Smart Grid and Renewable Energy, 2023, 14, 183-196
https://www.scirp.org/journal/sgre
ISSN Online: 2151-4844
ISSN Print: 2151-481X

Smart Python Agents for Microgrids

Salem Al-Agtash[1,2], Xueyan Xian[3], Kristina Petroshchuk[3], Heba Syeddah[3]

[1]Department of Computer Science and Engineering, Santa Clara University, Santa Clara, CA, USA
[2]Department of Computer Engineering, German Jordanian University, Amman, Jordan
[3]Department of Computer Science and Information Technology, Mission College, Santa Clara, CA, USA
Email: salem.alagtash@wvm.edu, xxian2@mywvm.wvm.edu, kpetrosh@mywvm.wvm.edu, hsyeddah@mywvm.wvm.edu

How to cite this paper: Al-Agtash, S., Xian, X.Y., Petroshchuk, K. and Syeddah, H. (2023) Smart Python Agents for Microgrids. *Smart Grid and Renewable Energy*, **14**, 183-196.
https://doi.org/10.4236/sgre.2023.1410011

Received: August 25, 2023
Accepted: October 28, 2023
Published: October 31, 2023

Abstract

Microgrids are revolutionary power systems that interconnect a mix of renewable power generation, load, storage systems, and inverters in a small-scale grid network. Operating microgrids while maintaining a consistent grid voltage and frequency during the islanding and disruption of renewables has been a challenging research problem. In this paper, a preliminary microgrid agent implementation is presented using SPADE (Smart Python Agent Development Environment) as a powerful development framework that has been used extensively in many application domains. Agents autonomously managed and operated microgrid individual components. A multiagent microgrid system was modeled to seamlessly operate and optimize energy balance by coordinating the actions of agents. Agents were built to forecast energy demand and solar power and coordinate to balance generation with load while maintaining optimal power flow and adequate network voltage and frequency.

Keywords

Microgrids, Agents, SPADE, Renewable Power

1. Introduction

Microgrids integrate renewable generation with distributed load in a small-scale grid network. Renewables are intermittent and therefore bring various operational challenges. Abrupt variations in generation and demand result in microgrid instability unless it is tightly coupled with the utility grid that provides enough spinning reserves [1] [2] [3] [4]. Research has identified intelligent agents as viable technologies that can operate individual microgrid components in a distributed architecture to balance power generation and demand while maintaining consistent voltage and frequency [5] [6] [7].

A microgrid is a group of interconnected loads and distributed energy resources in a single controllable entity that transits between grid-connected and islanded modes [8]. In a grid-connected mode, the microgrid operates in sync with the power grid, trading in and out power deficit and excess. In an islanded mode, the microgrid thrives on balancing generation and load in real time and maintaining reliable power flow [9] [10]. In remote and fragile areas, microgrids are isolated with no interconnection to the grid and operate as stand-alone power systems. Microgrid success hinges on standardization, technical and economic considerations, and robust control mechanisms.

Operating microgrids has been a challenging research problem. This is due to the intermittent nature of renewable resources and the lack of adequate spinning reserves to accommodate imbalances. When properly operated, microgrids seamlessly integrate renewables and shape the energy landscape [11]. The conventional Supervisory Control and Data Acquisition (SCADA) that has inherently been used in utility grids is expensive to customize and deploy in microgrids. The intermittent nature of renewables requires more sophisticated control platforms and intelligent software to maintain the generation load balance and to keep voltage levels within limits in real time.

Furthermore, SCADA protocols such as Rockwell Automation RSview32 have interoperability limitations in microgrid devices. Autonomous software agents are promising microgrid control technologies [12]. Java and Python have emerged as *de facto* programming tools for developing multi-agents in JADE (Java Agent Development Environment) and SPADE, respectively but still face various challenges such as communication, elasticity, scalability, and interoperability with human and systems [13] [14] [15].

Multiagent models in JADE use the Java Messaging Service (JMS) protocol [16]. JADE has been widely used to develop multi-agents in large-scale systems [17]. It employs simple reactive agents that are responsive to environmental changes. JADE comes with GUI tools, can be combined with MATLAB [18], and has been used in microgrid tertiary control and energy management systems [19]. It implements an agent platform that autonomously operates in the background and uses real-time event-handling protocols [20].

In contrast, SPADE supports XMPP, HTTP, and TCP communication protocols, allowing seamless integration with agents and humans [21] [22] [23] [24] [25]. It is open, scalable, flexible, and supports Python with a wide range of AI tools with a small memory footprint and requires fewer system resources, making it suitable for deployment on IoT devices and embedded systems. SPADE agents utilize a behavior-based model that combines classical behaviors (such as one-shot, periodic, or finite-automata) with BDI behavior. This allows for integrating procedural, object-oriented, and logic programming within the same agent. SPADE leverages XMPP (eXtensible Messaging and Presence Protocol), allowing open, decentralized communication between agents, artifacts, humans, and other entities. It employs an asynchronous programming paradigm with a wide range of interface tools and libraries [26].

