INTERNATIONAL BUILDING PERFORMANCE SIMULATION ASSOCIATION ENGLAND

Proceedings of BSO-V 2020
5th Building Simulation and Optimization Virtual Conference
21-22 September 2020
Loughborough, UK

Integration and Evaluation of Deep Reinforcement Learning Controller in a Building Co-Simulation Environment

Ahmed Amrani[1], Rim Kaddah[1], Jean-Philippe Tavella[2], Mathieu Schumann[2]
[1]IRT SystemX, Palaiseau, France
[2]EDF Lab Paris-Saclay, Palaiseau, France

Abstract

Deep Reinforcement Learning (DRL) is a promising Artificial Intelligence (AI) approach for buildings heating control. However, DRL controllers require dynamic simulation involving heterogeneous physical models (building, power-grid, etc.). Co-simulation allows the inter-operation between heterogeneous components when exported following the Functional Mock-Up Interface (FMI) standard.

Controllers based on DRL can be implemented using various languages but are mostly based on Python libraries like Tensorflow. Their integration into co-simulation environments requires their exportation as Functional Mockup Units (FMUs). Thus, language-specific FMI-compliant export tools are needed for every programming language, specific library or platform used. This process is costly in effort and time and results in large FMUs.

This paper proposes a novel method that simplifies AI-based controller integration regardless of the language or platform in a co-simulation environment and apply our methodology to assess a DRL controller. For the first objective, we use existing FMI export tools to create an FMU having the same input and output parameters as the controller. In the proposed architecture, the FMU acts as a proxy whose objective is to communicate with an external DRL controller deployed on a local or remote machine. We propose an application of this generic architecture for heating control in a house using DACCOSIM NG co-simulation environment. Through this architecture, the deployed DRL-based controller is connected to a house energy model, which includes weather conditions and heating. We show that our proposed controller is capable to learn the system dynamics and keep temperature within 1 degree of setpoint 93% of the time.

Introduction

Research questions related to building smart control systems can arise aim to tackle challenges at the building, the district or community level, a scale where the building interacts with the other components of the energy system and the grid, forming a complex system (Vialle et al., 2017; Reinbold et al., 2019). Research at these levels bring together experts from various fields, each with their own area of expertise, knowledge and models (building energy, electricity and heating networks, energy generation, etc.). The simulation of such complex environment involves a variety of components based on a specific physics, which might be created using generic or domain-specific tools and implemented by various teams. In this situation, the use of co-simulation environments allows the inter-operation between heterogeneous components, and an increasing number of modelling environments became compatible with the Functional Mock-Up Interface (FMI) standard for interoperability.

AI-based approaches are added to this cross-domain applications, creating a need to find a simple way to integrate such systems and make them interoperate especially AI algorithms involving online calibration.

As matter of fact and despite major technological developments, optimal control of heating in buildings remains an important challenge for which adaptive AI approaches can bring new effective solutions.

In practice, the complexity of building thermal dynamics, the stochasticity of occupant behaviour and the heterogeneous environmental disturbances make classical rule-based and model-based control strategies inefficient due to lack of adaptability and model update. Furthermore, it is still a challenge to implement an accurate representation of building dynamics into a controller in real buildings (Zacekova et al., 2014), and its use for real-time control raises issues of computation time (Yang et al., 2015).

Deep reinforcement learning (DRL) is being studied in recent works as a promising AI approach for heating control in buildings, based on real-time measured data as an input (Nagy et al., 2018; Wei et al. 2017; Kazmi et al., 2017; Mason and Grijalva, 2019). Moreover, compared to classical Q-learning, DRL is better suited for control problems in a large parameter space (Wei et al., 2017). Among the methods for assessing the performance of a DRL controller, the use of dynamic simulation is a flexible mean for testing various situations and explore parameters.

Controllers based on DRL themselves can also be implemented using various languages. Most recent works mainly use Python libraries such as TensorFlow or PyTorch (Nguyen et al., 2019). The integration of such components into a co-simulation environments would normally require to export them as executable components (FMU), which raises number of difficulties elaborated in related works section.

