Paving the Way for Reinforcement Learning in Smart Grid Co-Simulations*

Dominik Vereno[0000-0002-7930-6744], Jonas Harb[0000-0001-8018-4732], and Christian Neureiter[0000-0001-7509-7597]

Josef Ressel Centre for Dependable System-of-Systems Engineering, 5412 Puch/Salzburg, Austria
{firstname.lastname}@fh-salzburg.ac.at

Abstract. This paper identifies and addresses a gap in research on using reinforcement learning (RL) in co-simulation. Co-simulation is an effective simulation paradigm for systems of systems such as smart grids. It relies on combining heterogeneous simulators into a coupled simulation. RL is a promising machine-learning tool for complex grid applications—for instance, demand-side management. However, existing literature does not specifically address challenges of integrating RL with a co-simulation environment. Therefore, we focus on two challenges: how an RL agent is best integrated into a co-simulation architecturally, and to what extent typical RL frameworks are interoperable with orchestrated co-simulation tools. First, we introduce, categorize, and evaluate four approaches of architecturally integrating RL into co-simulation. Additionally, we provide guidance on selecting an appropriate approach. Second, we conduct a case study where we use and incorporate a framework-based RL agent into a co-simulation framework for a simple demand-side management scenario; we identify the need to change the control flow traditionally used in RL frameworks to achieve interoperability. In conclusion, our work is a basis for future academic or industrial applications of RL in co-simulation. Our architectural and framework-specific advice facilitates the implementation of RL in smart-grid co-simulations.

Keywords: Model-based systems engineering · Power-grid simulation · Demand-side management · Software architecture · Artificial intelligence.

1 Introduction

Recently, traditional power grids are evolving into so-called smart grids. They combine electrical infrastructure with information and communications technology to enable intelligent monitoring and control [10]. Contemporary power-systems engineering faces many challenges: The increasing share of hard-to-predict and volatile renewable-energy generation forces grids to become more

* The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development and the Christian Doppler Research Association as well as the Federal State of Salzburg is gratefully acknowledged.

flexible. In addition, the rising number of electric vehicles (EVs) is straining electrical infrastructure with high peaks in consumption [7]. This stress may be alleviated by intelligently controlling the charging process as part of demand-side–management strategies [6]. Demand-side management includes measures for improving the power-grid operation from consumer side [24].

Data-based methods—such as machine learning—have proven to be an effective tool for such tasks [2]. A promising branch of machine learning that has been gaining traction recently is reinforcement learning (RL) where an agent learns optimal behavior by trial and error. RL has the potential of finding new state-of-the-art algorithms that surpass expert knowledge [23]. The machine-learning paradigm shows great application potential for many smart-grid tasks [37]. Training an RL agent is usually done in a simulation. For RL training in smart grids, we need a realistic simulation that is both extensive and detailed. However, comprehensive simulations of smart grids, which constitute *systems of systems* according to the criteria defined by DeLaurentis [8], present particular challenges. This is because systems of systems span across multiple domains and have heterogeneous and independent subsystems. Co-simulation is capable of addressing these challenges by coordinating multiple heterogeneous simulators [22]. It is the coordinated execution of models with different representations and runtime environments [28]. To obtain a realistic co-simulation environment for RL, models that contain domain knowledge are required. Dealing with the heterogeneity of modeling paradigms and notations of cyber-physical systems of systems—such as modern power grids—is challenging [12]. Model-based systems engineering (MBSE) is an established discipline that deals with the modeling of such systems [20].

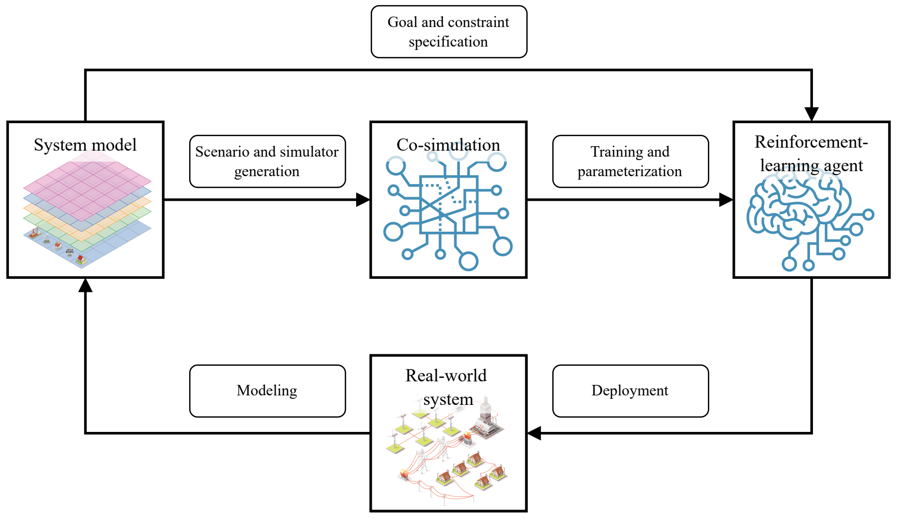

Goal and constraint specification
System model
Scenario and simulator generation
Co-simulation
Training and parameterization
Reinforcement-learning agent
Modeling
Real-world system
Deployment

Fig. 1: Overview of discipline artifacts and their interactions

It follows, that to exploit the potential of RL for smart-grid tasks such as demand-side management, four research disciplines must be considered. Figure 1 illustrates the four disciplines' central artifacts as well as their interactions. *Power-systems engineering*, represented by the real-world system, is needed to gain domain knowledge. This knowledge must be captured in system models using methods of *model-based systems engineering*. The models then facilitate the creation of a *co-simulation* and implementation of *reinforcement learning* algorithms. The co-simulation serves as the environment for developing and training the RL agent. This agent may then be deployed to the real-world system. To the best of our knowledge, existing research does not address how RL agents can be trained and validated in co-simulations. In particular, no research discusses architectural integration and framework-specific interoperability concerns between the respective tools.