Microgrid implementations are demonstrated in various GitHub repositories. MicrGrid and OpenAI Gym provide tools to simulate microgrid optimization; OpenModelica supports reinforcement learning in microgrid operations and control [27]; and PyMgrid simulator serves as a powerful tool for studying and experimenting with AI algorithms in the context of microgrid systems [28] [29].

However, no research has yet reported on the use of SPADE in microgrid multi-agent implementation. SPADE is an emerging powerful tool that has been demonstrated in other multi-agent application domains. This paper presents smart Python agents in SPADE to autonomously operate microgrid components, including load, solar, wind, battery, and control. Unlike JADE and other Agent platforms, SPADE is built in Python, which has powerful tools, libraries, and packages in machine learning and data analytics. It also supports XMPP seamless messaging communications and has become the *de facto* AI programming language. Microgrid historical data were used from Santa Clara University and Kaggle to simulate SPADE agents. The results show promising agent technologies for accurate forecasting, operation, and control.

The remaining sections are organized as follows: Section 2 presents the microgrid agent architecture. In Section 3, the smart Python agent development is presented. Section 4 illustrates microgrid SPADE for load, solar, wind, and control. Finally, the paper is concluded in Section 5.

2. Microgrid Agents

Figure 1 shows a sample microgrid topology that consists of renewable wind and photovoltaic (PV) power generating units, controllable load, battery energy storage system (BESS), AC (Alternating Current)/DC (Direct Current) inverters, AC bus system, and a substation to couple with the utility grid. A microgrid can also include other resources, such as small-scale diesel, fuel cell, or natural gas generating units and electric vehicles. A design of an agent control was made for a wind turbine, solar PV, load, inverter, BESS, and the system operator that manages and operates the microgrid in real-time to ensure power availability and reliability.

Agents are designed as autonomous intelligent nodes to operate microgrid components. Agents perceive the microgrid environment through an array of input sensors and output controls and coordinate with the other agents in a distributed manner. **Figure 2** shows the agent architectural design in a microgrid environment. It consists of a cognitive agent, knowledge base, coordination, and communication.

The design model of the cognitive agent model-based reflex and goal- and utility-based simplex learning. The cognitive agent typically involves the integration of artificial intelligence (AI) and machine learning (ML) techniques to enable intelligent decision-making and optimization within the microgrid system. It implements the following services:

1) Data Acquisition: gathering data from various sensors, agents, and other sources.

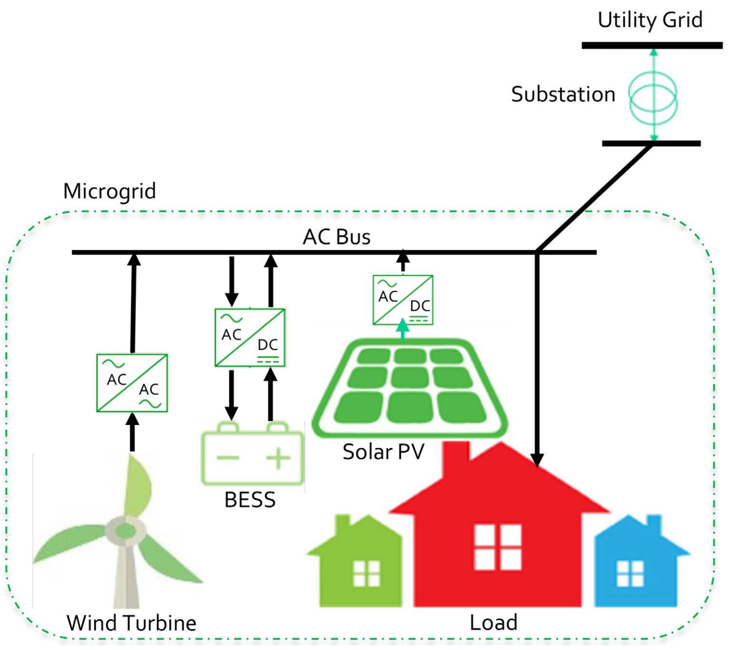

Figure 1. Sample microgrid topology.

