Discover Internet of Things

Research

Reinforcement learning and digital twin-driven optimization of production scheduling with the digital model playground

Arne Seipolt[1,2] · Ralf Buschermöhle[1] · Vladislav Haag[1] · Wilhelm Hasselbring[2] · Maximilian Höfinghoff[1] · Marcel Schumacher[1] · Henrik Wilbers[1]

Received: 22 August 2024 / Accepted: 13 December 2024
Published online: 23 December 2024

Abstract
The significance of digital technologies in the context of digitizing production processes, such as Artificial Intelligence (AI) and Digital Twins, is on the rise. A promising avenue of research is the optimization of digital twins through Reinforcement Learning (RL). This necessitates a simulation environment that can be integrated with RL. One is introduced in this paper as the Digital Model Playground (DMPG). The paper outlines the implementation of the DMPG, followed by demonstrating its application in optimizing production scheduling through RL within a sample process. Although there is potential for further development, the DMPG already enables the modeling and optimization of production processes using RL and is comparable to commercial discrete event simulation software regarding the simulation-speed. Furthermore, it is highly flexible and adaptable, as shown by two projects, which distribute the DMPG to a high-performance cluster or generate 2D/3D-Visualization of the simulation model with Unreal. This establishes the DMPG as a valuable tool for advancing the digital transformation of manufacturing systems, affirming its potential impact on the future of production optimization. Currently, planned extensions include the integration of more optimization algorithms and Process Mining techniques, to further enhance the usability of the framework.

Article Highlights

1. Open-Source Flexibility: As a user-friendly, adaptable framework, DMPG is comparable to commercial simulation tools regarding the simulation speed. It can be used to distribute simulations on high-performance clusters or to generate 2D/3D-Visualization of processes with Unreal.
2. Enhanced Production Scheduling: DMPG streamlines production scheduling using reinforcement learning. The extendable code structure allows the implementation of further simulation algorithms.
3. Ongoing Development: Future enhancements include detailed transport and process mining, broadening its application.

Keywords Hybrid simulation · Discrete event simulation · Reinforcement learning · Digital twins · Production scheduling

✉ Arne Seipolt, a.seipolt@hs-osnabrueck.de; Ralf Buschermöhle, r.buschermoehle@hs-osnabrueck.de; Vladislav Haag, vladislav.haag@hs-osnabrueck.de; Wilhelm Hasselbring, hasselbring@email.uni-kiel.de; Maximilian Höfinghoff, m.hoefinghoff@hs-osnabrueck.de; Marcel Schumacher, marcel.schumacher@hs-osnabrueck.de; Henrik Wilbers, henrik.wilbers@hs-osnabrueck.de | [1]Faculty of Management, Culture and Technology, Osnabrück University of Applied Sciences, Lingen, Germany. [2]Department of Computer Science, Kiel University, Kiel, Germany.

1 Introduction

The increasing importance of production data utilization in manufacturing companies is undeniable, offering myriad benefits such as enhanced visibility of manufacturing processes, easier adoption of Artificial Intelligence (AI) and Machine Learning (ML) technologies, and streamlined production scheduling [1]. The capability to collect data from various sources facilitates the creation of Digital Twins (DTs), digital representations of physical entities [2], which have been proven to significantly boost operational efficiency if used in production processes by an average of 15% and are highly valued for simulation services by 68% of organizations [3, 4]. Despite the complexity of production scheduling issues, often classified as NP-complete, Reinforcement Learning (RL) emerges as an effective approach to solve such combinatorial optimization problems [5, 6]. An early adoption was done by Crites [7] and Crites and Barto [8], who used a RL agent in a discrete event simulation to dispatch elevators. Nowadays, with more computational power and optimized algorithms, more complex problems can be solved. For example, the definition of tasks and destinations for autonomous guided vehicles in a modular production system [9] or to solve a dynamic job shop scheduling problem [10].