This paper contributes to the state of the art by addressing the challenges and obstacles of integrating an RL agent into a smart-grid co-simulation. The fundamental requirements for RL when it comes to interfacing with a simulation are quite basic and should be unproblematic in principle. However, we have identified two areas in need of research: First, we identify a lack of research about how to integrate an RL agent into a co-simulation architecturally. Thus, we identify architectural approaches, classify them, and give guidance for applying them. Second, there is no literature demonstrating the practical challenges of using RL in a co-simulation regarding their respective software tools. Consequently, we conduct case study–based experiments using a co-simulation and an RL framework to uncover integration obstacles and issues. We describe one such issue concerning the incompatibility of the typical control flow of RL frameworks and the typical control flow of orchestrated co-simulation. Consequently, we provide a solution for the particular tools used in the study. Our work intends to facilitate future industrial and academic applications of RL in co-simulation by providing guidance on overcoming architectural and tool-based challenges.

2 Background and Related Work

In the introduction, we highlight four important disciplines for using RL in co-simulations. This sections now provides important background information on RL, co-simulation, and MBSE. Furthermore, it elaborates on related work regarding the intersections between these disciplines and the fourth discipline: power-systems engineering.

RL is a class of machine learning algorithms characterized by learning through trial and error. In RL, the goal is for an *agent* to take favorable sequences of *actions* to affect the state of its *environment*. The agent decides on an action based on data describing the environment's state, so-called *observations*. Additionally, the agent is provided with a real-valued *reward* that quantifies how favorable an action is, given a certain state. This interaction between agent and environment is depicted in Figure 2. We refer interested readers to Sutton and Barto [32], who provide a comprehensive overview of RL.

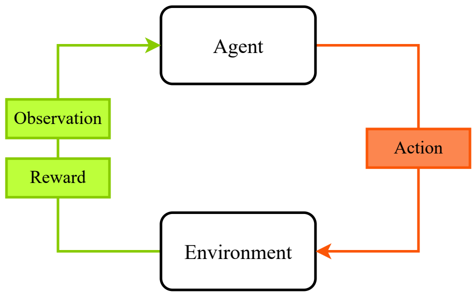


Fig. 2: Interaction between an RL agent and its environment

Since smart-grid systems are crticial infrastructure, they require a high degree of dependability. Therefore, real-world exploration of an untrained RL agent via trial and error is too risky to be feasible. Consequently, we need a simulated environment to train the agent.

Table 1: System-of-systems traits, their simulation challenges, and advantages of co-simulation for addressing the challenges

System-of-systems trait	**Simulation challenge**	**Advantage of co-simulation**
Heterogeneity	Subsystems are modeled with different tools that support their respective algorithmic and computational needs.	Coupling is done on solver level, removing the need for model consolidation.
Operational and managerial independence	Simulators of various organizations are combined. Some underlying models may be confidential.	Co-simulation facilitates black-box integration of simulators.
Trans-domain	Experts of specific domains do not have in-depth knowledge of the entire trans-domain system of systems.	Domain experts can work on an appropriate simulation of the subsystem without regard for the entire coupled simulation.

Simulation is well-established in the grid domain; in fact, Palensky et al. [25] describe it as "fundamental in power engineering". Simulating smart grids is a demanding task; in addition to being complex systems, smart grids can be classified as systems of systems [18]. Thus, they exhibit all their defining traits introduced by Sage and Cuppan [27] and expanded by DeLaurentis [8]. We deem three of them to be particularly problematic for simulation: heterogeneity of subsystems, their operational and managerial independence, and that they span multiple domains. These traits and some of the simulation challenges they exacerbate are listed in Table 1. An effective simulation paradigm for dealing with theses challenges is *co-simulation*. With co-simulation, a coupled system is simulated by coordinating stand-alone simulations of the constituent systems [13]. A simulation can be regarded as *co*-simulation if the coupled simulations differ regarding the used simulation tool, the solver algorithm, or the step size [15]. Co-simulation allows subsystems to be modeled and simulated in an environment native to them [21], [25]. Independently-developed simulations can be combined into large-scale scenarios [26]. Therefore, "modeling can be done on the subsystem level without having the coupled problem in mind" [28]. In Table 1, we explain how co-simulation can alleviate smart-grid simulation problems. Furthermore, simulators can either be coupled with bilateral interfaces or by using an orchestrating framework that handles data exchange between simulators and synchronizes their execution [31]. With bilateral interfaces, the data exchange and synchronization becomes increasingly complex. According to Nguyen et al. [21] framework-based, *orchestrated* co-simulation simplifies the simulation architecture. For more information on co-simulation in general, Schweiger et al. [30] give an empirical insight into the usage and prevalence of co-simulation while Gomes et al. [13] and Hafner and Popper [15] have each surveyed the field extensively.