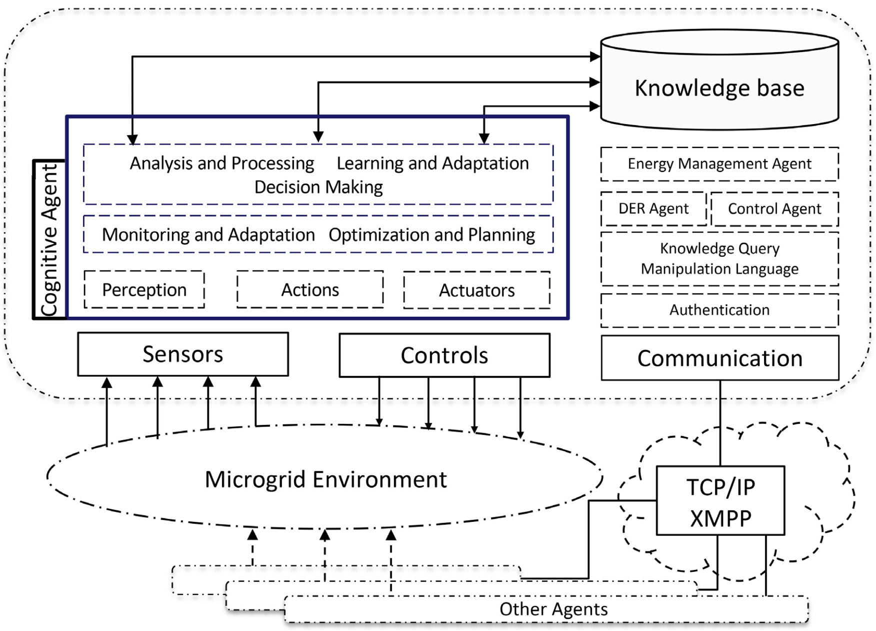

Figure 2. Agent architectural design.

2) Data Processing: processing acquired data using techniques such as data filtering, normalization, and feature extraction. This step helps in transforming raw data into meaningful information that can be used for analysis and decision-making.

3) Monitoring and Adaptation: continuously monitoring the performance of microgrid components and adapting its decision-making strategies as needed. Agents detect anomalies, identify failures, and initiate appropriate or corrective actions.

4) Knowledge Base: maintaining a knowledge base that contains information about the microgrid's components, operating constraints, historical data, and relevant domain knowledge. This knowledge base is a reference for the agent's machine learning and decision-making process.

5) Machine Learning: intensively using supervised, unsupervised, and reinforcement machine learning to analyze historical data and improve decision-making capabilities.

6) Decision-Making: making operator-like decisions to optimize microgrid operation with respect to generation, storage, demand response, and fault detection and diagnosis.

7) Optimization and Planning: optimization and planning to enhance the efficiency, reliability, and cost-effectiveness of microgrids through energy dispatch, scheduling energy resources, and battery storage management based on real-time conditions and future predictions.

8) Actions/Responses: generating actions or responses based on the decision-making process. Actions include adjusting the settings of energy resources, regulating energy flows, initiating control commands to devices within the microgrid, and communicating with external entities such as energy markets.

3. Smart Python Agent Development

The SPADE platform was used for development with XMPP communication, commonly used in instant messaging applications. This allows software agents to interact with agents and humans. Each agent can define one or more behaviors, which are executed independently by the platform. The message dispatcher is responsible for delivering incoming messages to the corresponding behaviors. SPADE offers different behavior types, including CyClicBehavior (runs continuously until the agent is stopped), OneShotBehavior (runs once and is then destroyed), PeriodicBehavior (runs at predefined intervals), TimeoutBehavior (a subtype of OneShotBehavior that runs after a timeout), and finite-state machine behaviors. CyclicBehavior serves as a basis for other behavior types [30].

3.1. SPADE Agents

The SPADE framework was built on a multi-agent system (MAS) architecture [31]. The MAS architecture is decentralized because there is no central control or coordination mechanism that directs agents' behavior. Instead, each agent operates independently and interacts with other agents through a communication protocol. Agents in the MAS architecture are autonomous in the decision-making and action-taking process based on their own knowledge and objectives. Agents in the MAS architecture are also heterogeneous, with different capabilities and behaviors. Agent communication protocol defines the language and format of messages agents exchange information and coordinate their actions [32].

Figure 3 shows the SPADE agent in a microgrid. Agents operate Solar, Load, Wind, Storage, and Utility Interconnection. They possess limited autonomous capabilities and interact with each other via the XMPP server to produce renewable power that meets the end-user power demand while maintaining a reliable microgrid power network [33]. Agent-based microgrid offers various advantages such as flexibility, resilience, and effective management in the increasing complexity of distributed renewable energy resources [34]. In addressing the complex tasks and challenges of distributed energy generation and control, varieties of agent design models are explored. Accurate component modeling and coordination between agents are crucial for effective control [35]. Software agent has become a powerful and rapidly growing technology in power systems, enabling intelligent and autonomous agents to operate in microgrid environments.

3.2. Classes and Plug-Ins

In SPADE, generic agents were built with messaging, directory facilitator, and XMPP plug-ins, defined as follows:

1) Python agent class—inherits from the spade. Agent base class. The class provides a set of functions that agents use to initialize themselves, send and receive messages, and manage their own life cycle.

