Testbed Application of Energy Agents

Nils Loose, Christian Derksen, Rainer Unland

Testbed Application of Energy Agents

Nils Loose, Christian Derksen, Rainer Unland

DAWIS, University of Duisburg-Essen, Schützenbahn 70, 45127 Essen, Germany

{nils.loose,christian.derksen,rainer.unland}@icb.uni-due.de

Abstract. This work introduces the concept of testbed application of energy agents, which is the intermediate step between testing agents in pure simulation environment and deploying them in real energy distribution systems. In the testbed application case, the energy agent is taken from the simulation environment and deployed to dedicated hardware, where it controls a simulated or real technical system, while still working against a simulated environment. Compared to a pure simulation environment, this application case raises a number of new challenges, mainly resulting from inter-platform agent communication. In this work these challenges are discussed and an implementation handling them is presented and evaluated.

Keywords: Testbed Agents, Energy Agents, Hybrid Energy Systems, Smart Grid

1 Introduction

Energy infrastructures are facing major challenges. With the liberalization of the energy markets in Germany, beginning in the 1990s, long-established monopolies have been broken up with the consequence that numerous new players have entered the stage. Additionally, with the increasing awareness about the environmental impact of fossil fuels, renewable energy sources like wind turbines and solar panels are used more and more, making electrical power supply more volatile, more decentralized and less plannable. Finally, classical producer and consumer roles dissolve, as more and more households mount solar panels on their rooftop and become energy producers at times with high solar radiation, while they still need supply from the grid at night or on cloudy days. Here, the already well-known term “prosumer” became established for this type of energy market participant.

With the time, this resulted in a higher level of complexity for the coordination and control of supply and demand, for which existing electricity grids have originally not been built. It is believed that the idea of a ‘Smart Grid’ represents an important keystone for handling this complexity, i.e. by integrating modern information and communication technology into the energy networks and thus enabling coordination and increasing flexibility within the grid [1, 2].

adfa, p. 1, 2011.

With the concept of an Energy Agent [3] and the Energy Option Model (EOM) [4], an agent-based approach for managing smart grids and their 'smart' participants has been developed (more details on both approaches will be provided in section 3.3). In this paper we will focus on the testbed application of energy agents, where the energy agent is deployed to a dedicated hardware and controlling a (simulated or real) energy conversion system, but working against a simulated environment. This raises a number of challenges, which will be discussed in this work. One main issue in this context is the aspect of communication between the testbed agent and the simulation. For this, the actual implementation will be presented and evaluated for the testbed application case with a simulated energy conversion system.

The remainder of this work is organized as follows: After a review of related work in section 2, the theoretical and technical background for the presented solution will be provided in section 3. In section 4 we will discuss the challenges resulting from running energy agents in a testbed scenario and present our testbed agent implementation, which will be evaluated in section 5. Finally, section 6 provides a conclusion and an outlook to future work.

2 Related Work

In the recent years, multi-agent based approaches for many smart grid related topics have been proposed, realizing technical solutions like virtual power plants [5] or micro grids [6], but also indirect control approaches like demand side management [7] or demand response concepts [8]. Also a number of agent-based decentralized control approaches for smart grids have been proposed, for example DEZENT [9], DeMaPos [10] or PowerMatcher [11]. The focus of these projects is on market based coordination. This can be covered by Energy Agents and EOM too, by providing price information and developing corresponding evaluation strategies. However, our main focus is on the technical aspects of the grid and the involved technical systems. Additionally, all mentioned approaches focus on electricity only, while our solution can handle different energy carriers and also conversion processes between them, which is useful to model for example gas-driven combined heat and power plants (CHPs) or Power-to-X solutions.

The topic of multi-agent based simulations has been thoroughly covered in [12], which provides a discussion of important aspects like environment and time models, as well as numerous examples of applications of MAS-based simulations.

The impact of messaging on the performance of MAS, among others in simulation scenarios, has been investigated by the VSIS group of the University of Hamburg, for example in [13] and [14]. Their work is based on JADEX, a BDI agent framework that also builds on the basic infrastructure provided by JADE. Their results support our findings from [15] that the performance in larger MAS suffers substantially from the use of ACL messaging.

3 Theoretical and Technical Background

This section will shortly give an introduction to agents and multi-agent systems in the first sub-section. This will be followed by a description of the agent framework JADE and the framework and application toolkit Agent.GUI. The last sub-section will provide a short summary about Energy Agents and the Energy Option Model (EOM).