There is a significant rise of publications in the implementation of RL in production planning, which demonstrates the increasing interest of the scientific community in this particular domain [11, 12]. Esteso et al. conducted a review of 181 papers, spanning from 1994 to 2021, and discovered that the majority of the research is centered around the area of production scheduling [12]. Other areas, like capacity planning, are not in the focus of the research. Panzer et al. conducted a thorough review of 129 papers, spanning the years 2010 to 2021, and came to the conclusion that in 89% of the benchmark papers, RL algorithms outperformed the algorithm it was compared with [11]. Nevertheless, 95% of the studies were performed in a simulated laboratory environment. Therefore, it is challenging to draw general conclusions about the reliability and sustainability of the results in real-world environments. As challenges, which still prevent widespread adoption in production systems, they summarize missing hands-on guidelines, limited use of the available algorithm base and the lack of evaluation in reality. According to them, future development should focus on further refinement of the simulations, using existing, more powerful RL algorithms and the elaboration of increased generalizability, among others.

Digital Twins can be used for production scheduling [13, 14] by utilizing the real-time perception and the simulation capability of the DT [15]. Additionally, several Authors combine RL and DTs for Production Scheduling [16, 17] and Ouahabi et al. state that this combination is overshadowing traditional metaheuristics used for Production Scheduling because of the real-time adaptability to disruptions [14]. According to Kritzinger et al. [18], there are three different types exist under the concept of DTs: the Digital Models (DM) consist of a physical and a digital object, with no automatic data flow between these objects. The next Level is the Digital Shadow (DS), where there is an automatic unidirectional connection between these objects and a change of the state of the physical object leads to an automatic update of the digital object. At the third level, the Digital Twin (DT), the automatic connection is bidirectional, so additionally the state of the physical object is changed automatically, if the state of the digital object is changed. This categorization is extended and formalized by Barbie and Hasselbring [2]. They define the Digital Model as the description of an object, process or a complex aggregation, either mathematical or by computer-aided design. Furthermore, there is a Digital Thread, which refers to the communication framework that allows a connected data flow and integrated view of the physical twin's data and operations throughout its life cycle. Both, Digital Shadow and Digital Twin are connected to the Physical Twin, which refers to the physical object, via the Digital Thread, either unidirectional or bidirectional. The Digital Shadow and the Digital twin can update the Digital Model, and the Digital Twin can also send commands to the Physical Twin. Therefore, the Digital Model is a central component of Digital Shadows and Digital Twins and replaces the digital object as defined by Kritzinger et al.

A concept related to Digital Twins is the smart Industrial Internet of Things, which is an integrated System that synthesizes cyber operations like communication or computation and physical processes [19]. Xu et al. state that smart IIoT systems and Digital Twins can be combined. The Digital Twin runs AI algorithms for experiments and interacts with the smart IIoT system.

A production process can be modeled as a predictive, mathematical model, which is required to simulate the different aspects of the production process, to improve or optimize it in a Digital Twin context [20]. This demonstrates the need for a simulation environment that can model production processes with a high level of detail and is compatible with common frameworks of powerful RL algorithms. Furthermore, it must be flexible enough to enable automated unidirectional or bidirectional connection to the physical object via a Digital Thread.

With the Digital Model Playground (DMPG), this paper proposes a flexible, user-friendly open-source framework for hybrid simulations, combining DES and continuous simulations. The architecture of DMPG allows fast and distributed simulation, to enable the use of numeric optimization. Furthermore, DMPG can be used to train RL-Agents, for example, for Production Scheduling (PS). The aim of this paper is to show, that DMPG enables researchers and practitioners to overcome the challenges, stated by Panzer et al. [11] and to develop real-world applications with RL in the area of production scheduling.

This paper introduces the DMPG, starting with an overview over related work for production scheduling with Reinforcement Learning. Then, the Framework DMPG is shown in general for basic simulation as well as to train an RL-Agent. Here, a simple example is used, to show the functionality. Then ongoing challenges with the DMPG are discussed, finishing the Paper with a conclusion.

2 Related work

The scheduling in a production context can be defined as follows:

"Scheduling problems can be understood in general as the problems of allocating resources over time to perform a set of tasks being parts of some processes, among which computational and manufacturing ones are most important." [21]. "In manufacturing, the purpose of production scheduling is to minimize production time and costs, by telling a production facility when to make something, with which staff, and with which equipment." [22].

An important focus in current research is the application of RL methods for production planning: Several authors have made Literature reviews and found an increasing number of publications in solving the combinatorial optimization problem of production scheduling with RL methods [11, 12, 23]. As stated above, Panzer et al. recommend, that future development should focus on further refinement of the simulations, using existing, more powerful RL algorithms and the elaboration of increased generalizability, among others.