A comprehensive co-simulation requires specifying the behavior of the subsystems and the simulated scenario, i.e. the entities and their connections to each other [30]. Such specifications can be supplied by models used in power-grid engineering—for example, the mathematical description of power flow, a model containing electric lines, buses, and transformers, or the architecture description of control software. However, MBSE provides a comprehensive methodology for managing these models "beginning in the conceptual design phase and continuing throughout development and later life cycle phases" [36]. It is inherently well suited to dealing with the complexity of smart grids [20]. In the context of this paper, we use the term *model* as an artifact containing a purposeful abstraction of a system. Creating detailed models that are a suitable basis for co-simulation requires modeling know-how as well as smart-grid domain expertise.

For this researcher endeavor, we must examine how the discussed disciplines—co-simulation, RL, and MBSE—interface with each other and with the power-systems domain. For RL, various research projects have demonstrated the efficacy of the paradigm for smart-grid applications. According to Zhang et al. [37], RL was used in numerous grid-related areas, such as cyber-security defense, load forecasting, anomaly detection, and demand-side management; Vázquez-Canteli and Nagy [35] survey the available literature on RL in demand-side management specifically. Cabot et al. [5] state that system models could also be used to support AI methods, although they identify a lack of research on that topic. Binder et al. [3] show that system models can be used as a basis for (semi-)automatic generation of co-simulation simulators. Regarding the simulation of smart grids, Palensky et al. [25] and Steinbrink et. al. [31] outline the open challenges of smart-grid co-simulation while highlighting its necessity. Even though the co-simulation paradigm is beneficial for many smart-grid simulations [25], to the best of our knowledge, its use with RL is very sparsely discussed in literature. Some examples that make use of co-simulation to train RL agents are Fischer et al. [11] and Veith et al. [33]. They present a specialized form of RL called Adversarial Resilience Learning and train it using co-simulation. Their work serves as an implicit demonstration that RL can be used in a co-simulation context. Although Veith et al. [33] go into more detail about their specific implementation, neither discuss the general architectual and tool-interoperability considerations explicitly.

3 Research Approach and Methodology

The overarching goal of this study is to explore and examine the challenges of integrating an RL agent into a smart-grid co-simulation. Achieving this goal facilitates future academic and industrial applications of RL training in co-simulation environments—for example, to develop an intelligent charging algorithm for demand-side management by training an RL agent. This goal includes an in-depth analysis of the approaches for inserting the agent into an orchestrated co-simulation architecture. Furthermore, a closer look at the compatibility and interoperability of RL and co-simulation tools is necessary.

Concerning architectural integration, we started with a descriptive approach of identifying architecture candidates and devising a classification scheme. First, we determined components and the required information flows for developing viable architecture candidates. Following that, we clustered the candidates and removed redundant ones. Thus, we arrived at a set of four candidates. Then, an extensive analysis of the candidates was conducted, which led to a categorization scheme that may be used as an aid for architectural decision making. In Section 4, a detailed discussion of the architectural integration can be found.

To address the goal of integrating RL and co-simulation tools (see Section 5) we first established an overview of widely used tools in both disciplines. Then, we selected one co-simulation and one RL framework that we have deemed to be representative. Our further examinations strongly hinge on a simple fictitious case-study scenario: conducting demand-side management for a smart EV-charging scenario using RL. Crozier et al. [6] describe smart charging as the "coordinated scheduling of the charging time and power of EVs". We implement a simple distribution-grid co-simulation in which we integrate an RL agent which controls the charging process. The agent's task is to avoid peaks in demand while keeping average charge rates as high as possible. The case study serves two purposes in the context of our research: On the one hand, we want to uncover framework integration issues and obstacles. Ideally, the findings should identify issues that do not just occur with the specific pair of frameworks, but issues that relate to how co-simulation tools and RL tools are generally structured. On the other hand, we use the case study to evaluate our architecture candidates. This evaluation allows us to better assess the characteristics of each candidate to give more comprehensive and accurate guidance on using them.

4 Architectural Integration of Reinforcement Learning Agents With Co-Simulation

In this section, we discuss architectural considerations and options for integrating RL agents in a co-simulation context. We identify and classify four flexible candidates for architectural integration and contrast them with a more naïve alternative. Finally, we provide guidance on choosing an appropriate candidate depending on different requirements and situations.

4.1 Introducing Terms and Notation

To facilitate discussion about the architecture candidates and their classification, we define their constituent components:

- The **RL agent** is the component that encapsulates the RL algorithm as well as any pre- and postprocessing of input and output data respectively. The reward may be passed together with the observation or is alternatively calculated internally. It should be noted that the literature is not clear on what exactly the boundaries of the agent are; there is no unanimous opinion whether aspects like pre- and postprocessing are part of the RL agent.
- In this context, a **simulation model** is a formal description of the simulated system's approximate behavior. It can be agnostic to its use in a co-simulation framework and does not have to adhere to a particular modeling paradigm or language. To emphasize that the model in question is the one controlled by the RL agent, we use the term *controlled (simulation) model*.
- The **(co-simulation) framework interface** enables data exchange between a simulation model and the orchestrating framework and allows the framework to control the simulation model (e.g. via a step function).
- A **simulator** contains one or more simulation models as well as a framework interface. It further comprises a simulation kernel on which the model is run [9] to make it an "independently executable piece of software that implements a simulation model" [31]. In a *co*-simulation, multiple simulators are coupled and coordinated.