We propose a generic approach for enabling compatibility between FMI-based co-simulation environments and

external algorithms, and apply this architecture to assess the performance of a DRL controller to learn an effective control strategy for a building heating system.

The paper is organised as follows: In the next section, we present related work regarding co-simulation and interoperability. Then, we present the DACCOSIM NG co-simulation platform which we use in our use case. Next, we present the new architecture designed to connect a DRL controller to a co-simulation environment. The proposed approach is then applied to the assessement of a DRL-based building heating control system which is presented along with promising results in section Simulation and experiments. Finally, we provide conclusions and perspectives.

Related work

Co-simulation

Physical models can be created using numerous tools (Dymola, Matlab, Simulink, etc.), or programming languages (Java, Python, etc.). Co-simulation is a solution for enabling interoperability between a variety of executable components, each based on a particular domain of expertise and specific physics. Co-simulation also allows scaling-up of very large Cyber-Physical Systems such as the energy systems, composed of various componens such as buildings and grids, by splitting the whole system in relatively independent components, each aggregating a part of the dynamics of the global system (Reinbold et al., 2019; Tavella, et al., 2016; Ptolemaeus, 2014). The Functional Mock-Up Interface standard (FMI standard, n.d.) is a framework for the development of such interoperable executable models. The components exported using this standard are called Functional Mock-Up Units (FMU) and are composed of an XML file which defines the model inputs, outputs and parameter, as well as a C-code compiled for execution on a targeted operating system.

We propose to use the DACCOSIM NG platform (Dad et al., 2016; Galtier et al., 2015; Galtier et al., 2017; Vialle et al., 2017), an open-source plateform for FMI-based co-simulation developped and maintained by EDF and CentraleSupélec (DACCOSIM NG tool, n.d.), and which is already applied to the study of buildings and energy networks as a complex, multi-domains Cyber-Physical System. DACCOSIM NG is a co-simulation master that allows parallel computation of all sub-models in order to achieve scaling-up and reduced computation time (ModeliScale, n.d.).

Control integration

Several libraries exist to allow the exploitation of FMUs from a Python program, such as pyFMI (Andersson et al., 2016) and OMPython (Ganeson et al., 2012). The OMPython library allows python code to interface with OpenModelica in order to run simulations. The pyFMI library is used by the libraries Dymrl and ModelicaGym (Lukianykhin and Bogodorova, 2019) in order to exploit Modelica or Dymola FMUs as an OpenAI Gym environment.

However in our context, the co-simulation environment that is used for the study of Cyber-Physical Systems is a master of co-simulation, and need to be able to call AI-based Python libraries or tools from within the co-simulation. In the co-simulation, the controller requires an interaction with the controlled component at each iteration. Therefore, it is necessary to generate the AI-based control components in the form of an FMU as is the case for the other components.

Thus, FMI compliant export capabilities would need to be implemented or adapted each time a new programming language, specific library or platform is used. JavaFMI (Hernández-Cabrera et al., 2020) and SimulatorToFMU (Nouidui and Wetter, 2018) are software packages which allows to export a Java or Python program as a Functional FMU for use in a co-simulation environment. However, depending on the targeted platform, language and library, this process can be costly in terms of development effort and time, or even be impossible at this time because of the complexity of the desired Python libraries. Moreover, it can generate large FMU models since it requires the inclusion of external libraries.

Co-simulation environment

To benefit from the advantages of co-simulation, we propose a new architecture to use AI-based components in a FMI-based co-simulation environment. Although the methodology can be generalized to co-simulation in general, we demonstrate the proposal in the DACCOSIM NG (DNG) co-simulation environment.