3.1 Agents and Multi-Agent Systems

Literature provides several definitions for the term 'agent' or software agent respectively. A widely accepted one is given by Wooldridge and Jennings:

> *"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives." [16]*

There are different ways to implement autonomous agent behavior. The simplest approach are purely reactive agents, which basically just react on an environment stimulus according to fixed rules [17]. A more complex concept is that of BDI-Agents (Believe, Desire, Intention), which work with an internal knowledge model. This model is updated according to the agent's perception of its environment, and serves as basis for choosing appropriate actions [18]. This approach allows implementing more complex behavior and also learning mechanisms, and thus more sophisticated agents that are known as deliberative agents.

[19] introduces advanced agents that provide autonomy, responsiveness/situatedness, pro-activeness, goal-orientation, smart behavior, social ability, and learning capabilities. This definition is the basis for industrial agents that are defined in [20] as follows:

> *"An industrial agent is an agile and robust software entity that intelligently represents and manages the functionalities and capabilities of an industrial unit. While it reveals the common features of an advanced agent it also has some specifics. It understands and efficiently handles the interface and functionality of (low level) industrial devices. Usually it belongs to an agent-based industrial application system within which it acts and communicates in an efficient, intelligent, collaborative, and goal-oriented way. In principal it is an autonomous and self-sustained unit. Nevertheless, it accepts and follows company guidelines, codes of conduct, general law and relevant directives from higher levels. Moreover, especially in emergency and real-time scenarios its autonomy may be compromised in order to permit fast and efficient reactions."*

While the concept of energy agents also permits the realization of comparatively simple agents the main idea is to provide an environment which is run and controlled by industrial agents in the above sense.

If multiple agents coexist in a shared environment, a Multi-Agent System (MAS) is formed. Here, the social ability of agents is a very important aspect. It is the foundation

for the interaction and cooperation between agents. Agents in a MAS can act cooperatively, but also competitively. To support compatibility and interoperability in heterogeneous MAS, a set of standards has been developed by the Foundation for Intelligent Physical Agents (FIPA), which is part of the IEEE computer society. Important FIPA standards specify a basic architecture for agent platforms, services for agent management, a message format for inter-agent communication realized by the Agent Communication Language (ACL), and a number of relevant interaction protocols. All standards are available on the FIPA website[1].

3.2 JADE and Agent.GUI

Our implementations are based on JADE (Java Agent Development framework)[2], a Java-based software framework for developing FIPA-compliant multi-agent systems. JADE provides basic functionality like agent life cycle management, communication services, interaction protocols etc. Thus, when using JADE, an agent developer can focus on the domain specific problems, while for the basic agent functions classes provided by JADE can be reused or extended.

While a detailed introduction to JADE is beyond the scope of this article, we will provide a brief overview of the JADE architecture; this is important to understand the approach of our testbed agents. A JADE platform is formed by one or more containers, which host the actual agents. Every platform consists at least of the main container, which is the bootstrap point for the platform and hosts some special agents providing FIPA-compliant services. To distribute the agents on several physical nodes, the JADE platform can be extended by starting containers on other computers. Within the same platform, agents from all containers can access the services provided by the main container. Communication between agents is not limited to the platform. ACL messages can also be exchanged with agents hosted on other FIPA-compliant platforms (not necessarily JADE platforms) using a message transfer protocol (MTP). A detailed introduction into JADE can be found in [21].

Based on JADE, the agent-based simulation framework Agent.GUI has been developed [15]. Like JADE, Agent.GUI is an open source software project[3]. It provides a graphical user interface which facilitates the usage of JADE for domain experts without deeper IT knowledge. Beyond this, it offers a wide set of features for developing and executing simulations based on JADE agents. For example, different basic environment models and time models are provided, including a graph-based environment model which is designed for modelling all kinds of networks, especially energy networks. Graphical tools for editing environment models or handling different simulation setups are also included, as well as technical tools like a load-balancing service for running large simulations on distributed JADE platforms.