4.2 Naïve Architecture Candidate

The most basic way of implementing an architecture for RL in co-simulation is direct integration of all RL logic into the simulation-model component. The RL agent receives observation data directly from the simulation model and passes the calculated action right back to it. This does not require any explicitly defined interface. Figure 3 depicts this architecture candidate.

This approach is intuitive; putting the system and its controlling algorithm into the same component makes sense, because they represent aspects of the same entity. This also does not require a lot of overhead and may be an attractive starting point for initial exploration. However, the tight coupling between the two components may lead to several issues. It may be difficult for different teams

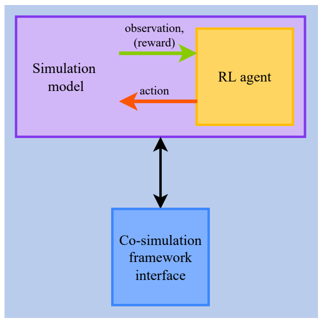


Fig. 3: Naïve approach for architectural integration

to work on the simulation model and RL agent independently. Furthermore, an existing simulation model may have to be changed significantly. Also, the simulation model is not reusable for other simulation scenarios without RL.

4.3 Alternative Architecture Candidates

To address the issues with the naïve approach, we introduce four architecture candidates for RL integration into co-simulation. We classify them based on two independent categories, and give suggestions on when to use each approach.

Classification Matrix and Architecture Candidates We consider two independent binary categories. One the one hand, we differentiate if the RL agent receives its own framework interface. This creates an independent RL simulator in the co-simulation that can be individually coupled to other simulators. On the other hand, the agent may either be limited to receiving observation and reward only from the simulation model or, alternatively, be open to receiving this information from other sources. If not limited, the RL agent can be supplied with data not available through the controlled simulation model. With these distinctions, we can create a classification matrix, as depicted in Figure 4. Each square in the matrix contains one architecture candidate:

- **Candidate A**: The interface component only exchanges data with the simulation model, while the RL agent only communicates with the simulation model. Comparing it to the naïve approach, the simulation-model component is loosely coupled with the RL agent. There must be a clearly defined interface for exchanging observation and action. This candidate does not require the framework interface to be altered due to the addition of RL.
- **Candidate B**: In contrast to Candidate A, the RL agent does not receive observation data from the simulation model. Instead, it receives the data directly from the framework interface. In this case, the interface must be changed to accommodate this data flow. However, the data flows to the agent can be defined and adapted more freely. For example, the simulation

model might only receive voltage data from other simulators whereas the RL agent is additionally supplied with temperature data; in this case the simulation model does not need to be altered to support the additional data flow.

- **Candidate C**: The RL agent receives its own framework interface, making it an independent simulator that must be coupled to the co-simulation. Similar to Candidate A, the RL agent only exchanges data with the simulation model. However, their communication is handled by the orchestrating framework and therefore passes through their respective framework interfaces.
- **Candidate D**: In contrast to Candidate C, the RL-agent simulator is now connected to an arbitrary number of simulators in addition to the simulation-model simulator. Optionally, the RL agent may receive data from the controlled system. However, the gathering of observation data can be largely independent of the simulation-model simulator.

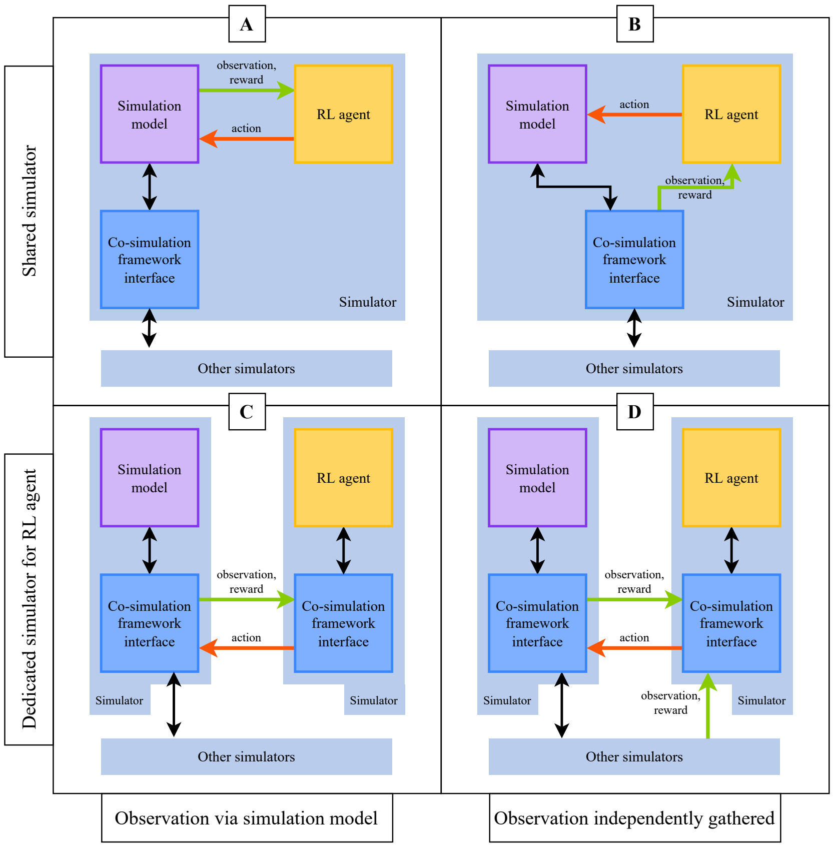
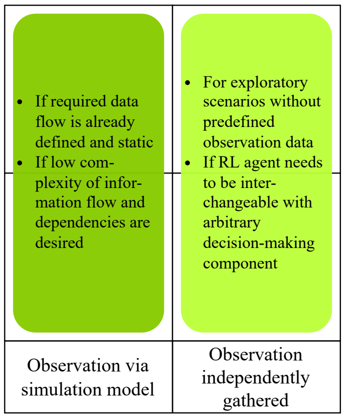

