2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) | 978-1-6654-2484-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/MODELS-C53483.2021.00038

Inference of Simulation Models in Digital Twins by Reinforcement Learning

Istvan David, Jessie Galasso, Eugene Syriani
DIRO, Université de Montréal, Canada
istvan.david@umontreal.ca, jessie.galasso-carbonnel@umontreal.ca, syriani@iro.umontreal.ca

***Abstract*—The typical optimization and control activities of Digital Twins are driven by high-performance simulators. Due to the significant complexity of systems subject to digital twinning, constructing simulators of appropriate details is a costly and error-prone endeavor. To alleviate these problems, we propose an approach for inferring simulation models of Digital Twins by machine learning. Instead of learning the simulation model of one specific simulator, we aim at learning their construction process. This generality enables reusing the inferred knowledge in different (but congruent) Digital Twin settings. To achieve this level of generality, we propose the Discrete Event System Specification (DEVS) formalism for capturing simulation models; and reinforcement learning (RL) for inferring DEVS models. In this paper, we explore the opportunities and challenges in combining these two techniques.**

***Index Terms*—digital twins, simulation, DEVS, reinforcement learning**

I. Introduction

Digital twinning is a transformational trend in many domains, such as smart production, precision healthcare, and industrial energy management. Digital Twins (DT) are virtual representations of physical assets, used for real-time monitoring, control, and optimization of the said physical asset [1]. The purpose of a DT is to provide a proxy towards data-intensive applications needing to access data on the physical asset. A pertinent example of such data-intensive applications are simulators. Since the early 2000s, the typical role of simulation has shifted from the design phase of complex systems to their run-time phase, as simulators became first-class components in nowadays' complex systems, and enablers to intricate techniques, such as DT [2]. At the core of the simulator, the physical asset is represented by a formal model, from which complex algorithms calculate the metrics of interest. This model has to capture the specificities of the physical asset in appropriate details in order to consider the results of the simulation representative. However, due to the complexity of the systems subject to digital twinning, constructing these models by hand is an error-prone, time-consuming, and costly endeavor [3]. Automation of model construction can significantly alleviate these problems.

In this paper, we propose an approach for the automated inference of simulation models in DT by machine learning. Investing resources into the learning process only pays off if the learned simulation model is representative to a *class of problems* rather than one specific case. For example, after learning how to construct the DT of a robot arm, one would like to use this knowledge to construct the DT of a robot leg. To this end, we propose the generalization of the simulation model at the core of the DT to increase its representativeness. The Discrete Event System Specification (DEVS) formalism [4] is a prime candidate for this purpose, as the versatility of DEVS allows modeling the reactive behavior of real systems in great details, including timing and interactions with the environment—two aspects predominantly present in the physical assets of DT [5]. This increased generality comes at the cost of increased essential complexity, making the manual construction of DEVS models an error-prone and costly endeavor. Thus, proper automation becomes paramount. Reinforcement learning (RL) addresses well this problem. First, RL can be used to learn *how to solve a class of problems*, rather than learning a solution to a particular case, if provided with a problem at the appropriate level of abstraction. DEVS has been shown to be the common denominator of many other simulation formalisms [6], thus choosing appropriately high levels of abstraction becomes feasible. Second, RL is a general framework to solve sequential decision-making problems, based on an explicitly modeled set of actions. DEVS models are constructed through a well-defined set of possible actions, derived from the strict mathematical specification of the DEVS formalism. The main contribution of this paper is a conceptual framework that integrates DEVS and RL for inferring simulation models of DT. Furthermore, we provide the specification of mapping DEVS models onto RL concepts, and identify the main challenges and opportunities in implementing and deploying the framework.

II. Background

Discrete Event Simulation and Machine Learning in and for Digital Twins. Simulation methods are among the most referred techniques in DT concepts [7]. Discrete event simulation (DES) techniques are particularly popular due to their versatility and applicability in various classes of engineering problems. Kayani et al. [8] present a case study of applying DES for reliable and fast scheduling analysis in pharmaceutical production systems. Karakra et al. [9] present how to use DES to assess the efficiency of existing health care delivery systems in hospitals. Advanced DES techniques are utilized in industrial-scale settings as well, e.g., for assessing thermal efficiency and resilience [10]. The design and development of DTs requires processing and analyzing big data. Due to this challenge, the development of intelligent DTs is only possible

978-1-6654-2484-4/21/$31.00 ©2021 IEEE
DOI 10.1109/MODELS-C53483.2021.00038

by applying advanced artificial intelligence (AI) and machine learning (ML) techniques on the collected data. The high-level workflow of DT development entails (i) collecting (big) data from the physical asset (usually via IoT) and (ii) feeding this data to an AI/ML model, which will (iii) create the DT [11]. AI/ML has been used in the design, calibration and run-time phases of DT engineering in numerous domains. Typical applications range from hardware optimization to consistency judgement [12]. Current DT development approaches aim at learning the solution to one particular DT case. We propose learning the *construction process* of *classes* of DT cases instead. To enable this level of generality, we employ a specific subset of ML techniques: reinforcement learning.

