Architecture for Digital Twin-based Reinforcement Learning Optimization of Cyber-Physical Systems*

Elias Modrakowski[1][0009−0006−0305−7848], Niklas Braun[2], Mehrnoush Hajnorouzi[1][0009−0007−1656−5670], Dr. Andreas Eich[3][0000−0002−0828−9385], Narges Javaheri[4], Dr. ir. Richard Doornbos[5], Sebastian Moritz[6][0009−0006−0344−6253], Jan-Willem Bikker[7], and Rutger van Beek[7]

[1] Institute of Systems Engineering for Future Mobility, German Aerospace Center (DLR e.V.), Oldenburg, Germany
elias.modrakowski@dlr.de
[2] AVL Deutschland GmbH, Karlsruhe, Germany
[3] Liangdao GmbH, Munich, Germany
[4] Thermo Fisher Scientific, Eindhoven, The Netherlands
[5] TNO-ESI, Eindhoven, The Netherlands
[6] TrianGraphics, Berlin, Germany
[7] CQM, Eindhoven, The Netherlands

Abstract. The optimization of complex cyber-physical systems is a crucial task for their correct functioning, usability, and commercial viability. Due to their complexity, scale and resource intensiveness, conventional manual optimization is infeasible in many instances. We investigate the combination of the Digital Twin paradigm and Reinforcement Learning framework to address the long response times, limited availability of data, and the intractability of such systems. Here, the Digital Twin functions as the training environment in different development phases of the optimization. In this position paper we showcase our ongoing research on developing a reference architecture of a Digital Twin-Artificial Intelligence optimization system. This includes presenting the development process of the optimization system in terms of phases, an architecture from four viewpoints and an exemplary implementation.

Keywords: Digital Twin · Reinforcement Learning · System Optimization · Cyber-Physical System · Reference Architecture · System Architecture · Training Data Generation.

1 Introduction

Complex high-tech Cyber-Physical Systems (CPSs) often have an unattainable scale (e.g., number of controls, settings, and usage scenarios) with limited availability for tests combined with high costs. The optimization in terms of adjustment of their configuration for proper functionality, usability, and commercial viability is required. However, achieving optimization objectives, due to indirect and long-term materializing optimization goals or complex high-dimensional system states, usually is challenging for conventional automated methods. Consequently, manual intervention becomes necessary which demands experts to spend valuable time and resources.

Typically, this process is slow, costly, and inexperienced personnel may inadvertently place the CPS in a non-optimal state, or disrupt the system. One solution to these problems is to determine optimal system settings by the use of Artificial Intelligence (AI), or more precisely, Reinforcement Learning (RL) techniques [31] which would yield these settings from system-generated data [7]. However, RL is known for demanding large amounts of training data making it incapable of being parameterized using the high-tech CPSs directly and therefore models and simulations are necessary for data generation [18].

To illustrate the challenge, let us assume a robot arm in a factory with a PID-controller regulating one of the actuators. In this case, the RL optimizes the CPS, i.e., the robot arm, by adjusting controller's parameters, guided by higher level objectives such as precise positioning of the robot arm's attachment or minimizing the failure rates on factory level. A human operator would need to adjust these parameters for every robot arm commissioned (as they all differ slightly from each other) and readjust them over time (aging of the machinery, changing of the attachment or manufacturing product, etc.). New controller settings may be suggested by the optimization AI and tested in simulation to minimize the downtime of the robot arm. While this example is simple, it showcases the potential of the idea as these optimizations may be done in an automated way.

In this position paper, we propose an approach of optimizing CPSs that incorporates two technologies; (i) Digital Twin (DT) to mitigate long response times and limited availability of CPS; (ii) RL to address complex systems optimization. The expected result is a setup that performs the optimization of a CPS quicker, safer, and less expensive compared to a human. More concretely, we address the question: How does an architecture look like that leverages RL training data generation using DT? This research is unique by outlining the key factors on the DT side necessary for RL training throughout a CPS's life cycle in form of a reference architecture. It is directed to system developers searching for a solution to slow, manual and expensive optimization of such systems. Subsequently, we introduce a DT-RL optimization framework, a reference architecture supporting its development, outline the application to an exemplary use case, and conclude by outlining the shortcomings resulting into future avenues of work.

2 Prerequisites and Related Work

2.1 Combining DTs and RL

Fig. 1 depicts the overall interactions during the DT-based optimization-RL life cycle that shall be enables by the architecture. First, DT of a typical system is modelled with the capability of representing variations of the system. It represents the environment with which the optimization-RL Agent (RLA) is then trained. In continuation, the optimization-RL is applied to actual system(s) where further data can be collected for fine-tuning purposes. Consequently, we derived three iterative phases:

Training phase In this phase, the RLA is trained using multiple varying instances of a typical CPS. This is done to prevent the problem of overfitting and to increase generalization to enable transfer learning (see [7,25]).

Operational phase Once the RLA has matured enough, it is deployed on an actual CPS to perform system optimization. Here, the RLA is assumed to interact directly with the CPS or is part of the CPS. During this period, data from the CPS is gathered for the next phase.