2) Behavior—set of instructions that agents follow to achieve specific tasks. Behaviors are implemented as Python classes that inherit from the spade. Behavior base class. Such tasks include how an agent responds to events.

3) Message—Agents communicate with one another through message exchange. Messages can contain many types of data—text, images, or numerical data.

4) Directory Facilitator—a component that provides a mechanism for agents to discover and communicate with each other. It is a central registry where agents can register their services and capabilities. Other agents can then query to find agents that can provide the required services.

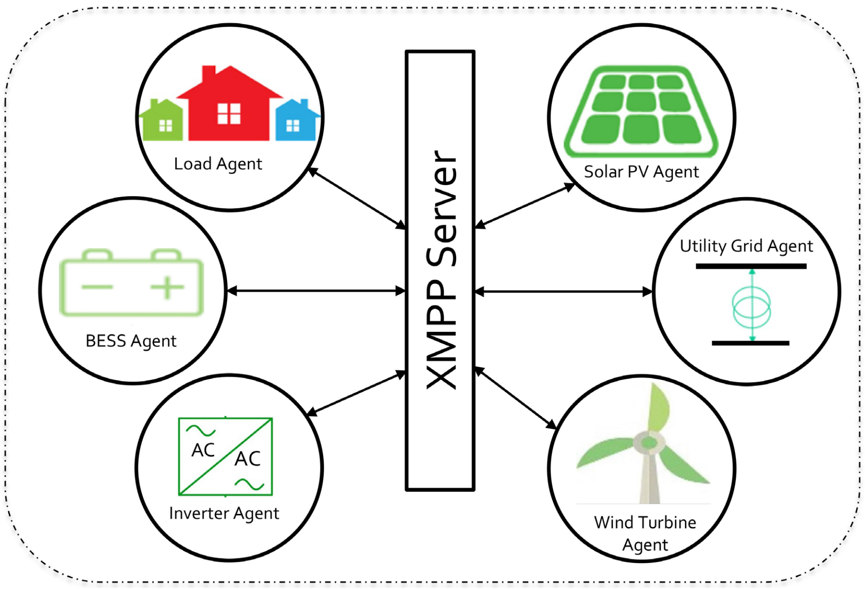

Figure 3. SPADE Agent Architecture.

5) XMPP library—allows agents to communicate with each other using the Extensible Messaging and Presence Protocol (XMPP). XMPP is a widely used protocol for real-time communication. This library provides functions for sending and receiving XMPP messages, as well as for handling XMPP presence notifications.

6) Plug-in architecture—allows developers to extend the functionality of the framework. Plug-ins can be used to add features of logging, visualization, and monitoring.

7) Communication library—provides a messaging system that allows agents to communicate with each other using various messaging protocols—XMPP, RabbitMQ, and ZeroMQ. It also provides APIs for sending and receiving messages, subscribing to message types, and filtering messages.

The details of the agent functions and services in communication, behavior, directory, and XMPP are given in the **Appendix**.

3.3. Behavior

Figure 4 shows a sample Python code implementation of an agent. The agent is a subclass of the SPADE agent and inherits its behavior and communication methods, and attributes. The behavior represents the main logic of the agent, generally forecasting, operations, and controls. Each microgrid agent has a distinct behavior that represents its functionality, and these are inherited from Spade. Pykqml was used for agent communication based on the KQML performative protocol [36]. Pandas, Tensorflow, Keras, and Sklearn were used as the main data science and machine learning libraries to implement the agent's forecasting behavior. Pymgrid, gym, and Pyomo were also used to implement the agent's power dispatch, power flow, control, and energy management.

4. Illustrative Example

PyCharm software package was used to develop load, solar, wind, battery and control SPADE agents. Each agent is identified with its own XMPP credentials on blah.im server that are used for message exchange. Agents use KQML performatives as their communication protocol.

```
class SolarBehavior(behaviour.CyclicBehaviour):
    async def run(self):
        await asyncio.sleep(1)
        print("solar behavior running")
        msg_solar = await.self.receive(..)
        if msg_solar:
            pv_forcast = PVforecast()
            await pv_forecast.run()
        else:
            print("no message received")
    async def on_end(self):
        print("Ending behavior")
async def setup(self):
    behavior = self.SolarBehavior()
    template = Template()
    template.set_metadata("performative",..)
    self.add_behaviour(behavior, template)
```

Figure 4. Sample Python code implementation of an Agent.