Reinforcement learning. RL is a class of machine learning algorithms that learn the solution patterns of sequential decision making problems, in which there is limited feedback [13]. The learning process is driven by an *agent* that carries out *actions* in its *environment*; and subsequently processes the feedback of the environment: the new *state* of the agent, and a scalar *reward signal*. An environment in RL is formalized as a Markov Decision Process $\langle \Sigma, A, P, R \rangle$, where Σ is the set of states, A the set of actions, $P : Pr(\sigma'|\sigma, a)$ the probability of state transitions for a specific action, and R a reward function. Through interactions with the environment, the agent learns by trial and error what the best actions to perform in the different states are, to maximize the cumulative rewards. The intelligence of the agent is encoded in the policy $\pi(a|\sigma)$ of choosing action a given the exhibited state σ. Reinforcement learning is typically used in conjunction with the DT to enable learning and fine-tuning [14] the desired behavior of physical assets. Closest to our approach is the work of Tomin et al. [15], who train the simulator of an electrical power grid by RL. Our approach, however, does not aim at learning the engineering process of one specific case, but rather, a process usable in a wider class of problems.

III. Approach

Fig. 1 outlines the proposed approach.

A. Learning simulation models

We assume a data collection facility on the physical side, that is able to gather a sequence of measurements to *represent* the *Physical asset*. The implementation of this step (Step *1*) is outside of our scope. The gathered data is used to infer the simulation model of the *Digital Twin*. To do so, the *RL Agent* processes the data and aims to predict later data points with minimal error. The states of the *RL Agent* encode the various configurations of the simulation model. As the *RL Agent* explores the environment (i.e., sequentially consumes the data), it proposes newly derived configurations to the *Evaluator* (Step *2a*), and based on the ability of the configuration to predict future data points, the *Evaluator* provides the *RL Agent* with the reward signal (Step *2b*). The agent then adjusts its policy π according to the received reward. This exchange continues until the whole dataset is processed. The trained agent can then *produce* the *Digital Twins* (Step *3*). Step *3* requires an

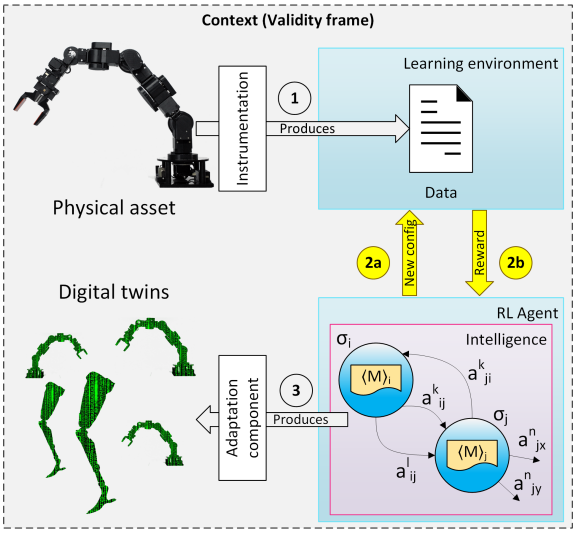

Fig. 1: Overview of the approach with the scope of our approach highlighted in *2a* and *2b*.

adaptation mechanism that is able to apply the *RL Agent*'s *Intelligence* onto the specific physical asset. By repeating this method on different physical assets, the *RL Agent* will learn how to generalize its policy to new assets, and thus, to address a *class* of problems, instead of a single case. Finding an appropriate DT model for a given physical asset can be seen as an instance of RL with a vast state space, deterministic transitions and dense rewards, where the role of generalization in the reuse between different physical assets is paramount.