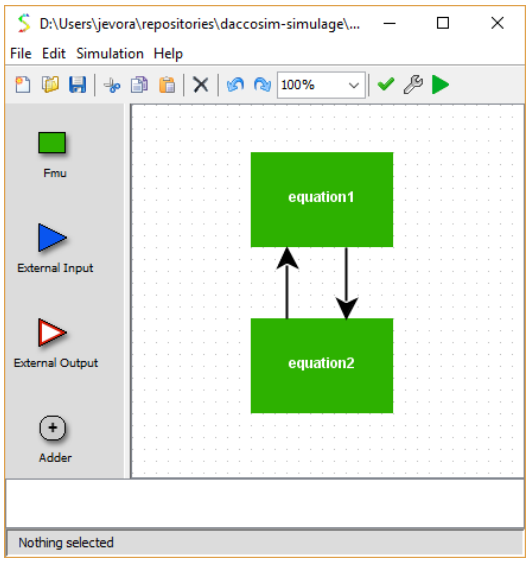

Figure 1: General co-simulation architecture (DACCOSIM NG tool, n.d.).

DACCOSIM NG (DNG) is a generic FMI-based co-simulation master and simulation environment developed in Java language. DNG allows the design and execution of co-simulation graphs based on executable components exported as FMUs from FMI-compatible tools. Co-simulation graphs are defined either through a Graphical

User Interface (GUI) or scripts (CLI). A user's guide is available with the tool distribution for more detailed information about the usage of DNG (DACCOSIM NG tool, n.d.).

Graphs

A co-simulation graph is composed of nodes and arrows, representing connections between nodes. The general architecture is depicted in figure 1. Nodes can be FMUs, operators (adder, multiplier, offset and gain), or external inputs/outputs. Connections define which output variables of a source node are connected to which input variables of a target node. Before running a graph, DNG starts by opening the file in which the co-simulation is defined and loads the graph in memory. Then each FMU in the graph is also loaded so that the DNG engine can operate the co-initialization and stepping.

Co-initialization

Co-initialization is the setting of consistent system-wide initial values for all the components. The algorithm starts by building a global dependency directed graph for the connected variables of the FMUs. It uses the connections established by the user to find external dependencies between the outputs from source FMUs and the inputs from sink FMUs. Thanks to Tarjan's SCC algorithm (Tarjan, 1972), Strongly Connected Components (SCCs) corresponding to cyclic dependencies in the graph are contracted into single vertexes to get a Directed Acyclic Graph (DAG) naturally giving the order in which the variables must be initialized. SCCs are solved in a parallel way thanks to an iterative algorithm called JNRA (Jacobian based Newton-Raphson Algorithm) inspired by traditional Newton-Raphson algorithms.

Stepping

DNG Master can be parametrized with constant or variable steppers. Variable stepping is more efficient as integration steps can increase when state variables are not strongly evolving. On the contrary, when the error controlled by the Master is too important, a smaller step is required and synchronization between FMUs is possible with the rollback feature offered by the FMI standard. A fixed step size (constant stepping) is used when FMUs cannot rollback.

AI-based model integration

Global view of the software architecture

In order to use a Python AI-based controller in an FMI-based co-simulation, we propose a software architecture that uses a dedicated FMU having the same input and output parameters as the controller, but acting as a proxy between the co-simulation environment and the external AI-based controller that can be deployed on a local or remote machine.

The proposed architecture is based on the following components (see figure 3):

- An inter-process communication mechanism, where the FMU controller proxy acts as a client and the effective controller acts as a server.
- An encoding/decoding message mechanism used to encode in binary format the structured messages transmitted between the client and the server.
- The effective (AI-based) controller, an external module implemented with the most suitable programming language and libraries. It requires dynamic interaction with the physical models.
- The FMU controller proxy, a new type of FMU integrated in the co-simulation environment. This FMU has the same inputs and outputs as the controller. It acts as a proxy whose objective is to communicate with the effective controller.

Inter-process communication mechanism

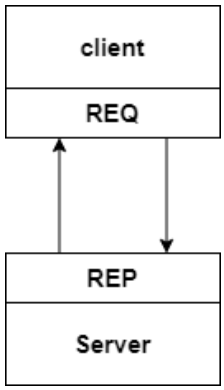

Figure 2: request-reply pattern.