While Commercial software for Discrete Event Simulation (DES) is available, Dagkakis and Heavey state they have limitations regarding cost, flexibility, and reusability [24]. Regarding Open Source (OS) DES projects, they name some critical success factors, including:

- Focus on industry: Many OS projects are outcomes of research funding, lasted for a limited timeframe and got abandoned.
- Addressing the needs of different Users

There are a couple of OS DES tools, which enable the use of RL for Production Scheduling. SimRLFab is a simulation and reinforcement framework for production planning and control of complex job shop manufacturing systems, based on Python [25]. The RL-Algorithms of the Tensorforce library are used. The last commit to this GitHub Project was in June 2020, which shows that the Project is not active anymore. Another OS RL Framework is FabricatioRL, which implements the OpenAI Gym Standard and is therefore compatible with different State of the Art RL-Algorithms [26]. Nevertheless, it lacks several features, which are required for industry scale simulations like machine breakdowns or personal schedules. Furthermore, the last commit to the GitHub Project is from July 2023, so this project is not active either. A Third framework is or-gym, which is a Python Library that contains environments consisting of operation research problems which adhere to the OpenAI Gym API, enabling the use of different State of the Art RL-Algorithms [27]. It has a list of standard applications, like the Knapsack or traveling salesman Problem, but does not support simulations of real life systems. In Conclusion, all of these Frameworks have neither focused on industry nor addressed the need of different users and seem to be abandoned.

3 Digital model playground

The findings mentioned before state the need for an easy to handle but adaptable simulation framework, which supports high-fidelity simulations as well as state-of-the art RL-Algorithms and other Algorithms for combinatorial optimization.

An open-source project at the Hochschule Osnabrück is meant to create this framework and to use it for research and teaching. It is named Digital Model Playground (DMPG) [28].

The DMPG is an open source discrete event-based simulation framework, based on SimPy [29]. With the library SimPy, the programming language Python provides the basis on which DMPG is built. SimPy is chosen over other simulation tools, since it is easy to learn, easy adaptable, is an active Project and available under a MIT-License. The DMPG is meant to extend SimPy to provide more user-friendly functions, to make it easier to model different processes in Python. For example, to calculate statistic measurements, graphical display of the simulation model or the use of common simulation objects like Sources, Servers, and Sinks. Further deployed open-source frameworks include Seaborn, Graphviz, SciPy, TF-Agents and DistFit.

DMPG it is meant to provide a Framework to implement Digital Twins and to use artificial intelligence, for example to optimize the production planning of existing production processes with RL. Therefore, an interface to TF-Agents exists. This chapter shows the framework of the DMPG, by first introducing the architecture for basic simulation and comparing the DMPG with commercial DES-software regarding the simulation speed. Then, the training of an RL-Agent for PS is shown on a simple example.

3.1 System architecture

The SimPy library (SimPy—Discrete event simulation for Python n.d.) served as the foundational framework for the DMPG. It provides a process-based simulation environment, where the sequence of events is controlled through Python's generator functions. DMPG extends this by introducing the above-named components necessary for discrete event simulations, mainly entities, sources, servers, and sinks. The features adopted from SimPy include:

- Environment: This is the core of any SimPy simulation, providing the context in which entities interact, and events are scheduled. DMPG utilizes the Environment to manage the simulation timeline and ensure that all events are processed in a timely manner.
- Timeout: SimPy's timeout events are used in DMPG to simulate processing times or delays within the components, managing how long an entity spends in a particular process.
- Process: This SimPy functionality is used to define the behavior of entities as they go through various simulation stages. In DMPG, processes are used to model the operations within sources, servers, and sinks.
- Event: SimPy events are mechanisms that trigger subsequent actions in the simulation. DMPG uses these to handle disruptions, such as machine breakdowns in servers, effectively managing the simulation's response to operational anomalies.

To solve differential equations for continuous elements in the simulation, the function **scypy.integrate.odeint** is used.

The DMPG is built around key components modeled through classes that manage different facets of a simulation. These components include entities, sources, servers, and sinks, each playing a specific role within the simulation framework:

- Entities are dynamic units that move through the simulation, undergoing various processes and transformations dictated by their interactions with servers and sinks.
- Sources generate entities according to specified distributions or schedules, initiating the simulation process.
- Servers represent processing stations where entities are handled. They can simulate operational constraints like processing times, random disruptions, capacity limits and equipment failures.
- Sinks are termination points for entities, where data about the entities' life cycles are collected and aggregated for analysis.