To foster reuse, the representation of the physical asset has to be appropriately abstract. For example, if the agent learned its policy for a robot arm, and this intelligence is to be applied for the case of a robot leg, the mechanics of the physical asset may be almost identical, with a slight difference in pitch, which is of the opposite direction in an elbow than in a knee. The robot leg, therefore, is situated within the same *Context*. However, a drone might differ too much from the robot arm, and thus, the *Intelligence* inferred by the *RL Agent* might not be applicable, placing the drone outside the *Context*. Such contextual information should be made explicit and accessible for the *RL Agent*, so that it can factor them into the learning process, and incorporate them in the policy π. In general, the transferability of policy between physical assets ϕ_i and ϕ_j should be expressed in relation with the set of properties $\mathbb{P}$ shared by ϕ_i and ϕ_j. As a satisfactory constraint, $\forall p \in \mathbb{P} : \phi_i \vDash p \wedge \phi_j \vDash p$ must be decidable. In RL terms, this information should be encoded in the states of the agent, along with the actual simulation model configurations. We see an opportunity in reducing the state space of the agent by encoding more than one simulation model ($\langle M \rangle_i$) configuration in one RL state, if the congruence of such configurations can be expressed properly.

B. Discrete Event System Specification (DEVS)

We propose DEVS as the simulation formalism due its versatility in modeling reactive behavior, and its potential to

serve as an assembly language of simulators [6]. Additionally, DEVS allows *Atomic* DEVS models to be recursively composed into *Coupled* DEVS models, allowing arbitrarily complex hierarchies of DEVS models. An Atomic DEVS model is defined as $M = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$, where X and Y are the set of input and output events; S is the set of sequential states; $ta : S \to \mathbb{R}^+_{0,+\infty}$ is the time advance function for each state; $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ is the set of states with the time elapsed in them; $q_{init} \in Q$ is the initial state; $\delta_{int} : S \to S$ is the internal transition function to the next state according to ta; $\delta_{ext} : Q \times X \to S$ is the external transition function to the next state when an event occurs; $\lambda : S \to Y$ is the output function [4]. Atomic DEVS models are composed of architectural elements (X, Y, S, q_{init}), and dynamic elements $(\delta_{int}, \delta_{ext}, \lambda, ta)$. Architectural elements are feasible to build manually. This is because the architectural patterns of DEVS models can be associated with specific classes of real systems, enabling the reuse of the same architectural templates for different system instances. However, dynamic elements cannot be generalized across a class of systems, and their development constitutes the most labor-intensive tasks of constructing DEVS models. Here, we focus only on Atomic DEVS, as Coupled DEVS models have no dynamic elements.

C. DEVS models as RL artifacts

The mapping of DEVS onto RL concepts (Fig. 1) is defined as follows. Assuming the environment $\langle \Sigma, A, P, R \rangle$,

- $\forall \sigma \in \Sigma : \sigma \to \langle M \rangle$, i.e., every state $\sigma \in \Sigma$ encodes one class of congruent DEVS model configurations $\langle M \rangle$.
- Actions A are the elementary valid modifications of DEVS models that bring class $\langle M \rangle_i$ to $\langle M \rangle_j$, i.e., $\forall a^k_{ij} \in A, \sigma_i, \sigma_j \in \Sigma : a^k_{ij}(\sigma_i) \mapsto \sigma_j$ (Section III-D).
- The state-transition probability matrix P is defined as $\forall a \in A, \sigma_i, \sigma_j \in \Sigma : P_a(\sigma_i, \sigma_j) \in P \to Pr(\sigma_j | \sigma_i, a)$, and $Pr(\sigma_j | \sigma_i, a) \mapsto (0, 1)$.
- $R : \Sigma \to \mathbb{R}$ maps each state to a metric (Section III-E).

D. Actions

Actions are derived from Atomic DEVS operations. For brevity, we only show the *add* operations here.

Add state $(M \mid s \notin S) \mapsto (M \mid S \cup \{s\})$
Add transition $(M \mid d \notin \delta_{int}) \mapsto (M \mid \delta_{int} \cup \{d\})$, $ta(d) \mapsto \mathbb{R}^+_{0,+\infty}$
Initialization $(M \mid Q) \mapsto (M \mid Q \cup \{q_{init}\})$
Add output $(M \mid y \notin Y, \lambda : S \to Y) \mapsto (M \mid Y \cup \{y\})$
Add input $(M \mid x \notin X, \delta_{ext} : Q \times X \to S) \mapsto (M \mid X \cup \{x\})$

Additional actions for Coupled DEVS can be defined for (i) coupling, including influence relationships and I/O translation; and (ii) refinement. The learning algorithm will start from a DEVS template the modeling expert has chosen to represent the physical asset, and infer how to gradually refine the DEVS model into the appropriate levels of detail, based on the well-defined set of modeling actions. We note that the formal approach works with an empty DEVS model too (i.e., when building a model from scratch); in practical applications, however, a model template is usually provided, encoding the rough estimates of the domain expert [15].