---

1 www.fipa.org
2 jade.tilab.com
3 http://www.agentgui.org/

An important feature of Agent.GUI in the context of this paper is the so-called Simulation Service. When starting to develop Agent.GUI, the original intention was to handle all interactions between agents and their environment via ACL messages. However, we realized that for bigger simulations exchanging this information quickly leads to a huge messaging load. Since the overhead for sending and receiving messages the agent-based asynchronous way is comparatively inefficient, this has a massive impact on the performance. Therefore, instead of using ACL messaging, we implemented a new JADE service that allows exchanging environment and status information between an environment managing entity (usually a special agent called 'Simulation Manager' that is responsible for managing the simulation environment) and the involved agents in a more direct way. As shown in [15], this significantly increases the simulation performance.

3.3 Energy Agents and the Energy Option Model

The concept of an Energy Agent was originally introduced in [3] as follows:

> *"An Energy Agent is a specialized autonomous software system that represents and economically manages the capacitive abilities of the energy consumption, production, conversion and storing processes for a single technical system and that is embedded and thus part of one or more domain specific energy-networks, capable to communicate therein and with external stakeholders."*

Two drawbacks have led to the development of this approach: Most control approaches for smart grids are based on proprietary solutions and, thus are incompatible to each other. The Energy Agent aims for establishing a unified approach, enabling interaction between different smart grid solutions. Secondly, the majority of published smart grid solutions focus on electricity only. By building on the foundations of thermodynamics, the Energy Agent approach can handle all kinds of energy flows, including conversion processes between different energy carriers.

In [3], a standardized development cycle for Energy Agents is proposed, which builds upon approaches like Hardware-in-the-loop simulations or Rapid Control Prototyping. The main idea of this development cycle is to move the agent through different phases while gradually changing the environment. It consists of the following steps:

1. Specification and modelling: The desired functionality and interactions of the system have to be described using a suitable modelling technique.
2. Implementation: The previously described software system has to be implemented in an appropriate programming language.
3. Simulation: The implemented software artefact is first tested in a simulation environment providing the same information sources and interaction possibilities as the real system.
4. Testbed application: The software artefact is deployed on dedicated hardware to be tested under field conditions. While the controlled technical system can be either simulated or real hardware, the environment is still a simulated one.

5. Deployment in a real system: After passing all tests in simulated environments, the agent can finally be tested in a real field environment, controlling a real technical system and interacting with a real network infrastructure.

In this work, we focus on the fourth step: the testbed application. After being thoroughly tested in a pure simulation environment in step three, the agent is deployed to a dedicated hardware and controlling a (simulated or real) technical system, but still working within a simulated environment. The challenges in this state are the seamless integration of actual real-world entities into the simulation environment, especially also the communication from outside to the simulation and, if switching to real hardware, the implementation of an I/O-behavior to interact with it.

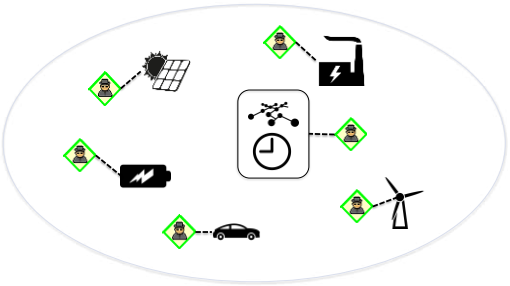
After passing this test, the Energy Agent can be deployed in the real world. Here it is not supposed to completely replace the existing real time control of the actual technical systems but just to supplement it. While the system is still controlled by its system-specific controller, the Energy Agent evaluates and manages the operational flexibility of the system in the smart grid context and gives instructions and suggestions to the controller how to operate the underlying system. To be able to do so, the Energy Agent needs comprehensive knowledge about the specific technical system. This is provided by the Energy Option Model (EOM), which can be seen as the internal knowledge model of the Energy Agent. Thus, the Energy Agent realizes a type of BDI-agent.

The EOM was originally introduced in [4]. As it is not in the focus of this work, we will just give a short overview here. The core of an EOM model is the description of the different operational states of a technical system, including the resulting energy flows at the interfaces of the system. As the possible transitions between states and their minimum and maximum duration are also specified, this results in a comprehensive description of the operational flexibility of the system. Based on an evaluation of this flexibility, appropriate execution schedules for the technical system can be generated. Information about energy costs or losses are provided and can be used to optimize the execution schedule for minimal costs or maximal efficiency. Examples for EOM-based evaluations and optimizations are given in [22] and [23].