Fine-tuning phase Periodically, the RLA is fine-tuned using the CPS's data from the operational phase to improve and specialize the learned policy w.r.t. the CPS instance by twinning the related real system. In addition, the variant agnostic RLA (result of the training phase) is also further trained, e.g., by using federated learning [20]. An extensive testing of the RLA adjustment is necessary, yet the frequency of the fine-tuning phase depends on the use case.

2.2 Fundamental considerations

A DT is defined as a virtual duplicate of a system built from a fusion of models and data [32]. The essential role of DT is to represent the behavior of real system

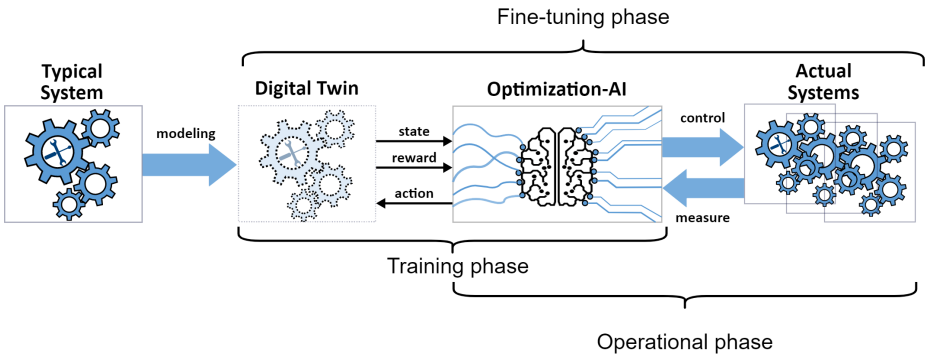


Fig. 1: Overview of the development process of DT supported RL training.

by periodically adapting to the actual system. This differentiates a DT from a digital model [5,12,29,32]. We adopt the notion of different integration levels by [2]. Different phases of system optimization apply varying degrees of integration, i.e., digital model up to DT. While a digital model of a generic CPS is sufficient for the training phase, the operational and fine-tuning acquire higher levels of integration, to the degree of twinning [17]. It is important to note that in this context, the DT encompasses the CPS itself and its surrounding environment and how it imposes influence on the CPS. With respect to the robot arm example, this would be the room temperature or properties of the manufacturing product the arm should process.

RL [31] is broadly applied in various disciplines for the purpose of system optimization or optimized control, e.g., [8,13,21,33]. RL is well-suited to the problems that include learning from unlabeled data, e.g., due to long-term materialization of optimality [13].

Several challenges arise from the proposed DT-RL approach, aiming to establish simulation-based training capabilities and enable a continual enhancement of optimization RL algorithm exploiting the advantages of DTs. It is crucial to meet RL-specific demands, namely result accuracy, efficient execution, standardized simulation input/output (state, action, reward) and managing variations for policy transfer into the real world. An additional focus lies in facilitating the model re-usability throughout its life-cycle.

2.3 Literature on DT-based RL

Recent research has delved into combination of these challenges across diverse domains. [25] introduces a RL training approach through simulations. The domain of application in this case are driving functions for automated vehicles. Nevertheless, it lacks an architectural view and continuous RLA refining. [8] emphasizes training RLA for fusion reactor control with architectural aspects taking a backseat in training consideration. [3] adopts a DT strategy for robot arm control including architectural specifications, akin to our proposal, but overlooks the post-initial learning. [18] and [11] provide insights into real-world transfer of RL policies and DT-based RL scheduling strategy training, respectively. However, architectural and, yet again, post-deployment learning aspects are under-explored. Similarly, [27] advocates DT and RL fusion for unmanned aerial vehicle control and proposes a DT-based "continuous evolution" in this context [28]. However, the work is domain-specific and generalizable architecture considerations are not present in this work.

In the field of "life-long" RL [19], scholars deal with continuous learning and policy transfer (e.g., from simulation to real world or adjustments from one system to the next). To the authors' best knowledge, availability of training data is assumed in most of the relevant studies. In context of continuous learning, Federated Learning (FL) [16] is a common practice for pre-training and consequent fine-tuning of AI. [30] showcases the combination of DT and FL but leaves the design of these DTs open for interpretation. In general, application independent

DT architectures are rare as a high percentage of solution-oriented architectures [10] show that every application context requires their own interpretation.

From the cited research, it is evident that the comprehensive exploitation of the DT paradigm's capabilities has not yet been fully realized in RL training. In all of the aforementioned literature, RL is utilized for system control, wherein we see system optimization a sub-field. Our objective is to delineate the essential considerations required on the DT side for RL training across the various stages of its life-cycle. We aim to provide an architectural recommendation that goes beyond mere building block descriptions.

3 Architecture

A multi-sided description is required to plan an architecture that considers a multitude of aspects. Among various architectural view models like [14], we have adopted four viewpoints similar as in SPES (Software Platform Embedded Systems) [26]. In spite of minimizing the number of viewpoints, namely requirements, functional, logical and technical views, they are capable of covering all relevant layers and aspects [6].

3.1 Requirements View

The main objective of our approach is to realize a self-optimizing solution for complex CPSs which can be then applied to analogous systems without necessitating extensive re-development, e.g., re-modelling of interfaces, or re-training the AI from scratch. The specific requirements of our approach mostly derived from the RL requirements on the training and fine-tuning data. All requirements mentioned in this view are on the system. However, the fulfillment of some requirements is facilitated by the architecture (see Table 1).