A
B
C
D
Shared simulator
Dedicated simulator for RL agent
Simulation model
RL agent
observation, reward
action
Co-simulation framework interface
Simulator
Other simulators
Observation via simulation model
Observation independently gathered

Fig. 4: Classification matrix for architecture candidates

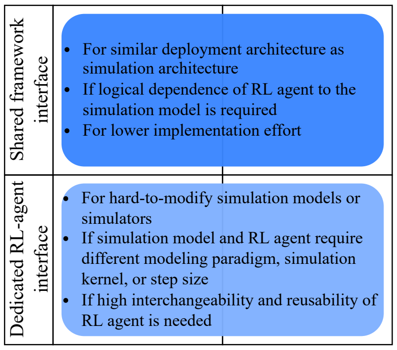
Guidance on Candidate Selection When deciding on how to integrate an RL agent into a co-simulation architecturally, one is faced with multiple considerations that are specific to the project at hand. We deem the presented classification scheme to be a helpful support for such a decision-making process. The two distinct binary categories shown in Figure 4 represent two independent binary decisions:

1. Should the observation (and the reward) for the RL agent be gathered independently from its simulation model?
2. Should the RL agent receive a dedicated co-simulation framework interface?

Regarding the first decision: An advantage of having independent data flow is that it allows for flexibility. This is especially useful for exploratory and experimental projects where the input data for the agent is not yet carved in stone. For example, a project may try to find which data can improve the RL agent's performance. Another project may attempt to evaluate the benefit of supplying the RL agent with an additional piece of information. Furthermore, the existing simulation components need less modification and are instead mostly extended, adhering to the open-close principle [19]. Also, the RL agent may be substituted for any other form of decision-making component that is not based on observation data. However, if the flexibility is not needed—in case of having unalterable and clearly defined data flow—the additional complexity of both information flow as well as dependencies, and implementation effort may be unnecessary. In the architecture candidates B and D shown in Figure 4, the agent gathers observation data independently. A summary of these considerations is presented in Figure 5a.


If required data flow is already defined and static
If low complexity of information flow and dependencies are desired
For exploratory scenarios without predefined observation data
If RL agent needs to be interchangeable with arbitrary decision-making component
Observation via simulation model
Observation independently gathered

(a) Independence of data gathering


Shared framework interface
Dedicated RL-agent interface
For similar deployment architecture as simulation architecture
If logical dependence of RL agent to the simulation model is required
For lower implementation effort
For hard-to-modify simulation models or simulators
If simulation model and RL agent require different modeling paradigm, simulation kernel, or step size
If high interchangeability and reusability of RL agent is needed

(b) Simulator dedicated to RL agent

Fig. 5: Considerations for selecting architecture candidates

To decide whether to use a dedicated simulator, the primary consideration is the mutability of the simulator containing the controlled simulation model. If the simulator is provided as a black-box executable, the only option is to create a dedicated simulator for the agent. This may be the case if an organization refuses to share implementation details of its simulation model. A further critical consideration is to what degree the simulation model is modifiable. Having a dedicated simulator alleviates the need for editing the simulation model since the communication runs across a predefined connection. Similarly, one must judge the heterogeneity of the simulation model and the RL agent. If the simulation model is of a different modeling language or paradigm, or requires a different runtime environment or simulation step size, it is advantagous to create an individual simulator for each. Moreover, having a dedicated simulator for RL leads to looser coupling and thus easier exchangeability as well as reusability of individual components. However, it separates components that are fundamentally conceptually linked and will likely be part of the same system in deployment. Architecture candidates C and D both use independent simulators, as shown in Figure 4. Figure 5b compactly presents the discussed trade-offs.

5 Identifying Framework Integration Problems

This section describes the implementation of an RL agent based on the case study. First, the co-simulation and RL framework we used are discussed and our reasons for choosing them are delineated. Next, we uncover an issue with control flow when using RL and co-simulation frameworks together. We also provide a possible solution that works with the tools we use.

5.1 Co-Simulation and Reinforcement Learning Frameworks

Our main requirements for choosing a co-simulation framework were that it be freely available as open-source, suitable to power-grid simulation, and flexible in two ways: First, it should allow programmatical scenario definition that is not bound to a graphical user interface. Second, the framework ought to be flexible in terms of what execution environment the simulators are run in. Among the 26 frameworks presented by Vogt et al. [34], we chose Mosaik[1], a smart-grid co-simulation framework originally introduced by Schutte et al. [29] and Rohjans et al. [26]. The framework's goal is to run coordinated simulations of energy-system scenarios that facilitate the use of existing simulators in a common context [17]. Mosaik provides two APIs for user interaction: The component API specifies the socket connection for data exchange between simulators and the framework. Via the scenario API, an executable co-simulation scenario can be created in which the entity instances, their parameters, and their connections are defined [31].