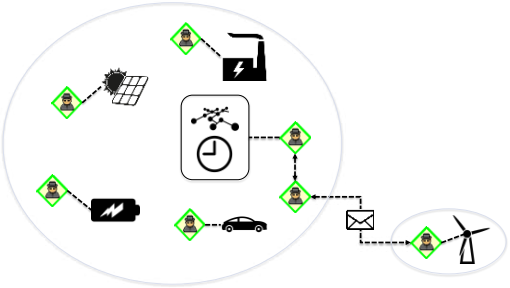
4 Challenges and Implementation

In a testbed application the energy agent is deployed on a dedicated hardware to be tested under field conditions, while still running within a simulated environment. The technical system controlled by the agent can be either a simulated one or real hardware, which leads to a further division into two sub-steps, which we refer to as “testbed simulated” and “testbed real”. Usually, an Energy Agent will first be tested in a pure simulation environment, then in a testbed with a simulated and finally in a testbed with a real technical system, before being deployed to the real world. **Fig. 1** illustrates those four application cases.

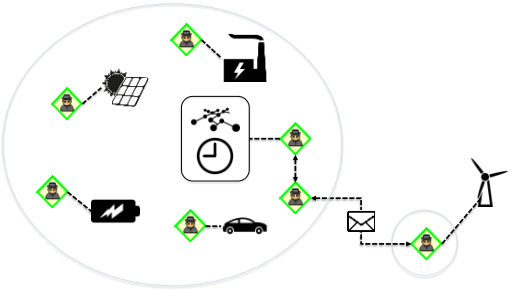
a) Simulation



b1) Testbed Simulation



b2) Testbed Real



c) Real System

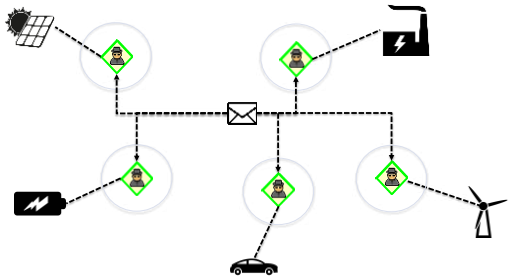


Fig. 1. Application cases for Energy Agents

As discussed in Section 3.2, there are two alternatives to have interacting JADE agents running on different physical nodes. First, a single JADE platform can be extended by starting containers on other computers. This is completely transparent for the agents, as services provided by the platform are usually accessible from all containers, and ACL messages can be sent by using the receiver's local ID within a platform.

However, this rather close coupling is more suitable for application cases like running a large simulation on several physical nodes in the same data center. A deployed Energy Agent in the field will usually be running on a computer mounted close to the energy conversion system it controls, possibly with a rather poor network connection via powerline or radio. Therefore, it seems more appropriate to choose a looser coupling and run the deployed Energy Agent in a separate JADE platform. As mentioned in section 3.2, inter-platform communication is possible in this setup by sending ACL messages over a Message Transfer Protocol (MTP).

In **Fig. 1**, every circle symbolizes a separate JADE platform. The agent in the bottom right, controlling the wind turbine, is deployed as a testbed agent in **Fig. 1** b1) and b2).

The use of separate platforms raises a number of challenges for the testbed application case, most of them related to the aforementioned inter-platform communication. Those challenges, and our solutions for them, will be discussed in the following subsections.

4.1 Communication with the Simulation

As discussed in section 3.2, in order to reduce overhead and improve the performance of a simulation, in Agent.GUI simulation status updates are not communicated via ACL messages, but with the help of a newly introduced JADE service, the simulation service. For the testbed application case this option is no longer available, as JADE services are only accessible from within a platform, but not from remote platforms.

To solve this problem, we introduced a new proxy agent, which acts as a mediator between the testbed agent and the simulation service. This is visualized in **Fig. 1** b1) and b2). An additional agent appears between the simulation manager agent and the testbed agent, which runs in a separate JADE platform. Towards the simulation service, this proxy agent acts just like a regular energy agent, sending and receiving status updates and notifications via the simulation service. Internally, it acts as a converter between simulation service and ACL messages. Updates coming from the simulation service are encapsulated in an ACL message and sent to the testbed agent on the remote platform, and vice versa for messages coming from the testbed agent. Because of that the simulation is completely transparent when it comes to the question whether an agent is running in the simulation or the testbed mode.