The functional requirements of the DT in general include (i) to model the components that contribute to the observable outputs / behavior sufficiently, (ii) to model the potential influences of the CPS's environment, and (iii) to possess parameters that determine its behavior, analogous to the corresponding physical counterpart.

Moreover, in the context of RL-training, the set of requirements of DT as the training data generator model, need to be expanded by minimized calculation time (time to result) and satisfactory data accuracy. While the previous comes naturally, defining "sufficient" accuracy is not straight forward. As a general principle, more information with higher quality results in improved decision making. However, machine learning algorithms can bridge the gap and transfer whatever they learned from a less accurate model to the actual operating system (see [9,18]). As an example, the autonomous driving functions trained by simulated data, can be applied successfully to the real-world use cases [25]. This relaxes the constraints on DT's requirements and improves execution efficiency on the way. Therefore, the required accuracy level of DT within the proposed

approach is highly dependent on the AI capabilities. As long as the RL model can bridge the gap, the accuracy of the DT is sufficient.

To effectively train the RLA, it is important to mitigate the impacts of noise and irrelevant information during training phase. Overfitting on these effects can hinder the trained RLA's adaptability and resilience to slightly changed circumstances comparing to the training data [4]. To address this issue, one approach is to train the RLA by higher-level features of the data, specifically by "averaging out" system-specific behavior. To this end, we propose to use a fleet of digital models or DTs representing the CPSs. These models should encompass multiple variants, each incorporating slight differences to represent the variations in real-world. The variation can be introduced by altering virtual environment effects and configuration changes within a predefined valid parameter space. This approach allows the RLA to learn from a spectrum of possible CPS variants instead of only from one specific instances. By training on a diverse set, the RLA is enabled to perform across multiple systems without requiring re-training [7].

The non-functional requirements can be summarized in footprint (minimization of resource usage), integrability (the ability to integrate heterogeneous systems and IP-protection), re-usability, and maintainability.

Table 1: Overview of key requirements and considerations at a high level, assessing their reflection by the architecture.

No.	Requirement	Description	Reflected in architecture
1	Execution efficiency	Calculation time needs to be minimized to make training time manageable.	No
2	Accuracy of result	The system must accurately enough simulate real-world behavior for successful RL-driven optimization in the operational phase.	No
3	Variants representation	The DTs must be able to represent real or virtual variants of a system w.r.t. to its environment and configuration.	Yes
4	Twinning	The system must be able to retrieve data from its physical twin and parametrize itself accordingly	Yes
5	Footprint	The resource usage for training and fine-tuning shall be minimized.	No
6	Re-Usability	To reduce resource and time consumption, developed models and infrastructure shall be reusable.	Yes
7	Maintainability	The system shall be easily maintainable.	Yes
8	Integrability	The integration of heterogeneous models and IP-protection shall be facilitated.	Yes

3.2 Functional View

The functional view describes the specification of solution-independent functional system, i.e., it describes the functionality that the system intended to provide in order to fulfill its purpose. Fig. 2 shows the functional model of optimization using data from DT and CPS during the training phase and operational phase, respectively. As mentioned earlier, the model shall be used interchangeably for different phases without requiring major adjustments to the architecture (requirement No. 6 in Table 1). Initially, operational goals and the twinning information are given to the system which initializes and "controls" the optimization by executing a scenario. An initial batch of parameters containing the parameters that shall be optimized as well as the parameters that define the model variants (fulfilling requirement No. 3 in Table 1) are given to an entity that performs the optimization. This might be the actual CPS or its DT. Optimization metrics and system defining information (e.g., previous parameter configuration) in form of state and reward according to the RL paradigm shall be collected one hand and also analyzed. Based on this and possibly the history of prior states, an improved configuration is proposed and returned to be executed again. It can be seen that the four major functionalities needed can be assigned to four main sub-systems: A control system, a CPS/DT within its environment, a data storage platform and an AI system.

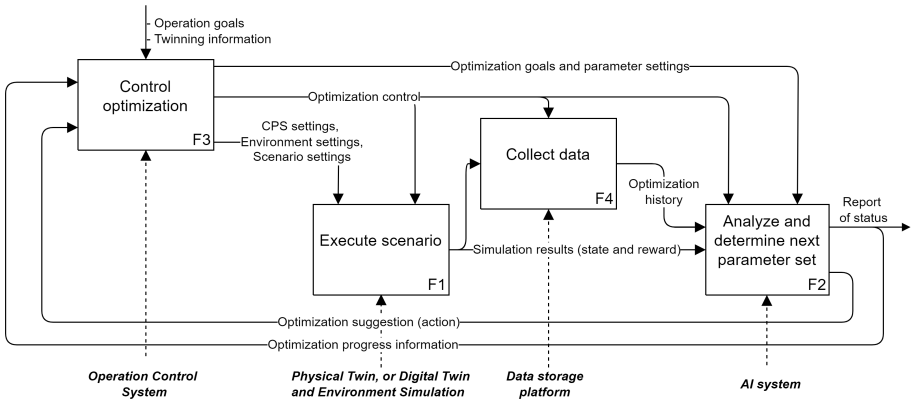


Fig. 2: Generic functional model of the system's optimization in form of IDEF0 (Integration Definition for Process Modelling).