E. Reward signal

The reward signal guides the RL Agent in exploring and eventually finding the most appropriate configuration. A lower reward is associated with less appropriate configurations, and a higher reward with more appropriate ones. The RL Agent aims to maximize the cumulative rewards of its sequence of actions, thus converging towards the solution. The reward signal is provided by the environment, and is generated by the reward function R. We envision at least two characteristically different classes of reward functions. First, the RL Agent should be guided towards learning to solve the initial case. Second, to improve generizability, the RL Agent should be guided towards learning to simulate the common properties of the class of physical assets of interest. Typical properties of interest are non-functional properties (NFP), such as safety, reliability, and performance. However, NFPs in DTs exhibit more complex semantics than in software. For example, safety might be described as a function of the inertia matrix of the physical asset [16]. The relevant aspects and properties of the physical asset have to be explicitly modeled using the most appropriate formalisms, and at varying levels of abstraction. The reward functions can then be combined in any meaningful algebraic way, e.g., by taking their weighted average $R = \frac{\Sigma_{i=1}^n w_i R_i}{\Sigma_{i=1}^n w_i}$. Flexibility in adapting the weights can be introduced to allow adapting the RL Agent to the different phases of the learning process. For example, at the beginning, learning for descriptive power might not be as important as learning for predictive power; while in the later phases the importance of these factors might be different.

IV. Challenges and opportunities

The approach outlined in this paper poses complex challenges to overcome, but offers many opportunities. In this section, we describe the ones we have already identified.

A. Challenges

One of the main challenges to overcome is the **generalizability of DT settings**. Our approach promises tackling this challenge by generalizing the simulation formalism, but in-depth investigation of the simulation models and the available RL frameworks is required to appropriately determine the practical feasibility. As discussed, generalizing the decision-making policy necessitates to include contextual information in the *RL Agent*'s states. The main challenge is formulating the representation of contexts while keeping the state-space as small as possible. **Appropriately capturing the reward signal** might become problematic when learning for better descriptive power. Another challenge will be to design the reward function to **efficiently guide the agent** towards good solutions. In general, ML requires **high volumes of data** to achieve adequate performance. In our context, the lack of openly available data might be an obstacle. Finally, the mechanism of **adapting/applying the inferred intelligence** to a specific instance might pose software engineering challenges.

B. Opportunities

One of the main directions to explore is the ability of **validity frames** [17] to capture the contextual information of the physical asset and its environment. By that, the conditions under which the policy π is transferable to other problems, could be expressed in a formal way. We see opportunities in **incorporating human actors** for numerous purposes. As discussed, **model templates** provided by domain experts increase the performance of the approach. While humans perform poorly in specifying optimal models, they perform considerably well in specifying reasonable ones [18]. Further extending the scope of human involvement, domain experts could be employed in the **oversight of the learning process**, or allowing the domain expert to take over the modeling tasks in specific cases, and allow the RL Agent to learn from the human. Despite the high costs of human reward function in naive RL approaches, **learning from the human** has been shown to be feasible in numerous settings [19], [20]. Alternative techniques, such as **learning from demonstration** [21] and **active learning** [22] should be considered to augment the approach with. Once the RL Agents become experienced enough with different instances and flavors of the same class of DT, new **agents can be trained through transfer learning** [23], i.e., by learning from the experience of older agents. Such settings would allow removing the human from the loop and replace the manually assembled default policies with the ones the agents themselves inferred from previous cases. To enable better and more targeted harvesting of data on the physical asset, we anticipate research directions on the **co-design of the instrumentation of the physical asset with twin models**. That is, while the RL Agent learns the optimal simulation model, it also provides hints on policies that are not attainable with the current instrumentation of the physical asset, but could be attained given a reasonable reconfiguration of the physical side.

V. Conclusion

In this paper, we outlined an approach for the inference of simulation models in digital twins by reinforcement learning. We have identified the lack of generizability being a potentially blocking problem in the automated construction of DT. Our proposed approach tackles this issue by generalizing the simulation formalism. This attempt comes at the cost of vastly increased essential complexity of the simulator. The resulting setting, however, enables applying reinforcement learning to automate the process. We plan to build a prototype using PythonPDEVS[1], an openly available implementation of the Parallel DEVS. For the reinforcement learning component, we will use a well-established deep reinforcement learning library, e.g. Tensorforce[2] or Stable Baselines3[3].