The contents of the exchanged messages can be a simple status update, but also larger and more complex content like complete environment models. There are different ways to use objects as content for ACL messages: Using formal ontologies and corresponding codecs, simple text strings or normal JAVA serialization. The benefits and drawbacks of each method are discussed in [14]. For the communication between our proxy and testbed agents we chose the JAVA serialization, because according to

this reference it is the most performant method available in an off-the-shelf JADE installation without further extensions. Drawbacks of this method are the lack of a well-defined semantics and the limitation to agents implemented in JAVA. For our testbed application case these issues can be neglected, as all involved agents are written in JAVA and developed by ourselves. In the field however, interaction with agents developed by others, and maybe even not in JAVA, might become necessary. For such a case a well-defined ontology might be the better choice.

4.2 Connecting proxy and testbed agent

To establish the connection between the testbed and the proxy agent, we introduced a Central Executive Agent (CEA). The CEA is always started at the simulation startup if at least one agent is configured to run in testbed mode. Its agent identifier (AID) is given to the testbed agent at deployment time. The AID consists of the globally unique ID of the CEA – for example *CentralExecutiveAgent@SimulationPlatform* – and the MTP address of the JADE platform, which specifies the host name or IP address and the network port for sending messages to this platform. When being started, both, the proxy and the testbed agent send registration requests to the CEA. After receiving both requests the CEA sends the AIDs of both agents to their counterpart. From this point onward a direct communication between the proxy and testbed agent is possible.

4.3 Security

Another important issue connected to inter platform communication is security, which is essential in critical infrastructures like energy supply grids. While it might be negligible for the testbed application case, it will definitely arise when deploying agents for applications in the field. Thus, we already considered it when developing our deployment process. Security in this context comprises two important aspects: Secure encryption of message contents, keeping them confidential and free from manipulations, and secure authentication of the communication partner.

As discussed before, for exchanging ACL messages between different platforms, JADE uses a message transfer protocol (MTP). By default, an MTP based on the common HTTP protocol is used, which means ACL messages are sent unencrypted and without authentication. But alternatively, a more secure MTP based on HTTPS can be chosen.

HTTPS uses an asymmetric key concept for message encryption. In this concept, every participant has a private as well as a public key. While the first one is kept secret, the second is exchanged with the communication partners. In our case, the participants are the different JADE platforms, as the agents do not communicate over the MTP directly, but use the platform's messaging service to do so. If sending a message to an agent on another platform via HTTPS, the content is encrypted by the sender using the public key of the target platform. To decrypt it, the corresponding private key is required, which means the message content is not accessible for unauthorized third parties.

The generation of the key pairs can easily be done using standard JAVA libraries. More complex is the question of how to exchange public keys. For our implementation we assume that a beforehand unknown number of remote agents – each running in its own remote JADE platform – might join the MAS. Therefore, it is not feasible to make all remote agent's public keys known to the central JADE platform in advance. While this might actually be possible in the testbed application case, it is not for the real field application, where new agents might join the system at runtime.

To solve this problem, we make use of the CEA again. At deployment time, the public key of the platform hosting the CEA is given to the testbed agents. Now they are able to send encrypted messages to the CEA. When sending its registration request, the agent includes its public key of the platform it is running on, which enables all agents hosted on the platform of the CEA to send encrypted messages to the testbed agent.

This solution enables encrypted communication also with agent platforms previously not known to the CEA. However, a secure authentication of the communication partner is not guaranteed, as anyone, also unauthorized and maybe malevolent third parties, could send their public key to the CEA if they know the CEA's AID. Therefore, before going into the field, a method for a secure authentication of the communication partner must be found.

4.4 Communication latencies

Another important issue for the testbed application case is the question of time synchronization. Agent.GUI offers two different time models for simulations, a discrete and a continuous one. In the discrete case, simulation time proceeds in fixed steps, for example ten seconds of simulation time for every simulation step. Before finishing one simulation step and proceeding to the next one, the simulation manager waits for feedback from all involved agents. This makes the integration of testbed agents easy, as the simulation manager will also wait for their feedback before proceeding.

The continuous case is more complex, as the simulation proceeds in (optionally accelerated) real time. This raises two problems. First, the system time of all involved JADE platforms has to be synchronized. As this problem occurs in many network-based applications, there is a standardized solution for it. There are servers providing time information using the Network Time Protocol (NTP), so platforms can be synchronized by obtaining their system time from the same NTP server.