Regarding functional capabilities of the DT as represented in Fig. 3, it must be able to control (Control simulation) and represent and execute simulation of system features (Simulate features) to mimic the CPS's behavior accurate enough and output relevant simulation results for further training. The features must be configurable to enable adaptation to a real or synthetic CPS variant.

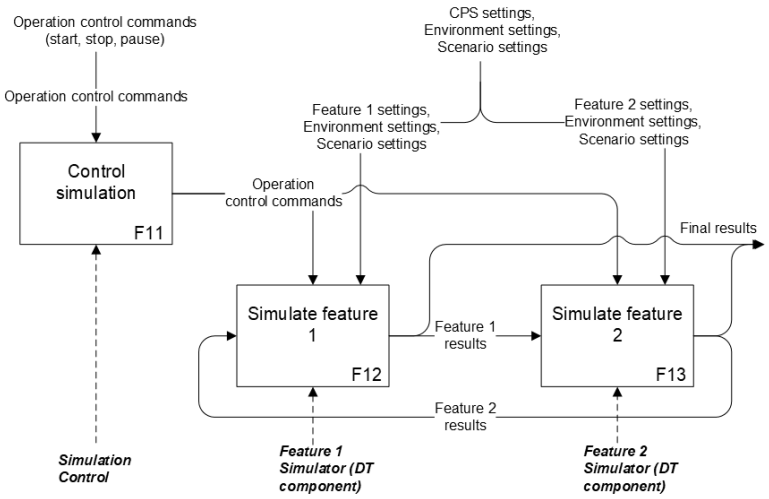


Fig. 3: Generic functional model of DT in form of IDEF0, exemplary with two simulation features.

3.3 Logical View

The logical view describes the design solution, providing functional building blocks by communicating components [26]. Due to the already stated variety of RL training algorithms, we showcase the simulation environment and the contained DT only. The simulation environment is not meant the context in which the CPS is located in, but rather the digital infrastructure for performing simulations. Both modules' structure is expressed in accordance with the SysML language as internal block diagrams (ibd) in the following. The simulation environment in Fig. 4 consists of a flow of data from the action port via a preprocessing component into the DT as a simulation input, and after computing from the DT's output via a post-processing component to the state and reward output of the simulation environment. The pre- and post-processing components serve the goal of conversion of the RLA/DT output to an adequate format. A controller manages the processing and optional databases providing additional information. In case of the operational and fine-tuning phase, a twinning block is implemented to orchestrate the retrieval of CPS data as necessary according to the requirements (requirement No. 4 in Table 1). During the training phase, as no physical counterpart exists, it is replaced by a variation generator within the DT mimicking the existence of multiple, slightly different CPS instances. The simulation environment interacts through five ports: (i) control inputs from controller that manages RL training, (ii) connection to the CPS and (iii) action, (iv) reward and (v) state ports for the communication with RLA.

The DT represented in Fig. 5, consists of none or more submodule-DTs and one or more logic block(s). The latter calculates the behavioral output of the DT

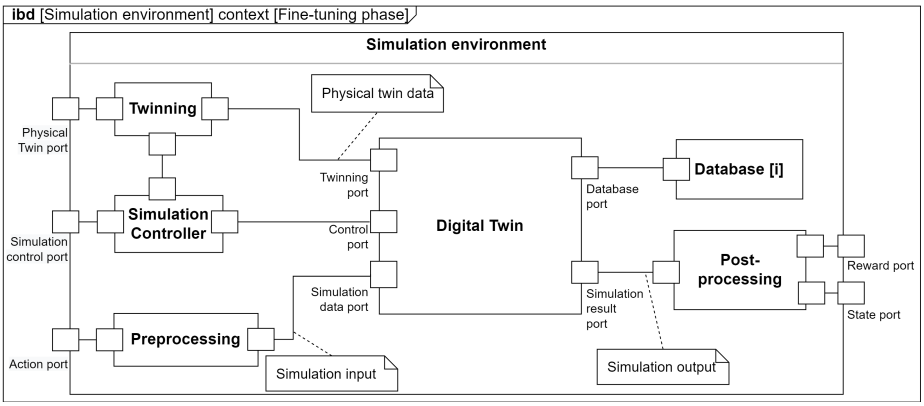


Fig. 4: Logical components and communication of the simulation environment.

by using (i) simulation data input to the DT itself, (ii) and/ or simulation data output of hierarchically nested sub-DTs, (iii) static configuration parameters sourced from persistent databases, and (iv) dynamic configuration parameters, obtained through the adaptation model.