The systems' architecture is shown in Fig. 1. The components Source, Server, and Sink inherit from the *ResetAbleNamedObject* class and are managed through the Model class. The Model class in the DMPG serves as a central registry that organizes and manages different types of simulation components, ensuring that each component type is systematically handled and accessible within the simulation environment. The *ResetAbleNamedObject* class provides a common base for all simulation objects, facilitating their initialization, state management, and reset capabilities. This inheritance ensures that all objects can be reset to their initial state, which is critical for running multiple simulation trials under consistent conditions. Management of these objects is further streamlined by the *ResetAbleNamedObjectManager*, a utility class that handles collections of simulation objects. This manager allows for group operations such as mass resetting of states. The flow of entities through the system is orchestrated by the *RoutingObject* class to manage dynamic routing of entities between different simulation stations. This class ensures

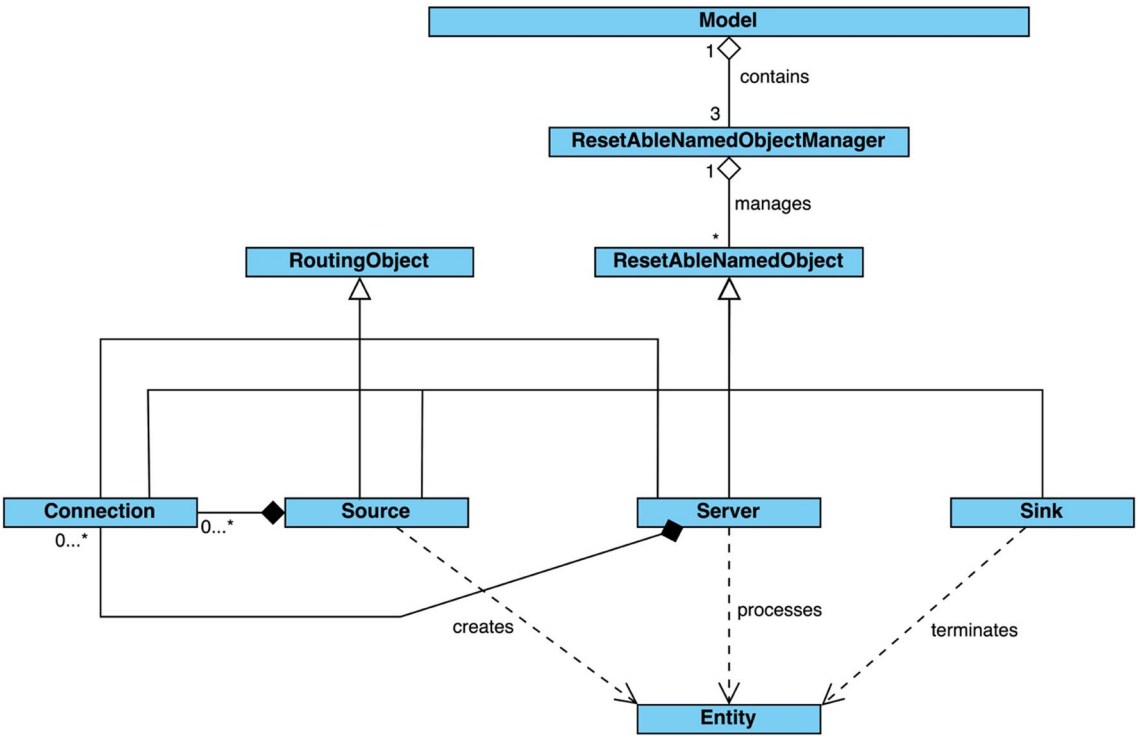


Fig. 1 Class relationships in the DMPG

that entities are directed correctly according to the simulation design. This architecture enables basic simulation of production processes. Additionally, the implementation of the following features further extends the flexibility of the DMPG.