The second problem is more difficult to solve. Sending status updates via ACL leads to latencies, caused by the actual message transfer process, the conversion performed by the proxy agent and, if using HTTPS, by the encryption and decryption. This means if an agent inside the simulation platform and a testbed agent on a remote platform perform an action at the same time, the action of the testbed agent will be registered by the simulation manager slightly later due to the latency. This problem is not solved yet. Up to now we only work with discrete simulations when using testbed agents. A first idea is to estimate the latency, for example by calculating the average latency for the last n messages, and then shift the system time of the testbed agent platform accordingly to compensate the latency. But this has neither be implemented nor tested yet. Further investigation on this topic will be necessary before switching to continuous simulation.

5 Evaluation

To evaluate the performance of our testbed agent implementation we executed several simulation runs with different numbers of testbed agents involved. Agent.GUI provides a load monitoring tool, which keeps track of the system load while executing a simulation, and also the number of simulation cycles per second as a measure for the performance of the simulation. A discrete time model was chosen and the simulation interval was set to 200 milliseconds. Thus, the target value is five cycles per second.

When using a discrete time model, a single simulation cycle is structured as follows: First, the simulation manager sends a time signal to all involved agents. Based on this time signal and their knowledge about the current network state, the agents determine their action for the current step and send their resulting system state to the simulation manager. The manager waits for the feedback from all agents, calculates the new network state based on this information, and then starts the next cycle by sending a new time signal. As mentioned before, while in a pure simulation environment all involved communication is done using the simulation service. For testbed agents it has to be done via ACL messages using a proxy as mediator. By measuring simulation cycle times for scenarios including testbed agents and comparing them with pure simulation scenarios, we can access how this solution affects the simulation performance.

The simulation scenario used for these tests is based on the model of an electric distribution grid located in a small German city. The network model is shown in **Fig. 2**. All involved agents are generic prosumer agents that follow a predefined schedule without performing any evaluation or optimization. Thus, we can be sure that our measurements are not influenced by algorithms executed by the agents.

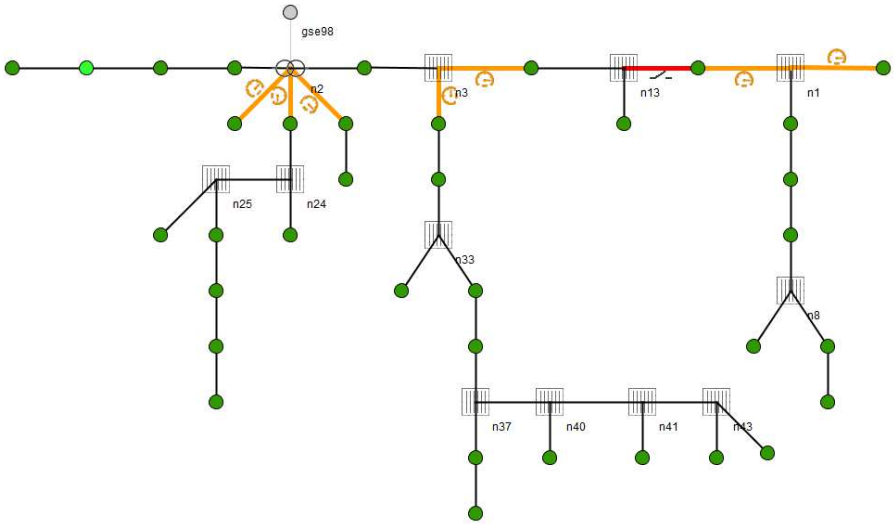


Fig. 2. The network model

For our measurements we executed a simulation based on this scenario and varied the number of testbed agents between zero and five, measured the number of simulation cycles per second and calculated the average duration of a simulation cycle, based on five runs with the same setup. This procedure has been executed twice, with inter-platform communication based on the HTTP as well as the HTTPS protocol. The results are visualized in **Fig. 3**.

For a pure simulation scenario with zero testbed agents, an average cycle time of 200.4ms has been measured. As in this scenario no inter-platform communication takes place, the message transfer protocol does not affect the result. It can be clearly seen from the figure that with an increasing number of testbed agents the simulation cycle time also increases. Thus, there is a negative effect of the testbed agents on the simulation performance. Not surprisingly, it is stronger for the HTTPS-based communication due to the message encryption and decryption. However, with an increase between one and two milliseconds for an additional testbed agent, the effect is rather moderate. This might result from the fact that after the initial distribution of the whole environment model before the actual simulation start, only rather small time signals and status updates are exchanged during the actual simulation steps. With larger contents exchanged the negative performance effect of ACL messaging would probably be much stronger.