Adaptation to changing behavior of the CPS is handled by the adaptation model. It identifies any emerging differences DT and CPS data comparison and compensate them via providing a set of dynamic configuration parameters to the DT logic block. Hierarchically nested sub-DTs get their CPS output data via a twinning port, to provide adaptation on multiple levels of the DT. It is impor-

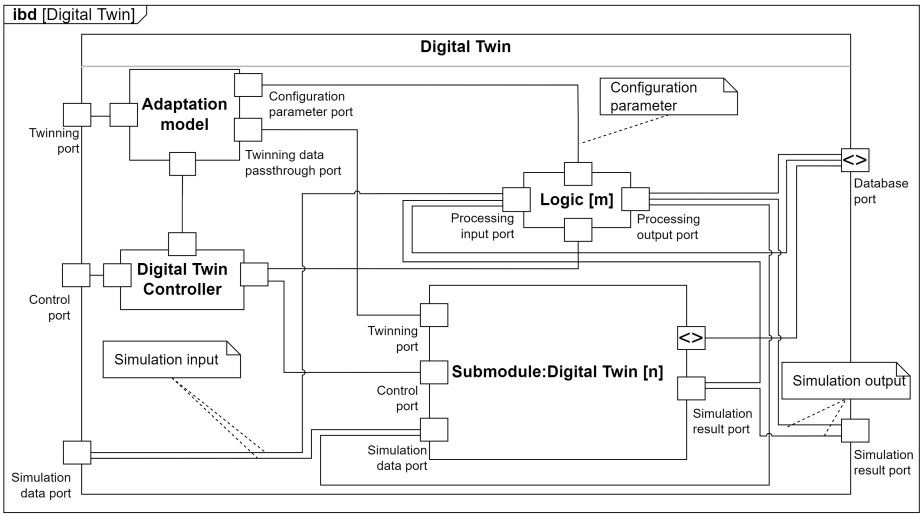
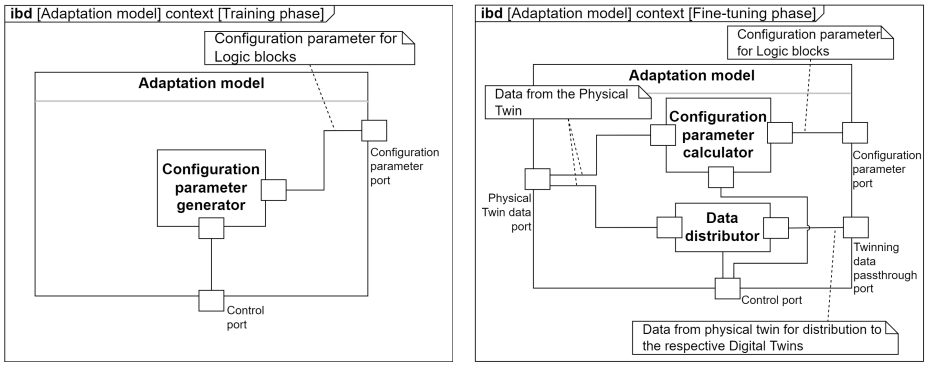


Fig. 5: Logical components and communication of the Digital Twin.

tant to define the configuration parameters that can (and cannot) be changed by every (sub-)DT to leverage the explainability and traceability provided by the architecture. The digital twin controller is used to control the adaptation model and the pass-through to sub-DTs. The logic of the adaptation model is hereby dependent on the current phase while the remainder of the architecture persists. During the training phase, the DT is used to create RL training data in a controlled manner for specified variants of the CPS but without synchronization to a CPS. Creating multiple instances of an adapted static DT representing a CPS variant for parallel RL training is possible. Slight derivatives of the adapted DT are possible to generate, via a configuration parameter generator. Generation of these parameters can be sampled from parameter distributions. These parameter distributions can either be estimated or measured based on one or more CPS that have already been synchronized during the fine-tuning phase. This operating mode is depicted in Fig. 6a. The fine-tuning phase sees a connected CPS, leading to the ability to synchronize behavior of the DT again. The main logical components can be seen in Fig. 6b. Twinning data from the CPS is used for calculating respective configuration parameters. This is in contrast to the training phase, where configuration parameters are generated or sourced from a parameter distribution. To realize the hierarchical DT structure, data pass-through is also handled inside the adaptation model via the data distributor.

In context of the robot-arm example, during the training phase parameter distributions are initially estimated e.g., for the room temperature, robot arm attachments properties, and manufacturing product's mass while the mechanical and electronic characteristics of the arm itself would typically be quite invariant. During the fine-tuning phase not only the optimization-RL can be improved with the DT twinned with data specific robot arm, but also the above mentioned parameter distributions can be adjusted with data from multiple robots arms. The latter is relevant for the training phase of a next iteration of the generic RLA.



(a) During training phase **(b)** During fine-tuning phase

Fig. 6: Logical components of the adaptation model.

3.4 Technical View

The concrete technical solution for the combination of DT and RL mainly depends on the specific demands of the use case. However, the integrability, reusability and maintainability are the biggest drivers for a standardized solution on a technical realization level as well as execution-efficiency (requirements No. 2, 6, 7 and 8 in Table 1).

On the DT level (see Fig. 5), co-simulation is an ideal approach for simulating DTs of multiple system components to provide the possibility of creating a modular architecture where all models interact with a central control unit. Examples are HLA [15], FMI [23] or proprietary software solutions such as Model.CONNECT[8]. They can enable the execution of simulations and calculations in a distributed computing manner using cloud-solutions and multiple on-site workstations. A benefit is increased execution speed of training data sampling which is, as already stated, a major requirement.