The implementation of the *Connections* class fundamentally mirrors that of servers, differing primarily in their attributes. *Connections*, unlike servers, possess a probability attribute, which specifies the likelihood of the respective connection being utilized during entity routing. These enhancements empower connections between servers to route entities not only instantaneously, but also consider temporal dynamics or incorporate supplementary expressions. Therefore, in addition to processing entities at designated workstations, the DMPG possesses the capability to simulate pathways and factor in inter-server distances. Thus, a real-life transportation of workpieces within a production environment between workstations can be represented.

Moreover, *entities* can be directed through customizable routing expressions rather than relying solely on stochastic methods. This implies that processed workpieces, for example, can be further processed or routed in different ways based on various product specific work plans or characteristics. Consequently, *connections* operate precisely like *server* objects. They independently process entities before their transfer to the respective server's subsequent queue. In the context of a real-life system, i.e., conveyor belts can thus be simulated to transport workpieces between individual stations.

Besides the core package, the DMPG facilitates the creation of model-specific entities through the *SubEntity* class, extending the versatility of the simulation environment. Multiple sub-entities can be accommodated within the same model, catering to diverse routing requirements, such as product differentiation between servers. Moreover, various sub-entities, generated by one or multiple *sources*, can undergo distinct routing trajectories within the system. Thus, it is possible to introduce different products into the i.e., simulated production and process them in different ways. Consequently, variants of a product are usable within the same simulation model. Additionally, each *entity* type or workpiece is empowered to generate its set of statistics when being processed. Custom processing statistics can be seamlessly incorporated into the corresponding product, either as pre- or postfixes, ensuring dynamic adaptability of the calculated results to meet specific simulation demands.

Lastly, to accurately reflect operational realities, dynamic work schedules regarding working stations can be set. Simulation models can incorporate multiple working schedules, which can be assigned to individual servers to emulate diverse work shifts across different stations. This comprehensive approach allows for the representation of various shift patterns, breaks, durations, and worker capacities, thus enhancing the fidelity of the simulated environment. Moreover,

the open-source nature of the framework enables users to create their custom simulation objects. This provides the flexibility to model high-fidelity simulations of complex processes.

The sequence diagram (Fig. 2) outlines the operational workflow within the DMPG, starting from user interaction and progressing through the model's setup and execution phases. Initially, the user defines the model by specifying configurations that shape the simulation's structure and behavior. Once the model is defined, the *run_simulation* function can be invoked by the user, which acts as the driver for the entire simulation process. Within the simulation, the EntityManager is initialized first to manage all entities created during the simulation. The manager ensures that entities are correctly added, tracked, and their destruction times are updated as they complete their processes. Entities are then initialized to represent the dynamic units that will pass through various simulation stages. Sources, Servers, and Sinks are created subsequently.

Next, the connections between sources, servers, and sinks are established to define the path that entities will take through the simulation. These connections are crucial as they dictate the flow and outcome of the simulation by determining how entities move from one component to another:

Entities are created by sources and then added to servers. Depending on the simulation's logic, entities might be routed to other servers or directed towards sinks, where they are ultimately processed and their lifecycle within the simulation ends.

The simulation's execution involves processing each entity individually through the configured routes. Servers handle entities by processing them according to the defined logic, which can include handling breakdowns or continuing processing under normal conditions. Once processing is completed, entities are either routed to another server for further processing or sent to a sink for termination. The simulation continues this loop until all entities are processed or the defined simulation time ends. Once the simulation concludes, statistics are calculated to analyze the performance and results of the simulation.

These statistics include average, minimum, and maximum of times in system, number of entities created and destroyed, as well as specific server metrics like average processing time, scheduled utilization, and total downtime. These metrics provide insights into the efficiency, effectiveness, and dynamics of the simulated environment, offering data that can be used to adjust the model or understand the system's behavior.

Setting up a simulation involves configuring various parameters for sources, servers, and sinks to reflect specific operational requirements and behaviors. Sources must be configured with parameters that define the frequency and conditions under which entities are generated. Servers require setup details including processing time distributions to simulate the time taken for tasks in the production process. Additional parameters might include capacity constraints and breakdown probabilities. Sinks are configured to collect and terminate entities.

To deploy a simulation, users must define the model configuration through code. Figure 3 shows a basic setup example, which demonstrates the initialization of a simple model where entities are created, processed, and terminated.

This code configures a source to generate entities at an exponential rate, a server to process them with a triangular distribution of processing times, and a sink to collect and terminate the entities. Afterward, the Source is connected to the Server and the Server to the sink, to define the routing logic. This will create the objects, shown in Fig. 4.