4.1. Load Agent

The 2021-2022 load historical data were used for a sample microgrid to illustrate the load agent's forecasting behavior in six buildings triggered by the control agent in msg.metadata = {"performative": "ask-about", "acl-representation": "xml"}. 75% and 25% of the data were used for training and testing, respectively. Figure 5 shows power predictions based on the training and testing data on a multi-layer network with 100,000 epochs for each building, with 1.68 and 3.35 RMSE (Root Mean Square Error) between the actual and the prediction training and testing data, respectively.

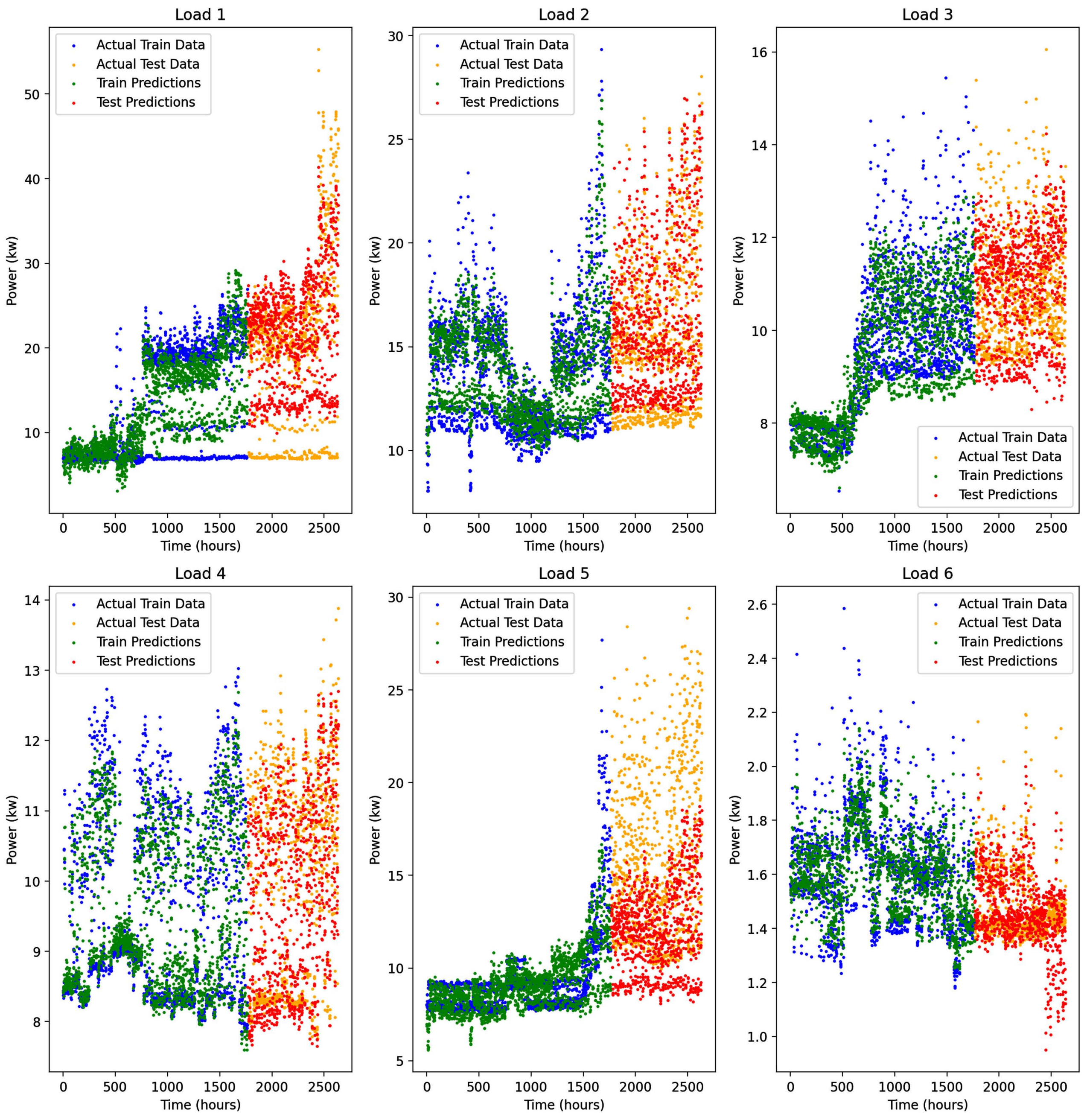

Figure 5. Power predictions in 6 buildings.

4.2. Wind Agent

The microgrid wind historical data from Kaggle were used in a multi-layer machine-learning network with inputs representing wind speed and direction and the output representing the wind-produced power. Figure 6 shows power predictions and the actual power values for both training and testing. 75% and 25% of the data were used for training and testing, respectively with 100,000 epochs run. The RMSE between the actual and the prediction training and testing data is 5.38 and 7.27, respectively.