On a higher level, the development and integration of the architecture benefits from containerization with tools such as Docker[9]. It enables seamless and consistent deployment of the components across diverse environments by encapsulating them along with their dependencies. As the communication between the containers is via network and well defined thanks to the architecture, maintainability and scalability (multiple simulation environments for parallel sampling) is non-issue, and distributed computing between multiple workstations or in the cloud is possible.

With respect to the linkage between the DTs and their physical counterpart, infrastructure is needed that enables reliable and straightforward communication between the DT and the CPS. It is needed for (I) retrieval of physical twin data and (II) execution of optimization configurations. The level of sophistication of the applied framework depends on the use case characteristics. For this, an example is Basys4[10] where the communication between the physical asset and digital applications is managed by an Asset Administration Shell (AAS) which provides standardized communication interfaces (HTTP/REST interface) via network to applications. This makes a cloud-based deployment of the optimization RLA non-complex. In context of the presented architecture, the AAS represents the "Twinning" block (see Fig. 4) while the simulation environment can be seen as an application to the Basys4 framework. The simulation controller (see Fig. 7) can request from the AAS/Twinning block (which is connected to the physical system via a sub model) to send data to the twinning port of the DT. In principle, a profound integration of the Basys4 framework into the proposed approach is possible when considering simulation models including their sub-modules as the DT. However, this potential integration is not reflected in the given architecture.

[8] https://www.avl.com/en/simulation-solutions/software-offering/simulation-tools-z/modelconnect

[9] https://www.docker.com/

[10] https://www.basys40.de/ and https://github.com/eclipse-basyx

4 Exemplary Application

The architecture is developed in conjunction with a use case from the highly-automated vehicle testing domain that aims for automatic creation of a 3D-environment for vehicle tests. The system to be optimized is the testbed including a virtual environment in conjunction with a vehicle. For development of automated vehicle driving functions, critical scenarios are to be optimized to ensure testing time is used for the most insightful tests in order to find e.g., weaknesses in the functions. Also, this approach could be used as open-exploration to build a database of scenarios. A scenario is usually characterized by specific criticality metrics that indicate its significance and potential impacts [24]. The RLA proposes concrete scenarios with concrete parametrization/state values from logical scenarios with parameter ranges of state values (see [22]) which are executed on the virtual or real testbed, and the results are then computed to evaluate the scenarios.

Fig. 7 showcases the logical view on the optimization system during the fine-tuning phase. Thanks to the reference architecture enabling hierarchical DTs, the DT of the vehicle test setup contains a virtual environment simulating e.g., the street as a logic block, and the DT of a vehicle enables a black-box integration of this sub-module. The vehicle DT can also contain other DTs as submodules e.g., of sensors or the chassis. A database is integrated to hold libraries of the virtual environment's assets. The simulation environment is connected to a vehicle its behavior it shall mimic. While in Fig. 7 the fine-tuning phase is depicted, the only difference to the training phase is the substitution of the adaptation model by a parameter generator as seen in Fig. 6. This results in the minimization of necessary adjustments to the system including data formats as the variation generator and the twinning function output data in identical format.

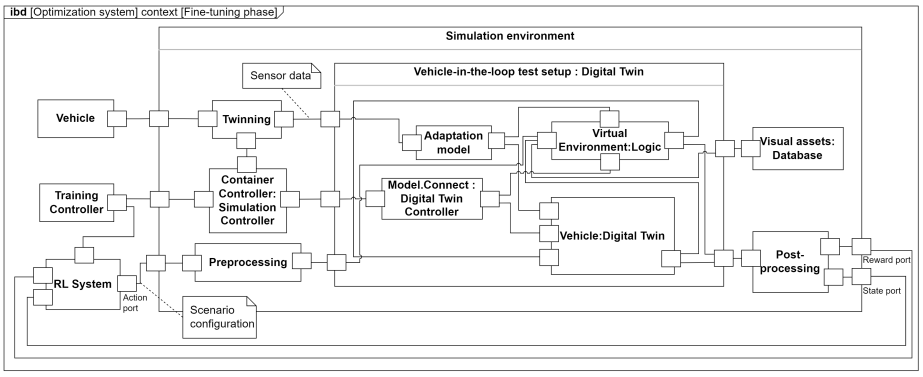


Fig. 7: Exemplary application of the architecture to a vehicle-in-the-loop test setup during the operational and fine-tuning phase wherein scenarios are optimized and additional data are gathered for further improvement of RLA.

On a technical level, a Docker-based architecture is used for continuous development and prototyping. As a DT's co-simulation framework, Model.CONNECT was selected due to its features designed for automotive domain. As this example is a prototypical implementation, simple Python scripts are used to enable the function of the twinning and adaptation model. The implemented RL algorithm ("RL System" block in Fig. 7) is Maximum a Posteriori Policy Optimization (MPO) [1] inspired by [8]: an off-policy actor-critic approach.