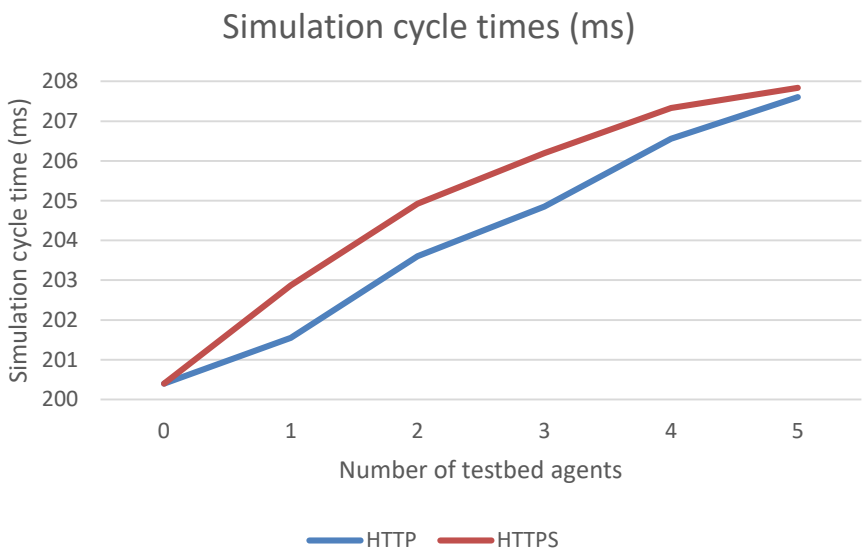


Fig. 3. Average simulation cycle times for different numbers of testbed agents

In our experiments, the testbed agents have been executed in separate JADE platforms, but on the same physical node. To get an impression whether the lack of actual network communication significantly affects the results, we executed another set of runs with one testbed agent running on a separate system in our local network. Due to technical problems this experiment could only be executed with HTTP communication.

While the average cycle time for a single testbed agent running on the same machine was 201.5ms with HTTP and 202.9ms with HTTPS communication, it was 202.5ms on a separate machine with HTTP. So the network latency seems to have an effect on the performance. However, it is weaker than that of the HTTPS communication.

It has to be pointed out that this measurement was executed within the local network of our university. In a real field scenario, the available network infrastructure may be less performant and reliable. A test under more realistic conditions would be desirable. However, as this highly depends on the conditions found in the field, it could not be done in the context of this work.

6 Conclusion and outlook

The concept of testbed application of Energy Agents has been introduced and the related challenges, mainly resulting from the necessary inter-platform communication, have been discussed. An implementation of testbed agents has been presented, and evaluated for testbed agents controlling simulated energy conversion systems. The evaluation results show that the additional communication has a negative effect on the simulation performance. However, this effect is rather moderate, probably because within a simulation cycle only small messages like time signals and status updates are exchanged. It must be noted, however, that the effect of network latencies was neglected during this evaluation. Thus, for energy agents deployed in real energy distribution systems, probably with limited network connectivity, the negative impact will most likely be stronger.

In this work, only testbed agents controlling simulated technical systems have been used. The next step towards real field applications will be testbed agents controlling real hardware, which requires developing Input/Output-behaviors for the Energy Agent that are capable of interacting with real sensors and effectors. In this context, it is also necessary to switch from discrete to continuous simulation time. While this has already successfully been done for pure simulations, it might rise new issues for the testbed agent application case, especially concerning time synchronization.

When a testbed application with real hardware is successfully developed, the next step is testing the energy agent approach in the context of real energy distribution systems. For this application case new challenges resulting from communication over probably limited network infrastructure and also from high security requirements will have to be solved. These are our next research challenges.