[1]http://msdl.cs.mcgill.ca/projects/DEVS/PythonPDEVS

[2]https://github.com/tensorforce/tensorforce

[3]https://github.com/DLR-RM/stable-baselines3

References

[1] A. Rasheed, O. San, and T. Kvamsdal, "Digital twin: Values, challenges and enablers from a modeling perspective," *IEEE Access*, vol. 8, pp. 21 980–22 012, 2020.

[2] S. Boschert and R. Rosen, "Digital twin—the simulation aspect," in *Mechatronic futures*. Springer, 2016, pp. 59–74.

[3] F. Bordeleau, B. Combemale, R. Eramo, M. van den Brand, and M. Wimmer, "Towards Model-Driven Digital Twin Engineering: Current Opportunities and Future Challenges," in *International Conference on Systems Modelling and Management*. Springer, 2020, pp. 43–54.

[4] B. P. Zeigler, A. Muzy, and E. Kofman, *Theory of modeling and simulation: discrete event & iterative system computational foundations*. Academic press, 2018.

[5] S. Mittal *et al.*, "Digital twin modeling, co-simulation and cyber use-case inclusion methodology for iot systems," in *Winter Simulation Conference*. IEEE, 2019, pp. 2653–2664.

[6] H. Vangheluwe, "DEVS as a common denominator for multi-formalism hybrid systems modelling," in *Symposium on computer-aided control system design*. IEEE, 2000, pp. 129–134.

[7] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–1022, 2018.

[8] H. Kaylani and A. M. Atieh, "Simulation approach to enhance production scheduling procedures at a pharmaceutical company with large product mix," *Procedia Cirp*, vol. 41, pp. 411–416, 2016.

[9] A. Karakra *et al.*, "Pervasive computing integrated discrete event simulation for a hospital digital twin," in *Computer Systems and Applications*. IEEE, 2018, pp. 1–6.

[10] "GE Digital Twin. Analytic Engine for the Digital Power Plant." https://www.ge.com/digital/sites/default/files/download_assets/Digital-Twin-for-the-digital-power-plant-.pdf, Retrieved: 03/07/2021.

[11] M. M. Rathore, S. A. Shah, D. Shukla, E. Bentafat, and S. Bakiras, "The role of ai, machine learning, and big data in digital twinning: A systematic literature review, challenges, and opportunities," *IEEE Access*, vol. 9, pp. 32 030–32 052, 2021.

[12] K. Fischer and M. Heintel, "Examining a consistency between reference data of a production object and data of a digital twin of the production object," May 4 2021, US Patent 10,999,293.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[14] C. Cronrath, A. R. Aderiani, and B. Lennartson, "Enhancing digital twins through reinforcement learning," in *Automation Science and Engineering*. IEEE, 2019, pp. 293–298.

[15] N. Tomin, V. Kurbatsky, V. Borisov, and S. Musalev, "Development of digital twin for load center on the example of distribution network of an urban district," in *E3S Web of Conferences*, vol. 209, 2020.

[16] I. Dávid, H. Vangheluwe, and Y. Van Tendeloo, "Translating engineering workflow models to devs for performance evaluation," in *Winter Simulation Conference*. IEEE, 2018, pp. 616–627.

[17] S. Van Mierlo, B. J. Oakes, B. Van Acker, R. Eslampanah, J. Denil, and H. Vangheluwe, "Exploring Validity Frames in Practice," in *Systems Modelling and Management*. Springer, 2020, pp. 131–148.

[18] M. A. Wiering and M. Van Otterlo, "Reinforcement learning," *Adaptation, learning, and optimization*, vol. 12, no. 3, 2012.

[19] W. B. Knox and P. Stone, "Reinforcement learning from simultaneous human and MDP reward." in *AAMAS*, 2012, pp. 475–482.

[20] R. Loftin, B. Peng, J. MacGlashan, M. L. Littman, M. E. Taylor, J. Huang, and D. L. Roberts, "Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning," *Autonomous agents and multi-agent systems*, vol. 30, no. 1, pp. 30–59, 2016.

[21] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[22] B. Settles, "Active learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 6, no. 1, pp. 1–114, 2012.

[23] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.