To ensure communication between the FMU controller proxy (client) and the effective AI-based controller (server) we use the ZeroMQ (ZeroMQ, n.d.) middleware. ZeroMQ is a messaging system that uses sockets to carry atomic messages. ZeroMQ allows different messaging patterns such as synchronous Request/Response, asynchronous Request/Response, Publish/Subscribe, or Push/Pull. To establish the communication between the FMU controller proxy and the effective controller, we use synchronous Request/Replay messaging type (figure 2). In this pattern, the client sends a request and the server replies to it. The server blocks on receive until it receives a request and the client blocks on send until it receives a reply back from previous request.
The low-level library of ZeroMQ is implemented in C++ and exposes a C-API. ZeroMQ supports binding for several languages such as Python with PyZMQ library. It offers a native implementation for some languages such as Java with JeroMQ library (Jeromq, n.d.).

The encoding/decoding message mechanism

In our architecture, there are two types of messages exchanged: control actions, and the states of some FMU

physical models. In order to encode and to decode these messages, we use the Google protocol buffers (Protobuf) which is a mechanism for serializing structured messages. With Protobuf, the structure of the messages exchanged must be specified using the protocol specific format. Then, for each language, Protobuf generates a code to encode and decode the messages. In our architecture, the generated code is used in the effective controller and in the FMU proxy controller.

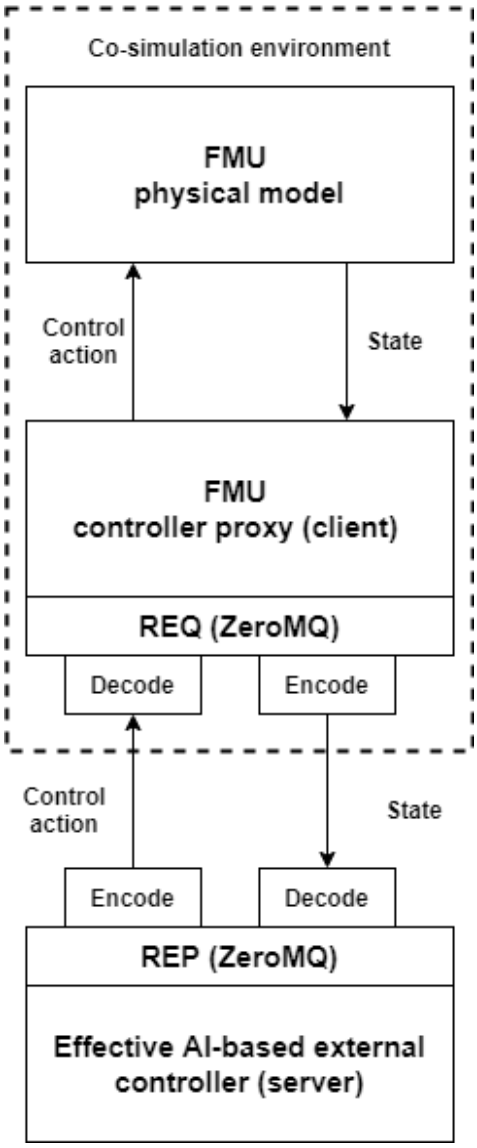

Figure 3: AI-based control architecture.

The effective controller

AI-based controllers can be implemented using different languages. However, most recent, advanced and powerful deep learning algorithms are based on Python libraries such as TensorFlow, Keras or keras-rl. We implemented the building heater controller in Python using these libraries.

As for the specific type of AI-based control, we look into Deep Reinforcement Learning schemes as a way to integrate an adaptive control mechanism. Hence, at each co-simulation iteration, the DRL controller receives the system state and decides on a control action. Since the controller acts as a server, it sends an action every time it receives state information of the simulated physical system. Therefore, the controller implements the ZeroMQ messaging patterns "synchronous Request/Response" using the Python PyZMQ library.

Generation of the FMU Controller proxy

The FMU controller proxy is the FMU integrated in the co-simulation platform. We implement it in Java and export it into an FMU using the Java FMU-builder tool part of JavaFMI (Hernández-Cabrera et al., 2020). The FMU controller proxy acts as a client: at each iteration of co-simulation, it sends the system state to the effective controller and receives the control action. Similarly to the effective controller, the FMU controller proxy implements the ZeroMQ messaging patterns "synchronous Request/Response" using the Java JeroMQ library (Jeromq, n.d.).