References

1. Blumsack, S., Fernandez, A.: Ready or not, here comes the smart grid! 7th Biennial International Workshop “Advances in Energy Studies” 37, 61–68 (2012)
2. Ramchurn, S.D., Vytelingum, P., Rogers, A., Jennings, N.R.: Putting the 'smarts' into the smart grid: a grand challenge for artificial intelligence. Communications of the ACM 55, 86–97 (2012)

3. Derksen, C., Linnenberg, T., Unland, R., Fay, A.: Unified Energy Agents as a Base for the Systematic Development of Future Energy Grids. In:, pp. 236–249. Springer Berlin Heidelberg (2013)
4. Derksen, C., Linnenberg, T., Unland, R., Fay, A.: Structure and classification of unified energy agents as a base for the systematic development of future energy grids. Engineering Applications of Artificial Intelligence 41, 310–324 (2015)
5. Tröschel, M., Appelrath, H.-J.: Towards Reactive Scheduling for Large-Scale Virtual Power Plants. In:, pp. 141–152. Springer, Berlin, Heidelberg (2009)
6. Kantamneni, A., Brown, L.E., Parker, G., Weaver, W.W.: Survey of multi-agent systems for microgrid control. Engineering Applications of Artificial Intelligence 45, 192–203 (2015)
7. Logenthiran, T., Srinivasan, D., Shun, T.Z.: Multi-Agent System for Demand Side Management in smart grid. In: 2011 IEEE Ninth International Conference on Power Electronics and Drive Systems, pp. 424–429 (2011)
8. Li, H.A., Nair, N.K.C.: Multi-agent systems and demand response: A systematic review. In: 2015 Australasian Universities Power Engineering Conference (AUPEC), pp. 1–6 (2015)
9. Lehnhoff, S.: Dezentrales vernetztes Energiemanagement. Vieweg+Teubner, Wiesbaden (2010)
10. Linnenberg, T., Wior, I., Schreiber, S., Fay, A.: A market-based multi-agent-system for decentralized power and grid control. In: ETFA2011, pp. 1–8 (2011)
11. Kok, K.: The PowerMatcher: Smart Coordination for the Smart Electricity Grid
12. Uhrmacher, A., Weyns, D. (eds.): Multi-agent systems. Simulation and applications. CRC Press/Taylor & Francis, Boca Raton (2009)
13. Braubach, L., Pokahr, A.: Method Calls Not Considered Harmful for Agent Interactions. International Transactions on Systems Science and Applications (ITSSA) 1/2, 51–69 (2011)
14. Jander, K., Lamersdorf, W.: Compact and Efficient Agent Messaging. In: Mehdi Dastani, Jomi F. Hübner, Brian Logan (ed.) Programming Multi-Agent Systems, pp. 108–122. Springer, Berlin, Heidelberg (2012)
15. C. Derksen, C. Branki, R. Unland: Agent. GUI: A multi-agent based simulation framework. In: 2011 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 623–630 (2011)
16. Wooldridge, M.: An introduction to multiagent systems. Wiley, Chichester (2009)
17. Brooks, R.A.: Intelligence without representation. Artificial intelligence 47, 139–159 (1991)
18. Rao, A.S., Georgeff, M.P., others: BDI agents: From theory to practice. In: ICMAS, 95, pp. 312–319 (1995)
19. Unland, R.: Software Agent Systems. In: Leitão, P., Karnouskos, S. (eds.) Industrial Agents: Emerging Applications of Software Agents in Industry, pp. 3–22. Elsevier Science (2015)
20. Unland, R.: Industrial Agents. In: Leitão, P., Karnouskos, S. (eds.) Industrial Agents: Emerging Applications of Software Agents in Industry, pp. 23–44. Elsevier Science (2015)

21. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing multi-agent systems with JADE. John Wiley, Chichester, England, Hoboken, NJ (2007)
22. Derksen, C., Unland, R.: The EOM: An Adaptive Energy Option, State and Assessment Model for Open Hybrid Energy Systems. In: 2016 Federated Conference on Computer Science and Information Systems, pp. 1507–1515. IEEE (2016)
23. Loose, N., Nurdin, Y., Ghorbani, S., Derksen, C., Unland, R.: Evaluation of Aggregated Systems in Smart Grids: An Example Use-Case for the Energy Option Model. In: Bajo, J., Escalona, M.J., Giroux, S., Hoffa-Dąbrowska, P., Julián, V., Novais, P., Sánchez-Pi, N., Unland, R., Azambuja-Silveira, R. (eds.) Highlights of Practical Applications of Scalable Multi-Agent Systems. The PAAMS Collection: International Workshops of PAAMS 2016, Sevilla, Spain, June 1-3, 2016. Proceedings, pp. 369–380. Springer International Publishing, Cham (2016)