Although a comprehensive assessment of the architecture's effectiveness and efficiency in facilitating the creation of a system capable of training and fine-tuning an optimization-RLA is ongoing, initial findings indicate favorable outcomes in terms of maintainability and reusability (see Table 1). The development process has been notably streamlined, for the incorporation of new features proving to be a straightforward task. However, a crucial aspect of the architecture and the system yet to be validated pertains to the representation capabilities of the variants, including the assessment of twinning.

5 Conclusion and Future Work

This position paper presents our work-in-progress regarding a reference architecture for DT-based AI-training for CPS optimization. In four viewpoints we highlighted the necessary considerations and requirements on the simulation environment including the DT as the virtual representation of the CPS, a model view of its functionality, a design-solution and technical considerations for the implementation. The differences of the architecture over the life-cycle of RL optimization (training-, operational- and fine-tuning phase) are highlighted and exemplary implementation in an automotive use case is shown.

In the future we will focus on improving the level of detail putting a strong focus on the differences between phases. The current state lacks detail on the requirements on data type and considerations of the twinning and variation generator. Further validation and integration are necessary in the above-mentioned use case to evaluate the improvement of the approach beyond the state of the art and verification of the broader applicability by utilizing it for the development of an alternative use case is needed. In addition, we will increase the research on behavioral modelling and the impacts on the architecture.

Acknowledgments. The research is carried out as part of the ITEA4 20216 ASIMOV project. The ASIMOV activities are supported by the Netherlands Organization for Applied Scientific Research TNO and the Dutch Ministry of Economic Affairs and Climate (project number: AI211006). This research is also partially funded by the German Federal Ministry of Education and Research (BMBF) within the project ASIMOV-D under grant agreement No. 01IS21022B [AVL], 01IS21022D [Liangdao], 01IS21022F [TrianGraphics] and 01IS21022G [DLR] based on a decision of the German Bundestag.

References

1. Abdolmaleki, A., Springenberg, J.T., Tassa, Y., Munos, R., Heess, N., Riedmiller, M.: Maximum a posteriori policy optimisation (2018). https://doi.org/10.48550/arXiv.1806.06920
2. Aheleroff, S., Xu, X., Zhong, R.Y., Lu, Y.: Digital twin as a service (dtaas) in industry 4.0: An architecture reference model. Advanced Engineering Informatics **47**, 101225 (2021). https://doi.org/10.1016/j.aei.2020.101225
3. Alexopoulos, K., Nikolakis, N., Chryssolouris, G.: Digital twin-driven supervised machine learning for the development of artificial intelligence applications in manufacturing. Int. J. Comput. Integr. Manuf. **33**(5), 429–439 (2020). https://doi.org/10.1080/0951192X.2020.1747642
4. Amiranashvili, A., Argus, M., Hermann, L., Burgard, W., Brox, T.: Pre-training of deep rl agents for improved learning under domain randomization. eprint arXiv:2104.14386 (2021). https://doi.org/10.48550/arXiv.2104.14386
5. Barricelli, B.R., Casiraghi, E., Fogli, D.: A survey on digital twin: Definitions, characteristics, applications, and design implications. IEEE Access **7**, 167653–167671 (2019). https://doi.org/10.1109/ACCESS.2019.2953499
6. Brankovic, B., Binder, C., Draxler, D., Neureiter, C., Lastro, G.: Towards a cross-domain modeling approach in system-of-systems architectures. In: Boy, G.A., Guegan, A., Krob, D., Vion, V. (eds.) Complex Systems Design & Management. pp. 164–175. Springer (2019). https://doi.org/10.1007/978-3-030-34843-4{_}14
7. Cobbe, K., Klimov, O., Hesse, C., Kim, T., Schulman, J.: Quantifying generalization in reinforcement learning. In: 36th International Conference on Machine Learning, vol. PMLR 97. PMLR, Long Beach, USA (2019)
8. Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de las Casas, D., Donner, C., Fritz, L., Galperti, C., Huber, A., Keeling, J., Tsimpoukelli, M., Kay, J., Merle, A., Moret, J.M., Noury, S., Pesamosca, F., Pfau, D., Sauter, O., Sommariva, C., Coda, S., Duval, B., Fasoli, A., Kohli, P., Kavukcuoglu, K., Hassabis, D., Riedmiller, M.: Magnetic control of tokamak plasmas through deep reinforcement learning. Nature **602**(7897), 414–419 (2022). https://doi.org/10.1038/s41586-021-04301-9
9. Dulac-Arnold, G., Levine, N., Mankowitz, D.J., Li, J., Paduraru, C., Gowal, S., Hester, T.: Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Machine Learning **110**(9), 2419–2468 (2021)
10. Ferko, E., Bucaioni, A., Behnam, M.: Architecting digital twins. IEEE Access **10**, 50335–50350 (2022). https://doi.org/10.1109/ACCESS.2022.3172964
11. Gan, X., Zuo, Y., Zhang, A., Li, S., Tao, F.: Digital twin-enabled adaptive scheduling strategy based on deep reinforcement learning. Science China Technological Sciences pp. 1–15 (2023)
12. Grieves, M., Vickers, J., (None): Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In: Kahlen, F.J., Flumerfelt, S., Alves, A. (eds.) Transdisciplinary Perspectives on Complex Systems, pp. 85–113. Springer (2016). https://doi.org/10.1007/978-3-319-38756-7
13. Haj-Ali, A., Ahmed, N.K., Willke, T., Gonzalez, J., Asanovic, K., Stoica, I.: A view on deep reinforcement learning in system optimization
14. Hankel, M., Rexroth, B.: Das Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0)
15. IEEE: Ieee standard for modeling and simulation (m&s) high level architecture (hla): Framework and rules (2010), https://standards.ieee.org/ieee/1516/3744/