Regarding reinforcement learning, OpenAI Gym [4] presents a general structure for modeling environments as single classes with specified interfaces. The

[1] https://mosaik.offis.de

agent then exchanges action, observation, and reward with the environment (cf. Figure 2). Many RL frameworks support OpenAI Gym environments, such as TensorForce[2], Stable Baselines 3[3], and TF-Agents[4]. We have chosen to use the latter for this case study due to its flexibility. TF-Agents is described by Guadarrama et al. [14] and is based on the TensorFlow machine-learning library presented by Abadi et al. [1]. The framework implements several state-of-the-art RL algorithms and supports user-created environments. It uses modular components that are made to be extensible, and allows both high- and low-level access to many of its features.

5.2 Carrying Out and Implementing the Case Study

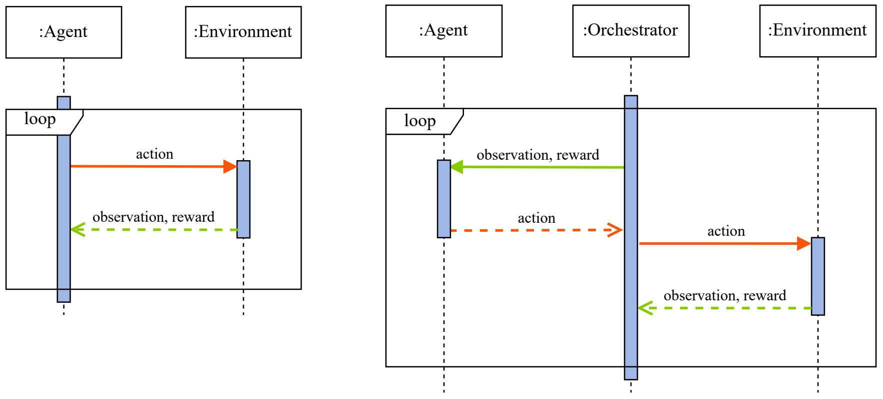
We conduct our case-study experiments based on the fictitious scenario outlined in Section 3. A small distribution grid was simulated including simulators for the power grid, the EV charging stations, a charging-station management system, as well as households. We implemented a Deep Q-Learning [16] agent to control the charging-station management system which is responsible for limiting the available charging power for all stations. The agent's learning goal was to determine an appropriate charging strategy that maximizes the available charging power while avoiding demand peaks. We tested the naïve architecture candidate (see Section 3) and the four candidates described in Section 4.3 while leaving the specifics of the RL agent unchanged. The experiments uncovered an issue: The intended control flow of TF-Agents must be changed for agents to be integrated into an orchestrated Mosaik co-simulation. This issue, its implications, and a possible solution are discussed in the following section.

5.3 Need for Changed Control Flow

RL frameworks generally implement a control paradigm based on the information flow seen in Figure 6a; these frameworks put the agent in control of stepping the environment through simulated time. The environment realizes the action and then returns an observation and usually a reward. In other words, the environment remains idle until it is prompted by the agent. In contrast, orchestrated co-simulation requires both the agent and the environment be part of the co-simulation; the orchestrator is responsible for stepping the simulators and exchanging data between them (see Figure 6b). Therefore, the RL frameworks' typical practice of modeling the environment as a single class and placing the agent in control of stepping the environment is incompatible with the paradigm of orchestrated co-simulation. This makes it necessary to consider how a given RL framework can be adapted to work with co-simulation, or—alternatively—to use a framework-independent RL implementation.

[2] https://github.com/tensorforce/tensorforce
[3] https://github.com/DLR-RM/stable-baselines3
[4] https://www.tensorflow.org/agents

When integrating an RL framework into an orchestrated co-simulation, several modifications are necessary. For example, when using TF-Agents, the agent class may stay intact; however, we suggest removing the environment class altogether. Instead, the orchestrator gathers observation and reward from other simulators and passes it to the simulator with the RL agent. As discussed in Section 4, that could be a simulator exclusively housing the agent or the agent together with other simulation models. The agent generates the action, which is collected by the orchestrator the next time it can be processed. Furthermore, it is necessary to implement some functionality from the environment class in the agent's simulator. This includes translating observations into a format that is compatible with the agent and generating a reward for each state-action pair. With these changes, TF-Agents was successfully integrated with Mosaik.



(a) Traditional framework-imposed control flow

(b) Orchestrator-based control flow

Fig. 6: Control flow and information exchange between RL agent and environment

6 Conclusion and Outlook

RL shows great potential for smart-grid applications such as demand-side management. However, RL hinges on the quality of simulation, and smart grids, as systems of systems, are inherently difficult to simulate. Co-simulation is a promising tool to address this issue and enable RL in complex systems of systems by providing a suitable environment for training and testing. However, research on RL in co-simulation is lacking. With this paper we take a first step towards closing that gap in literature. The paper establishes a preliminary overview of the disciplines required for using RL in a smart-grid co-simulation and analyzes co-simulation-specific challenges for RL. First, we examined architectural integration. Second, we conducted a case study to reveal framework-specific interoperability challenges.

To address the first research goal, we identify and assess a set of four architecture candidates. We further categorize them in a 2-by-2 matrix using two independent binary categories: the independence of the data flow to the integrated RL agent, and whether the agent receives a dedicated co-simulation framework interface. We then test the architecture candidates to evaluate them. Next, we discuss the application scenarios of each candidate according to their categorization and give guidance on when which candidate is appropriate. Furthermore, we tackle the second research goal—identifying framework integration issues—using a case study–based approach. The co-simulation framework Mosaik was used in tandem with the RL framework TF-Agents to implement a simple, fictitious scenario where smart EV charging is used for demand-side management. The experiments uncovered an issue: While RL frameworks typically assume a specific control flow, using RL in an orchestrated co-simulation requires changing that control flow. For the frameworks used in the case study, this issue could be resolved, showing that TF-Agents can be trained in Mosaik co-simulation scenarios.