4.3. Solar Agent

The microgrid PV historical data were used from Kaggle in a multi-layer machine-learning network with inputs representing ambient and module temperatures and irradiation and the output representing the PV-produced power. Figure 7 shows power predictions and the actual power values for both training and testing. 75% and 25% of the data were used for training and testing, respectively, with 100,000 epochs run. The RMSE between the actual and the prediction training and testing data is 2.32 and 1.21, respectively.

Agents have also been developed for the battery and the control. The control agent communicates with the microgrid agents to acquire operational and forecasting data to run several levels of control. The tertiary, secondary, and primary controls are being built for economic dispatch, optimal power flow, voltage and frequency controls, and energy management scenarios. At this stage, the control agent implementation is in progress in an autonomous and decentralized architecture.

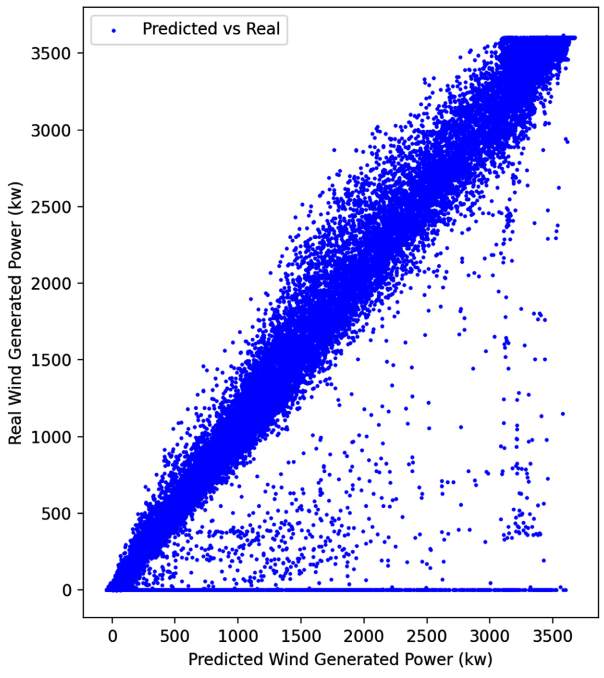

Figure 6. Predicted versus Real Power.

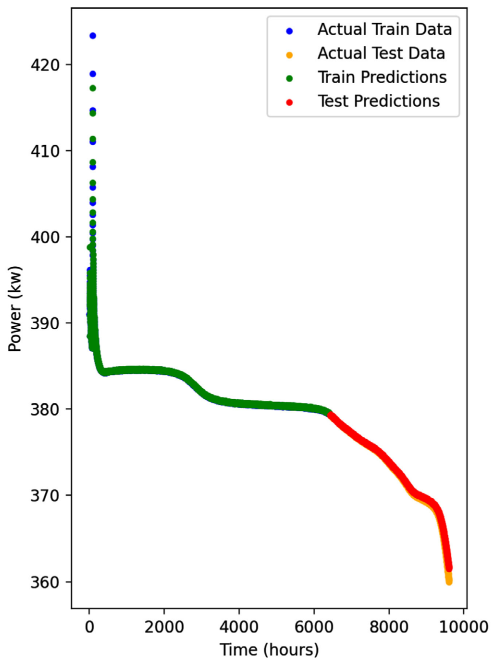

Figure 7. Solar power prediction.

5. Conclusion

This paper presents smart Python agents for microgrids using SPADE. Agents emerge as viable technologies to seamlessly operate a mix of microgrid components in a distributed architecture. Agents are designed to embed intelligence and autonomous capabilities and coordinate with each other to balance generation and demand and maintain a reliable microgrid network. The SPADE agent software has been simulated for load, solar, and wind using microgrid historical data from Kaggle and Santa Clara University. Preliminary results show accurate forecasts for the control agent to run the microgrid network's optimal economic and power flow schedules and operate a reliable and stable small-size power system.

Fund

This work was funded by the HSI-STEM Summer research grant at Mission College, the West Valley-Mission Community College District, Santa Clara, California.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Dan, T. and Merrill, S. (2012) The U.S. Department of Energy's Microgrid Initiative. *The Electricity Journal*, **25**, 84-94. https://doi.org/10.1016/j.tej.2012.09.013

[2] Lasseter, B. (2001) Microgrids [Distributed Power Generation]. *Proceedings of the IEEE Power Engineering Society Winter Meeting*, **1**, 146-149.

[3] Lasseter, R. (2002) MicroGrids. *Proceedings of the IEEE Power Engineering Society Winter Meeting*, **1**, 305-308.