16. Jamil, S., Rahman, M., Fawad: A comprehensive survey of digital twins and federated learning for industrial internet of things (iiot), internet of vehicles (iov) and internet of drones (iod). Applied System Innovation **5**(3), 56 (2022)
17. Jones, D., Snider, C., Nassehi, A., Yon, J., Hicks, B.: Characterising the digital twin: A systematic literature review. CIRP Journal of Manufacturing Science and Technology **29**, 36–52 (2020). https://doi.org/10.1016/j.cirpj.2020.02.002
18. Ju, H., Juan, R., Gomez, R., Nakamura, K., Li, G.: Transferring policy of deep reinforcement learning from simulation to reality for robotics. Nature Machine Intelligence **4**(12), 1077–1087 (2022). https://doi.org/10.1038/s42256-022-00573-6
19. Julian, R., Swanson, B., Sukhatme, G.S., Levine, S., Finn, C., Hausman, K.: Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning. arXiv preprint arXiv:2004.10190 (2020)
20. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Nitin Bhagoji, A., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R.G.L., Eichner, H., El Rouayheb, S., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P.B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konecný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Qi, H., Ramage, D., Raskar, R., Raykova, M., Song, D., Song, W., Stich, S.U., Sun, Z., Suresh, A.T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F.X., Yu, H., Zhao, S. (eds.): Advances and Open Problems in Federated Learning, Foundation and Trends in Machine Learning, vol. 14. now Publishers Inc. (2021). https://doi.org/10.1561/2200000083
21. Matulis, M., Harvey, C.: A robot arm digital twin utilising reinforcement learning. Comput. Graph. **95**, 106–114 (2021). https://doi.org/10.1016/j.cag.2021.01.011
22. Menzel, T., Bagschik, G., Maurer, M.: 2018 IEEE Intelligent Vehicles Symposium (IV): 26-30 June 2018. IEEE, Piscataway, NJ (2018)
23. Modelica Association Project FMI: Functional mock-up interface for model exchange and co-simulation (02102019)
24. Neurohr, C., Westhofen, L., Henning, T., de Graaff, T., Möhlmann, E., Böde, E.: Fundamental considerations around scenario-based testing for automated driving. IEEE Intell. Veh. Symp. Proc. pp. 121–127 (2020). https://doi.org/10.1109/IV47402.2020.9304823
25. Osiński, B., Jakubowski, A., Zięcina, P., Miłoś, P., Galias, C., Homoceanu, S., Michalewski, H.: Simulation-based reinforcement learning for real-world autonomous driving. In: 2020 IEEE International Conference on Robotics and Automation (ICRA), pp. 6411–6418. IEEE (2020)
26. Pohl, K., Broy, M., Daembkes, H., Hönninger, H. (eds.): Advanced Model-Based Engineering of Embedded Systems. Springer International Publishing (2016). https://doi.org/10.1007/978-3-319-48003-9
27. Shen, G., Lei, L., Li, Z., Cai, S., Zhang, L., Cao, P., Liu, X.: Deep reinforcement learning for flocking motion of multi-uav systems: Learn from a digital twin. IEEE Internet of Things Journal **9**(13), 11141–11153 (2021)
28. Shen, G., Lei, L., Zhang, X., Li, Z., Cai, S., Zhang, L.: Multi-uav cooperative search based on reinforcement learning with a digital twin driven training framework. IEEE Transactions on Vehicular Technology (2023)
29. Stark, R., Damerau, T.: Digital twin. In: Chatti, S., Tolio, T. (eds.) CIRP Encyclopedia of Production Engineering, pp. 1–8. Springer Berlin Heidelberg, Berlin, Heidelberg (2019). https://doi.org/10.1007/978-3-642-35950-7{_}16870-1

30. Sun, W., Lei, S., Wang, L., Liu, Z., Zhang, Y.: Adaptive federated learning and digital twin for industrial internet of things. IEEE Transactions on Industrial Informatics **17**(8), 5605–5614 (2021). https://doi.org/10.1109/TII.2020.3034674
31. Sutton, R.S., Barto, A.: Reinforcement learning: An introduction. Adaptive computation and machine learning, The MIT Press, Cambridge, Massachusetts and London, England, second edition edn. (2018)
32. Wagg, D.J., Worden, K., Barthorpe, R.J., Gardner, P.: Digital twins: State-of-the-art and future directions for modeling and simulation in engineering dynamics applications. ASCE - ASME Journal of Risk and Uncertainty in Engineering Systems **6**(3) (2020). https://doi.org/10.1115/1.4046739
33. Zhang, Z., Zahng, D., Qiu, R.C.: Deep reinforcement learning for power system: An overview. CSEE Journal of Power and Energy Systems **6**(1) (2020). https://doi.org/10.17775/CSEEJPES.2019.00920