Key features of the proposed approach

The proposed approach has several advantages over the approach of generating an FMU from a Python program and its dependencies:

- The controller can be executed on the local machine or on a remote machine. In the case where the controller requires large computational resources as it might be the case for Deep Learning, it is possible to deploy it on remote machine with an adapted computing power (e.g. GPU machine).
- The phase of code porting to the co-simulation execution environment (e.g., different CPU, operating system, etc.) can be avoided. Depending on the operating system, language and library, code porting can be very complex and time consuming.
- The generation of large FMUs is avoided when the AI model is based on libraries of significant size.
- We keep the possibility to easily evaluate several algorithms developed with different languages, different frameworks and for different platforms.

In the following section, we use this new architecture to assess the efficiency of a DRL-based controller for heating control in a residential buildings.

Simulation and experiments

Description

For this Use Case, we built a single-family house energy model using the BuildSysPro Modelica library (BuildSysPro, n.d.; Plessis et al., 2014). The building envelope is a detailed monozone thermal model including

1-D walls, windows as well as air renewal to take all the heat transfer modes into account (figure 4).

The global building model, as shown in figure 5, includes a weather file reader, and two ways of controlling heating: by means of an ideal PI controller, or by controlling directly the power injected in the building air node. In both cases we assume the heating system to be ideal, such as the injected heating power is equal to the consumed electrical power.

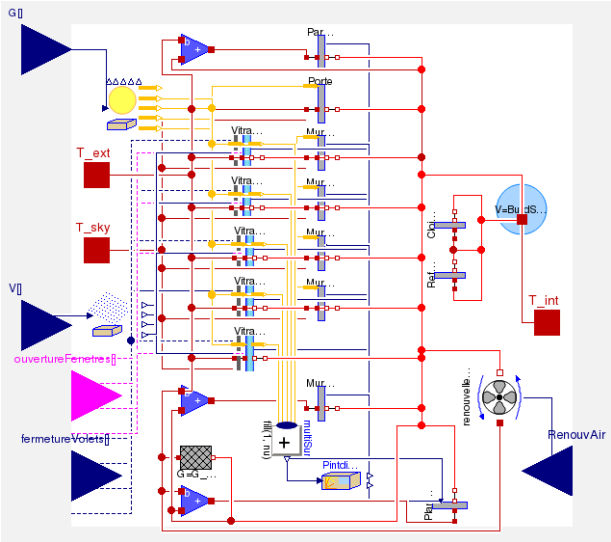

Figure 4: Modelica building envelope model.

The model includes inputs (occupant presence, set point for power, set point for temperature) and outputs (temperatures and heating power and energy), that are visible in a co-simulation environment once exported as an FMU.

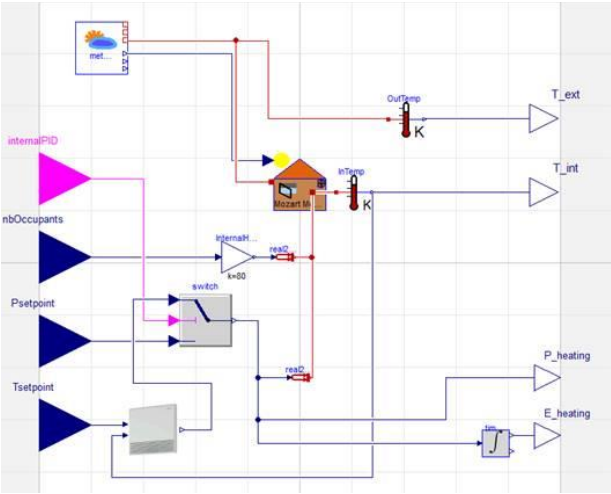

Figure 5: Building model for heating control, to be exported as an FMU.

