("prosumer1", demandBelief);
        logger.info("Added belief to prosumer1: " + demandStr);

        String neighborStr = "neighbor(prosumer2)";
        Literal neighborBelief = Literal.parseLiteral(neighborStr);
        addPercept("prosumer1", neighborBelief);
        logger.info("Added belief to prosumer1: " + neighborStr);
    }
```

Mango framework in STONKSpp?

	PROS	CONS
Python-agentspeak	Simplicity, 1:1 with Jason, strict BDI syntax, FIPA standard	Less powerful, not fully leverage python, no codecs/TCP/MQTT support
Mango	Codecs & TCP/MQTT support, power and features, FIPA standard	Complexity, not directly comparable with agentspeak




How to use Mango?



The general procedure implies implementing the code into a new kind of simulator, namely Mango Container, wrapping it into the Mosaik's ContainerSimulator. The latter will exchange messages thanks to Cosima CommunicationSimulator, as it's currently happening with basic Cosima Simulators.

Why is it hard?

At first glance, to implement Mango in the current code, an easy procedure can be followed. However, the problem lies in how Prosumers are implemented.

Further steps and extensibility

Baroche, T., Moret, F., Pinson, P., 2019. Prosumer markets: A unified formulation	Completely centralized, monolithic and sequential prosumer exchange code. Not suitable for network simulation.
Towards STONKSpp (1)	Transitioning to message-passing paradigm, but still centralized
Towards STONKSpp (2)	Abstracting prosumers as entities on the network, with 1:1 mapping between exchanged messages from X and proposed trades from the same X
L.Foschi, M.Dell'Amico, G.Ferro, G.Longo, E.Russo. 2025 Software Tool for Orchestrating Networks: a Kit of Simulators of P2P Prosumers	Network simulation over prosumers, with many ways to simulate traffic, delays, algorithms, byzantine behavior and so on.
Towards Mango	Re-implementation of Prosumers as real Cosima simulators, a heavy task!
Actual Mango implementation	Wrapping Mango across Prosumer agents, as done before with the whole Simulator object

References

L.Foschi, M.Dell'Amico, G.Ferro, G.Longo, E.Russo. 2025 Software Tool for Orchestrating Networks: a Kit of Simulators of P2P Prosumers – near the review process

Baroche, T., Moret, F., Pinson, P., 2019. Prosumer markets: A unified formulation, in: 2019 IEEE Milan PowerTech, pp. 1–6. doi:10.1109/PTC. 2019.8810474.

Bukar, A.L., Hamza, M.F., Ayub, S., Abobaker, A.K., Modu, B., Mohseni, S., Brent, A.C., Ogbonnaya, C., Mustapha, K., Idakwo, H.O., 2023. Peer-to-peer electricity trading: A systematic review on current developments and perspectives. Renewable Energy Focus 44, 317–333. URL:https://www.sciencedirect.com/science/article/pii/S1755008423000091, doi:10.1016/j.ref.2023.01.008.

Viviana Mascardi, Unige, May 3 2024, Seminars on Research in AI for the Energy and Infrastructure Sector (CETINA)

Oest, F., Frost, E., Radtke, M., Lehnhoff, S., 2022. Coupling omnet++ and mosaik for integrated co-simulation of ict-reliant smart grids. arXiv preprint arXiv:2209.12550 doi:10.48550/arXiv.2209.12550.

Ofenloch, A., et al., 2022. Mosaik 3.0: Combining time-stepped and discrete event simulation, in: 2022 Open Source Modelling and Simulation of Energy Systems (OSMSES), pp. 1–5. doi:10.1109/OSMSES54027.2022.9769116.

Varga, A., Hornig, R., 2008. An overview of the omnet++ simulation environment, in: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools), p. 60. doi:10.1145/1416222.1416290.

Nair, A.S., Hossen, T., Campion, M. et al. Multi-Agent Systems for Resource Allocation and Scheduling in a Smart Grid. Technol Econ Smart Grids Sustain Energy 3, 15 (2018). https://doi.org/10.1007/s40866-018-0052-y

Deng, Y., An, B. Utility distribution matters: enabling fast belief propagation for multi-agent optimization with dense local utility function. Auton Agent Multi-Agent Syst 35, 24 (2021).https://doi.org/10.1007/s10458-021-09511-z

Inês F.G. Reis, Ivo Gonçalves, Marta A.R. Lopes, Carlos Henggeler Antunes, (2022) Towards inclusive community-based energy markets: A multiagent framework

UniGe

DIBRIS