Future research should be conducted in several areas. First, we assume that the framework-integration issue is not just limited to TF-Agents and Mosaik but instead we postulate that it hints at a general incompatibility of the control flow typically found in RL frameworks and the concept of orchestrated co-simulation. However, further research is required to verify that claim. To this end, it would be beneficial to analyze various combinations of tools to check for compatibility. Moreover, our architectural guidance should be applicable to multi-agent RL as well as the single-agent RL paradigm discussed here. This could be validated by applying our findings to multi-agent RL in smart-grid co-simulations, ideally in a more realistic and comprehensive case-study experiment.

Acknowledgements We thank Katharina Polanec for insightful discussions, valuable critique, and thorough proofreading.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, software available from tensorflow.org
2. Antonopoulos, I., Robu, V., Couraud, B., Kirli, D., Norbu, S., Kiprakis, A., Flynn, D., Elizondo-Gonzalez, S., Wattam, S.: Artificial intelligence and machine learning approaches to energy demand-side response: A systematic review. Renewable and Sustainable Energy Reviews **130**, 109899 (2020). https://doi.org/10.1016/j.rser.2020.109899

3. Binder, C., Fischinger, M., Altenhuber, L., Draxler, D., Lastro, G., Neureiter, C.: Enabling architecture based co-simulation of complex smart grid applications. Energy Informatics **2**(S1) (Sep 2019). https://doi.org/10.1186/s42162-019-0084-0
4. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
5. Cabot, J., Clarisó, R., Brambilla, M., Gérard, S.: Cognifying model-driven software engineering. In: Seidl, M., Zschaler, S. (eds.) Software Technologies: Applications and Foundations. pp. 154–160. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-74730-9_13
6. Crozier, C., Morstyn, T., McCulloch, M.: The opportunity for smart charging to mitigate the impact of electric vehicles on transmission and distribution systems. Applied Energy **268**, 114973 (2020). https://doi.org/10.1016/j.apenergy.2020.114973
7. Das, H., Rahman, M., Li, S., Tan, C.: Electric vehicles standards, charging infrastructure, and impact on grid integration: A technological review. Renewable and Sustainable Energy Reviews **120**, 109618 (Mar 2020). https://doi.org/10.1016/j.rser.2019.109618
8. DeLaurentis, D.: Understanding transportation as a system-of-systems design problem. In: 43rd AIAA Aerospace Sciences Meeting and Exhibit. pp. 123–136. American Institute of Aeronautics and Astronautics, Reno, Nevada (2005). https://doi.org/10.2514/6.2005-123
9. Denil, J., Meyers, B., De Meulenaere, P., Vangheluwe, H.: Explicit semantic adaptation of hybrid formalisms for fmi co-simulation. In: Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative & Symposium. p. 99–106. DEVS '15, Society for Computer Simulation International, San Diego, CA, USA (2015)
10. Farhangi, H.: The path of the smart grid. IEEE Power and Energy Magazine **8**(1), 18–28 (Jan 2010). https://doi.org/10.1109/MPE.2009.934876
11. Fischer, L., Memmen, J.M., Veith, E.M., Tröschel, M.: Adversarial resilience learning - towards systemic vulnerability analysis for large and complex systems (2018), https://arxiv.org/abs/1811.06447
12. Fitzgerald, J., Pierce, K., Larsen, P.G.: Co-modelling and co-simulation in the engineering of systems of cyber-physical systems. In: 2014 9th International Conference on System of Systems Engineering (SOSE). pp. 67–72 (2014). https://doi.org/10.1109/SYSOSE.2014.6892465
13. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: A survey. ACM Comput. Surv. **51**(3) (may 2018). https://doi.org/10.1145/3179993
14. Guadarrama, S., Korattikara, A., Ramirez, O., Castro, P., Holly, E., Fishman, S., Wang, K., Gonina, E., Wu, N., Kokiopoulou, E., Sbaiz, L., Smith, J., Bartók, G., Berent, J., Harris, C., Vanhoucke, V., Brevdo, E.: TF-Agents: A library for reinforcement learning in tensorflow. https://github.com/tensorflow/agents (2018), https://github.com/tensorflow/agents, [Online; accessed 25-June-2019]
15. Hafner, I., Popper, N.: An overview of the state of the art in co-simulation and related methods. SNE Simulation Notes Europe **31**(4), 185–200 (Dec 2021). https://doi.org/10.11128/sne.31.on.10582
16. Hasselt, H.v., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. p. 2094–2100. AAAI'16, AAAI Press (2016). https://doi.org/10.1609/aaai.v30i1.10295