[4] Li, Y., Nejabatkhah, F. and Tian, H. (2023) Power Management System (PMS) in Smart Hybrid AC/DC Microgrids. In: *Smart Hybrid AC/DC Microgrids: Power Management, Energy Management, and Power Quality Control*, Wiley, Hoboken, 121-154. https://doi.org/10.1002/9781119598411

[5] Khan, M. and Wang, J. (2017) The Research on Multi-Agent Systems for Microgrid Control and Optimization. *Renewable and Sustainable Energy Reviews*, **80**, 1399-1411. https://doi.org/10.1016/j.rser.2017.05.279

[6] Eddy, Y., Gooi, H. and Chen, S. (2014) Multi-Agent System for Distributed Management of Microgrids. *IEEE Transactions on Power Systems*, **30**, 24-34. https://doi.org/10.1109/TPWRS.2014.2322622

[7] Shrivastwa, R., Hably, A., Melizi, K. and Bacha, S. (2019) Understanding Microgrids and Their Future Trends. *IEEE International Conference on Industrial Technology*, Melbourne, 13-15 February 2019, 1723-1728. https://doi.org/10.1109/ICIT.2019.8754952

[8] De Jaeger, E. (2020) Microgrids: Non-Exhaustive Review of Technical Issues 2017. https://documents.pub/document/microgrids-non-exhaustive-review-of-technical-issues-4-use-case-scenarios-for.html?page=1

[9] Pompodakis, E., Kryonidis, G. and Alexiadis, M. (2020) A Comprehensive Load Flow Approach for Grid-Connected and Islanded AC Microgrids. *IEEE Transactions on Power Systems*, **35**, 1143-1155. https://doi.org/10.1109/TPWRS.2019.2945011

[10] Awal, M., Yu, H., Tu, H., Lukic, S. and Husain, I. (2020) Hierarchical Control for Virtual Oscillator Based Grid-Connected and Islanded Microgrids. *IEEE Transactions on Power Electronics*, **35**, 988-1001. https://doi.org/10.1109/TPEL.2019.2912152

[11] Chartier, S.L., Venkiteswaran, V.K., Rangarajan, S.S., Collins, E.R. and Senjyu, T. (2022) Microgrid Emergence, Integration, and Influence on the Future Energy Generation Equilibrium—A Review. *Electronics*, **11**, Article No. 791. https://doi.org/10.3390/electronics11050791

[12] Feroze, H. (2009) Multi-Agent System in Microgrids: Design and Implementation. Ph.D. Dissertation, Virginia Tech, Blacksburg.

[13] Sanchis, A., Julián, V., Corchado, J.M., Billhardt, H. and Carrascosa, C. (2014) Using Natural Interfaces for Human-Agent Immersion. In: Corchado, J.M., *et al.*, Eds., *Highlights of Practical Applications of Heterogeneous Multi-Agent Systems*, Springer, Cham, 358-367. https://doi.org/10.1007/978-3-319-07767-3_32

[14] Suganuma, T., Oide, T., Kitagami, S., Sugawara, K. and Shiratori, N. (2018) Multiagent-Based Flexible Edge Computing Architecture for IoT. *IEEE Network*, **32**, 16-23. https://doi.org/10.1109/MNET.2018.1700201

[15] Ayala, I., Amor, M. and Fuentes, L. (2015) The Sol Agent Platform: Enabling Group Communication and Interoperability of Self-Configuring Agents in the Internet of Things. *Journal of Ambient Intelligence and Smart Environments*, **7**, 243-269. https://doi.org/10.3233/AIS-150304

[16] Alseyat, A.D. and Park, J. (2019) Multi-Agent System Using JADE for Distributed DC Microgrid System Control. *Proceedings of the* 2019 *North American Power Symposium* (*NAPS*), Wichita, 13-15 October 2019 1-5. https://doi.org/10.1109/NAPS46351.2019.9000215

[17] Bergent, F., Caire, G., Monica, S. and Poggi, A. (2020) The First Twenty Years of Agent-Based Software Development with JADE. *Autonomous Agents and Multi-Agent Systems*, **34**, 36. https://doi.org/10.1007/s10458-019-09424-y

[18] Kanatamneni, A., Brown, L., Parker, G. and Weaver, W. (2015) Abhilash Survey of Multi-Agent Systems for Microgrid Control. *Engineering Applications of Artificial Intelligence*, **45**, 192-203. https://doi.org/10.1016/j.engappai.2015.07.005

[19] Rivera, S., Faid, A.M. and Youcef-Toumi, K. (2014) A Multi-Agent System Transient Stability Platform for Resilient Self-Healing Operation of Multiple Microgrids. 5*th IEEE PES Innovative Smart Grid Technologies Conference*, Washington DC, 19-22 February 2014, 1-5. https://doi.org/10.1109/ISGT.2014.6816377

[20] Colson, C.M., Nehrir, M.H. and Gunderson, R.W. (2011) Multi-Agent Microgrid Power Management. *IFAC Proceedings*, **44**, 3678-3683. https://doi.org/10.3182/20110828-6-IT-1002.01188

[21] Palanca, J., Terrasa, A., Julian, V. and Carrascosa, C. (2020) SPADE 3: Supporting the New Generation of Multi-Agent Systems. *IEEE Access*, **8**, 537-549. https://doi.org/10.1109/ACCESS.2020.3027357

[22] Escriva, M., Palanca, J., Aranda, G., Garcia, A., Julian, V. and Botti, V. (2006) A Jabber-Based Multi-Agent System Platform. 5*th International Joint Conference on Autonomous Agents and Multiagent Systems* (*AAMAS* 2006), Hakodate, 8-12 May 2006, 1282-1284.