The building model is used to compute indoor temperature by taking into account the dynamics of the building envelope, the use of a heating system, and the impact of external weather conditions.

The proposed controller is also connected to the weather reader in order to integrate external environmental variables, as well as indoor temperature and the household occupancy schedule.

To demonstrate the functioning of our proposed architecture, discuss the effectiveness of DRL for heating, and pave the way for scaling-up at the district/smart grid scale, we use the DACCOSIM NG environment to co-simulate the household and the controller (figure 3).

To implement the DRL controller, we use a Deep Q Learning (DQN) approach (Mnih et al., 2015). DQN has been studied in previous litterature as well as other Deep learning based algorithms (Wei et al., 2017; Nagy et al., 2018) . In our proposed method, the controller acts on the heating system injected power, making control more challenging but opens new opportunities as elaborated in the conclusion. Our solution is implemented as follows.

Controller state space

In this model the heating control is impacted by indoor temperature and the outdoor temperature T_{out} (°C).

Since, in order to use DQN, Markov property should be valid, we found that including historical information on T_{in} is required. This way we include in the control process relevant information on the building thermal state and inertia that we don't directly access through the indoor temperature or external environmental variables at each time step. So, the state at time step t is represented by $S_t = (T_{out}^t, T_{in}^t, \ldots, T_{in}^{t-n})$. Input values, in this case, are continuous.

Controller action space

We propose to control the power consumed by the heating system and provided to the building, assuming the heater efficiency is equal to 1. This provides a larger flexibility by modulating power input. For this, our action space is represented by a discretisation of power into power levels. These levels include turn off capability associated with a power level equal to 0. Action taken at time slot t is represented by $A_t = (P_t)$ where P_t denotes power consumption.

Controller model and q-values update

The controller model encoding q-values is an Artificial Neural Network (ANN) formed of an input layer expressing the state space, two hidden layers with a sigmoid activation function and final output layer expressing Q values for each action.

In the training process, q-value estimates are updated following eq. (1):

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\,(r_{t+1} + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)) \quad (1)$$

In eq. (1), $\alpha \in [0,1]$ is the learning rate where 0 is associated with no learning and 1 is associated with leaning and no memory on previous q-values.

The discount factor $\gamma \in [0,1]$ allows in our case to integrate in the importance of heating inertia. Hence, a value of 0 does not integrate any effect of decision on the future and 1 attributes a high importance to future states and associated optimal decisions.

The controller is assessed based on its capability to follow a target temperature denoted by T_{ref}. We use reward function of eq.(2) to model the desired behaviour.

$$r_t = -\left(T_{in}^t - T_{ref}\right)^2 \quad (2)$$

The training process is conducted on batches selected from historical decisions and state transition experience.

So given a batch size N, a batch is the set of N latest observations ending by the last state-decision made.

Action selection strategy

In the learning process, the decision is taken such that the action providing the greatest q-value is selected. We found that through learning, an implicit exploration will take place around the desired setpoint. Implementing a ε-greedy random exploration, in this case, showed worst results as it drives the exploration in the direction of overheating or underheating which are not in our zones of interest.

Results and discussion

Analysis parameter taken to implement the DQN described previously are summarized in table 1.

Table 1: Numerical analysis parameters.

Parameter	Value	Parameter	Value
Simulation period	63072000s (2 years)	γ	0.2
co-simulation time step	300s	Action power levels (W)	[0, 1000, 2000]
α	1	T_{ref}	22°C

The simulation period is represented by 4 years of winter period (2 years in term of total time). For ANN model parameters, input historical information on T_{in} is chosen n=1. Hidden layers are of size 16. Batch size N is taken equal to 576 (2 days). The model is implemented using the Tensorflow library.

Controller temperature tracking results are shown in figure 6. The controller is capable of remaining very close to T_{ref} with an average temperature of 21.83°C and a standard deviation of 0.92 during the first winter period then an average temperature of 21.9°C and a standard deviation of 0.58 in each of the subsequent winter periods;

As a matter of fact, temperature is kept within 0.5 degree of setpoint 83% of the time and within 1 degree 93% of the time. Indeed, as shown in figure 7, it requires 8100 iteration for the algorithm to reach a stable behaviour (~1 month).