17. Lehnhoff, S., Nannen, O., Rohjans, S., Schlogl, F., Dalhues, S., Robitzky, L., Hager, U., Rehtanz, C.: Exchangeability of power flow simulators in smart grid co-simulations with mosaik. In: 2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES). pp. 1–6 (2015). https://doi.org/10.1109/MSCPES.2015.7115410
18. Lopes, A., Lezama, R., Pineda, R.: Model based systems engineering for smart grids as systems of systems. Procedia Computer Science **6**, 441–450 (2011). https://doi.org/10.1016/j.procs.2011.08.083
19. Meyer, B., Milner, R., Bertrand, M.: Object-oriented Software Construction. Prentice-Hall international series in computer science, Prentice-Hall, Hoboken, NJ (1988)
20. Neureiter, C., Binder, C., Lastro, G.: Review on domain specific systems engineering. In: 2020 IEEE International Symposium on Systems Engineering (ISSE). pp. 1–8 (2020). https://doi.org/10.1109/ISSE49799.2020.9272214
21. Nguyen, V.H., Besanger, Y., Tran, Q.T., Boudinnet, C., Nguyen, T.L., Brandl, R., Strasser, T.I.: Using power-hardware-in-the-loop experiments together with co-simulation for the holistic validation of cyber-physical energy systems. In: 2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe). pp. 1–6. IEEE, Turin, Italy (Sep 2017). https://doi.org/10.1109/ISGTEurope.2017.8260122
22. Nguyen, V.H., Besanger, Y., Tran, Q.T., Nguyen, T.L.: On conceptual structuration and coupling methods of co-simulation frameworks in cyber-physical energy system validation. Energies **10**(12) (2017). https://doi.org/10.3390/en10121977
23. Nian, R., Liu, J., Huang, B.: A review on reinforcement learning: Introduction and applications in industrial process control. Computers & Chemical Engineering **139**, 106886 (2020). https://doi.org/10.1016/j.compchemeng.2020.106886
24. Palensky, P., Dietrich, D.: Demand side management: Demand response, intelligent energy systems, and smart loads. IEEE Transactions on Industrial Informatics **7**(3), 381–388 (2011). https://doi.org/10.1109/TII.2011.2158841
25. Palensky, P., Meer, A.A.V.D., Lopez, C.D., Joseph, A., Pan, K.: Cosimulation of intelligent power systems: Fundamentals, software architecture, numerics, and coupling. IEEE Industrial Electronics Magazine **11**(1), 34–50 (Mar 2017). https://doi.org/10.1109/MIE.2016.2639825
26. Rohjans, S., Lehnhoff, S., Schütte, S., Scherfke, S., Hussain, S.: mosaik - a modular platform for the evaluation of agent-based smart grid control. In: IEEE PES ISGT Europe 2013. pp. 1–5 (2013). https://doi.org/10.1109/ISGTEurope.2013.6695486
27. Sage, A.P., Cuppan, C.D.: On the systems engineering and management of systems of systems and federations of systems. Information-Knowledge-Systems Management **2**, 325–345 (2001)
28. Schloegl, F., Rohjans, S., Lehnhoff, S., Velasquez, J., Steinbrink, C., Palensky, P.: Towards a classification scheme for co-simulation approaches in energy systems. In: 2015 International Symposium on Smart Electric Distribution Systems and Technologies (EDST). pp. 516–521. IEEE (Sep 2015). https://doi.org/10.1109/SEDST.2015.7315262
29. Schutte, S., Scherfke, S., Troschel, M.: Mosaik: A framework for modular simulation of active components in smart grids. In: 2011 IEEE First International Workshop on Smart Grid Modeling and Simulation (SGMS). pp. 55–60. IEEE, Brussels, Belgium (Oct 2011). https://doi.org/10.1109/SGMS.2011.6089027
30. Schweiger, G., Gomes, C., Engel, G., Hafner, I., Schoeggl, J., Posch, A., Nouidui, T.: An empirical survey on co-simulation: Promising standards, challenges and

research needs. Simulation Modelling Practice and Theory **95**, 148–163 (Sep 2019). https://doi.org/10.1016/j.simpat.2019.05.001

31. Steinbrink, C., Blank-Babazadeh, M., El-Ama, A., Holly, S., Lüers, B., Nebel-Wenner, M., Acosta, R.R., Raub, T., Schwarz, J., Stark, S., Nieße, A., Lehnhoff, S.: CPES testing with mosaik: Co-simulation planning, execution and analysis. Applied Sciences **9**(5), 923 (Mar 2019). https://doi.org/10.3390/app9050923
32. Sutton, R.S., Barto, A.G.: Reinforcement Learning. Adaptive Computation and Machine Learning series, Bradford Books, Cambridge, MA, 2 edn. (Nov 2018)
33. Veith, E.M., Wenninghoff, N., Frost, E.: The adversarial resilience learning architecture for ai-based modelling, exploration, and operation of complex cyber-physical systems (2020), https://arxiv.org/abs/2005.13601
34. Vogt, M., Marten, F., Braun, M.: A survey and statistical analysis of smart grid co-simulations. Applied Energy **222**, 67–78 (Jul 2018). https://doi.org/10.1016/j.apenergy.2018.03.123
35. Vázquez-Canteli, J.R., Nagy, Z.: Reinforcement learning for demand response: A review of algorithms and modeling techniques. Applied Energy **235**, 1072–1089 (2019). https://doi.org/10.1016/j.apenergy.2018.11.002
36. Walden, D., Roedler, G., Forsberg, K., Damelin, D., Shortell, T.: INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, 4th Edition. Wiley, Hoboken, NJ (2015)
37. Zhang, D., Han, X., Deng, C.: Review on the research and practice of deep learning and reinforcement learning in smart grids. CSEE Journal of Power and Energy Systems **4**(3), 362–370 (2018). https://doi.org/10.17775/CSEEJPES.2018.00520