[23] Salceda, J.V. (2019) Department of Computer Science. UPC Dario Garcia Gasulla Barcelona Supercomputing Center, Barcelona.

[24] Gnatyshak, D. (2019) Adapting the Smart Python Agent Development Environment for Parallel Computing. Master Thesis, Universitat Politècnica de Catalunya Barcelona Tech, Barcelona, 7-11, 30. https://upcommons.upc.edu/bitstream/handle/2117/133128/139210.pdf

[25] XMPP Is the Open Standard for Messaging and Presence. https://xmpp.org

[26] Palanca, J., Terrasa, A., Julian, V. and Carrascosa, C. (2020) SPADE 3: Supporting the New Generation of Multi-Agent Systems. *IEEE Access*, **8**, 182537-182549. https://doi.org/10.1109/ACCESS.2020.3027357

[27] https://github.com/upb-lea/openmodelica-microgrid-gym

[28] https://github.com/Total-RD/pymgrid

[29] https://github.com/zhang614/MicroGrid/tree/master

[30] Palanca, J., Rincon, J., Julian, V., Carrascosa, C. and Terrasa, A. (2022) Developing IoT Artifacts in a MAS Platform. *Electronics*, **11**, Article No. 655. https://doi.org/10.3390/electronics11040655

[31] Welcome to SPADE's Documentation! https://spade-mas.readthedocs.io/en/latest/

[32] Criado, N., Argente, E., Julian, V. and Botti, V. (2008) Organizational Services for the Spade Agent Platform. *IEEE Latin America Transactions*, **6**, 550-555. https://doi.org/10.1109/TLA.2008.4908189

[33] Khan, M.W., *et al*. (2019) Optimal Energy Management and Control Aspects of Distributed Microgrid Using Multi-Agent Systems. *Sustainable Cities and Society*, **44**, 855-870. https://doi.org/10.1016/j.scs.2018.11.009

[34] Altin, N., Eyimaya, S. and Nasiri, A. (2023) Multi-Agent-Based Controller for Microgrids: An Overview and Case Study. *Energies*, **16**, Article No. 2445. https://doi.org/10.3390/en16052445

[35] Mohammadreza, S. and Mollaie, E. (2022) Energy Management and Harmonic Compensation of Micro-Grids via Multi-Agent Systems Based on Decentralized Control

Architecture. *IET Renewable Power Generation*, **17**, 1267-1285. https://doi.org/10.1049/rpg2.12653

[36] KQML Performatives. https://jmvidal.cse.sc.edu/talks/agentcommunication/kqmlperformatives.html

Appendix

A. Communication

1) Send(): sends a message to another agent.

2) receive(): receives a message from another agent.

3) subscribe(): subscribes to specific types of messages.

4) filter(): filters incoming messages.

5) notify(): notifies other agents of change in its state.

B. Behavior

6) on_start(): initializes and sets up the initial state of the agent.

7) on_receive(): called when an agent receives a message from another agent.

8) on_tick(): called at regular intervals to implement periodic behaviors.

9) on_end(): called when an agent is terminated to clean up resources.

10) add_behavior(): adds a new behavior.

11) remove_behavior(): removes a behavior from an agent.

C. Directory Facilitator

12) register_service(): registers a specific service that it provides.

13) unregister_service(): unregister a previously registered service.

14) search_service(): searches for other agents that provide a specific service.

15) register_agent(): registers an agent in the DF, so other agents can discover it.

16) unregister_agent(): unregister an agent from the directory so that other agents can no longer discover it.

17) search_agent: searches for agents based on their properties.

D. XMPP

18) set_jid(): sets an XMPP ID (JID) of an agent. The JID is a unique identifier to address the agent in XMPP messages.

19) connect(): connects an agent to an XMPP server. The XMPP server acts as a messaging broker, facilitating exchange of messages between agents.

20) send(): sends a message to another agent using XMPP.

21) add_presence_subscriber(): subscribes to the presence updates of another agent. Presence updates provide information about the availability and status of the agent.

22) remove_presence_subscriber(): unsubscribes from presence updates of another agent.

23) add_message_listener(): adds a listener called when an agent receives a new message.