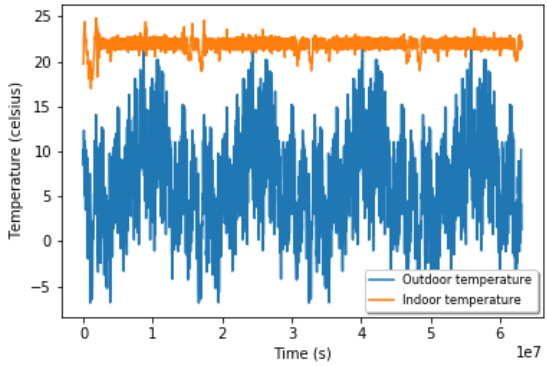

Figure 6: Evolution on indoor and external temperature as a function of time.

While giving the possibility of taking 3 power levels, in the end, the controller only makes on/off decisions with on corresponding to 2000W.

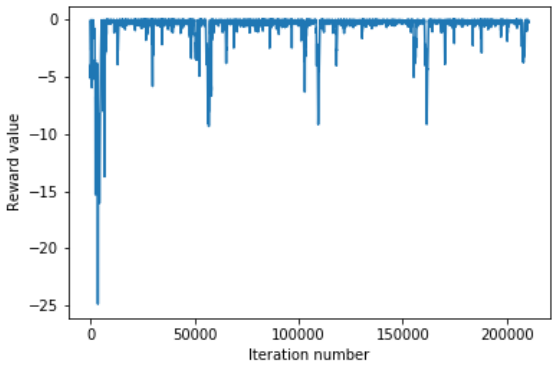

Figure 7: Evolution of reward over iterations.

Conclusion

We presented a novel methodology to ease the testing of new AI-based adaptive control models when used in a co-simulation environment. For this, we capitalise on efficient technologies allowing to work around FMU model encapsulation, to propose a proxy controller which is able to communicate with external AI libraries. This methodology could be extended to any type of communication between an executable FMU model in a co-simulation environment and an external algorithm.

An efficient DRL method has been presented and shows the capability of the controller to remain in the vicinity of the desired temperature setpoint. Since proposed solution acts on power, an extension of the solution will be developed and analysed in future works to provide

demand side management services targeting a desired power profile on a neighborhood level. In this case, power is modulated in a way that temperature does not stray away from setpoint expressed by the user while providing service to the power grid. This will yield in a more complex model that can integrate for instance strategies to reduce energy cost based on a price signal. We also plan on conducting further analysis on the impact of missing variables and the way it can affect the efficiency of DRL control, as well as computational performance and overhead when scaling up to the district level.

Acknowledgement

This research work has been carried out at the Institute of Technological Research SystemX, within the scope of a joint collaboration with EDF in the Paris-Saclay Energies (PSE) project.

References

Andersson, C., Åkesson, J. and Führer, C. (2016). PyFMI: A Python Package for Simulation of Coupled Dynamic Models with the Functional Mock-up Interface. Retrieved from https://portal.research.lu.se/portal/files/7201641/pyfmi_tech.pdf

BuildSysPro. (n.d.). Retrieved from https://github.com/EDF-TREE/BuildSysPro

DACCOSIM NG tool. (n.d.). Retrieved from https://bitbucket.org/simulage/daccosim

Dad, C., Vialle, S., Caujolle, M., Tavella, J.-P. and Ianotto, M. (2016). Scaling of distributed multi-simulations on multi-core clusters. IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, 142-147.

FMI standard. (n.d.). Retrieved from https://www.fmi-standard.org

Galtier, V., Ianotto, M., Caujolle, M., Corniglion, R., Tavella, J.-P., Gómez, J. É. and Kremers, E. (2017). Building Parallel FMUs (or Matryoshka Co-Simulations). Proceedings of the 12th International Modelica Conference, 663-671.

Galtier, V., Vialle, S., Dad, C., Tavella, J.-P., Lam-Yee-Mui, J.-P. and Plessis, G. (2015). FMI-based distributed multi-simulation with DACCOSIM. Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, 39-46.

Ganeson, A., Fritzson, P., Rogovchenko, O., Asghar, A., Sjölund, M. and Pfeiffer, A. (2012). 9th International Modelica Conference (Modelica'2012). An OpenModelica Python Interface and its use in PySimulator, (pp. 537-548). Munich.

Hernández-Cabrera, J. J., Évora-Gómez, J. and Roncal-Andrés, O. (2020). javaFMI. Retrieved from https://bitbucket.org/siani/javafmi/wiki/Home

Jeromq. (n.d.). Retrieved from https://github.com/zeromq/jeromq

Kazmi, H., Mehmood, F., Lodeweyckx, S. and Driesen, J. (2017). Gigawatt-hour Scale Savings on a Budget of Zero: Deep Reinforcement Learning based Optimal Control of Hot Water Systems. Energy, 144.

Lukianykhin, O. and Bogodorova, T. (2019). EOOLT. ModelicaGym: applying reinforcement learning to Modelica models. pp 27–36

Mason, K. and Grijalva, S. (2019). A review of reinforcement learning for autonomous building energy management. *Computers & Electrical Engineering*, 78, 300-312.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M. and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529-533.

ModeliScale. (n.d.). Retrieved from https://www.tenerrdis.fr/fr/projets/modeliscale

Nagy, A., Kazmi, H., Cheaib, F. and Driesen, J. (2018). Deep Reinforcement Learning for Optimal Control of Space Heating. Building Simulation and Optimization Conference, (pp. 96-103). Cambridge.

Nguyen, G., Dlugolinsky, S., Bobák, M., Tran, V., García, Á. L., Heredia, I. and Hluchý, L. (2019). Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review 52*. 77-124

Nouidui, T. and Wetter, M. (2018). Building Performance Analysis Conference and SimBuild. SimulatorToFMU: A Python Utility to Support Building Simulation Tool Interoperability. Chicago.

Plessis, G., Kaemmerlen, A. and Lindsay, A. (2014). BuildSysPro: a Modelica library for modelling buildings and energy systems. Modelica Conference. Retrieved from https://github.com/EDF-TREE/BuildSysPro

Ptolemaeus, C. (2014). System Design, Modeling, and Simulation Using Ptolemy II.

PyFMI. (n.d.). Retrieved from https://pypi.org/project/PyFMI/

Reinbold, V., Protopapadaki, C., Tavella, J.-P. and Saelens, D. (2019). Assessing scalability of a low-voltage distribution grid co-simulation through functional mock-up interface. *Journal of Building Performance Simulation*, 12(5), 637-649.

Tarjan, R. (1972). Depth first search and linear graph algorithms. SIAM JOURNAL ON COMPUTING, 1.

Tavella, J.-P., Caujolle, M., Vialle, S., Dad, C., Tan, C., Plessis, G. and Revol, S. (2016). Toward an accurate and fast hybrid multi-simulation with the FMI-CS standard. IEEE 21st International Conference on Emerging Technologies and Factory Automation, 1-5.

Vialle, S., Tavella, J.-P., DAD, C., Corniglion, R., Caujolle, M. and Reinbold, V. (2017). Scaling FMI-CS Based Multi-Simulation Beyond Thousand FMUs on Infiniband Cluster. Modelica Conference , 15-17.

Wei, T., Wang, Y. and Zhu, Q. (2017). Deep reinforcement learning for building HVAC control. Design Automation Conference. pp. 1-6

Yang, L., Nagy, Z., Goffin, P. and Schlueter, A. (2015). Reinforcement learning for optimal control of low exergy buildings. *Applied Energy*, 156, 577-586.

Zacekova, E., Vána, Z. and Cigler, J. (2014). Towards the real-life implementation of MPC for an office building: Identification issues. *Applied Energy*, 53-62.

ZeroMQ. (n.d.). Retrieved from zeromq: https://zeromq.org/