The *setup_model* consists of three *ResetAbleNamedObjectManagers,* each managing a single object. *Connections* are routing *entities* between *source* and *server, as* well as between *server* and *sink*. The simulation is then run using specific parameters set in the *run_replications* function, which allows for multiple replications of the model over a set time frame.

Besides the RL-Package, which is introduced in the next chapter, there is a project which makes use of a high-performance cluster by introducing distributed computing and a package for 3D-Visualization, which is briefly introduced in the following.

During the Software Development Project at the Osnabrueck University of Applied Sciences, a group of students developed a service to run the simulation on the High-Performance Cluster of the University [30]. The service is provided on a Web server, where students can use their university-credentials to authenticate. Afterward, the model can be defined by using drop-down menus. When the job is deployed to the server, an estimated remaining time is given. The results of the simulation runs are stored in a database, to be evaluated. This provides the possibility so simulate numerous scenarios of complex models in a short period of time.

Another Software Development Project has created a 3D-Visualization of process models with Unreal [31]. When the tool is started, a simulation model can be chosen from a dropdown menu. Next, either automatic mapping can be applied, or the objects can be mapped manually. Figure 5 shows screenshots of an example model.

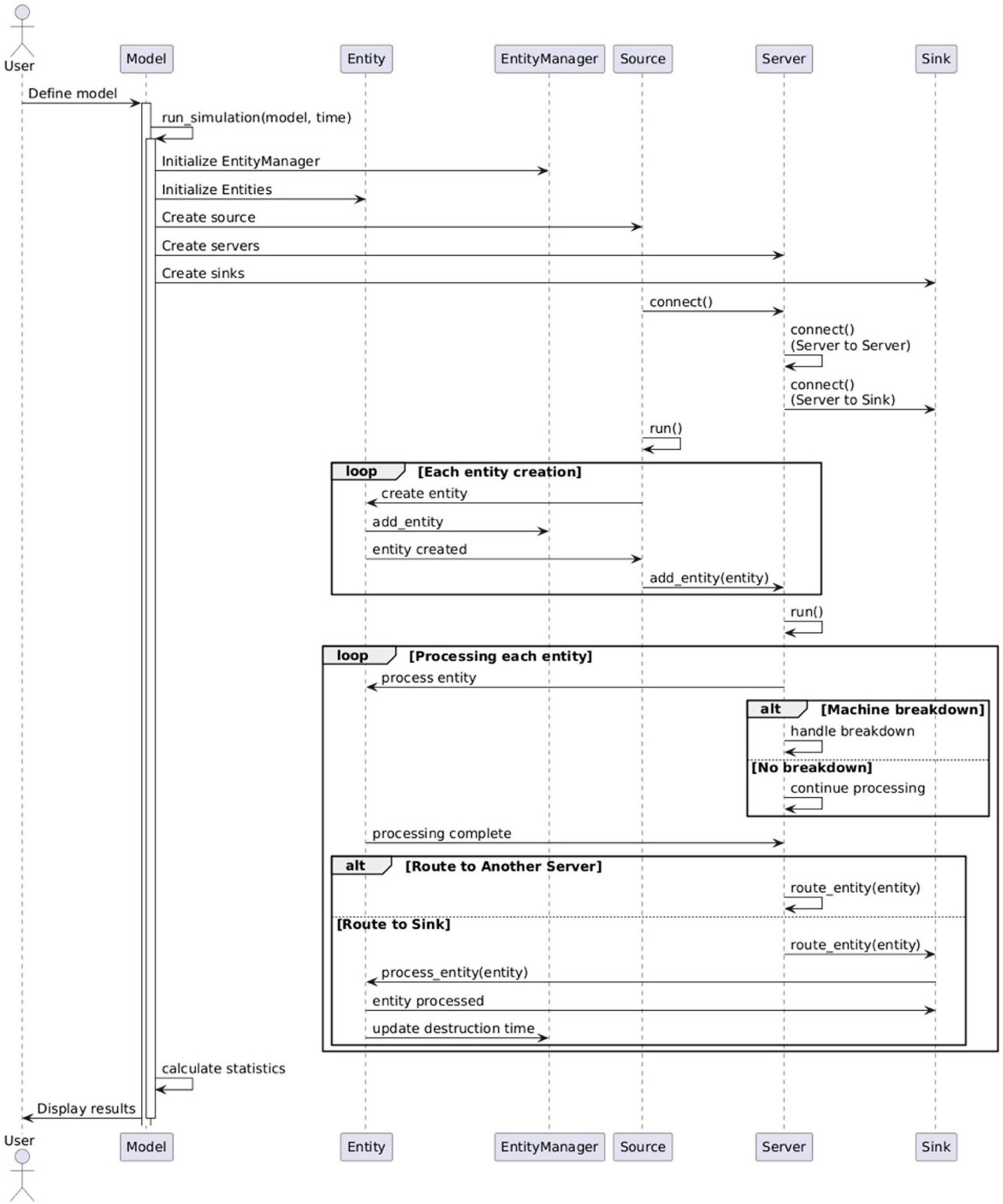


Fig. 2 Sequence diagram of a DMPG-based simulation

```python
def setup_model(env):
    source = Source(env, name: "Source", creation_time_distribution_with_parameters: (random.expovariate, 1 / 6))
    server = Server(env, name: "Server", processing_time_distribution_with_parameters: (random.triangular, 3, 4, 5))
    sink = Sink(env, name: "Sink")

    source.connect(server)
    server.connect(sink)


def main():
    run_replications(model=setup_model, minutes=72000, num_replications=10, multiprocessing=True)
```

Fig. 3 Example of a simulation setup for DMPG

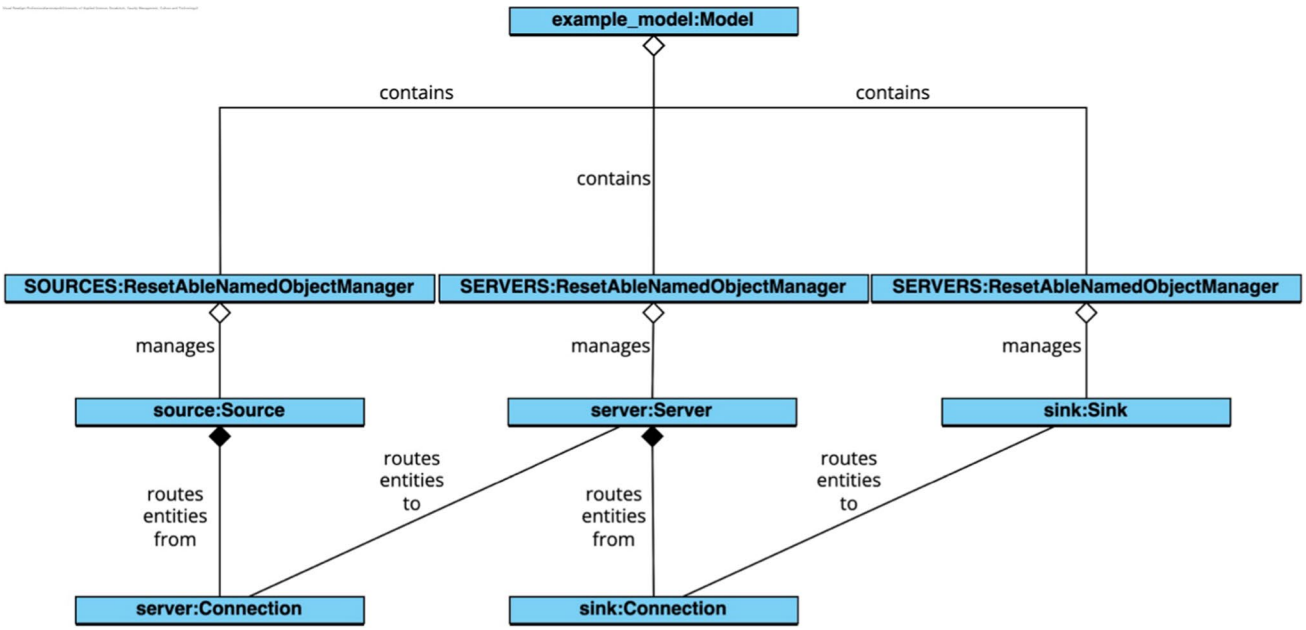


Fig. 4 Object overview for the example model

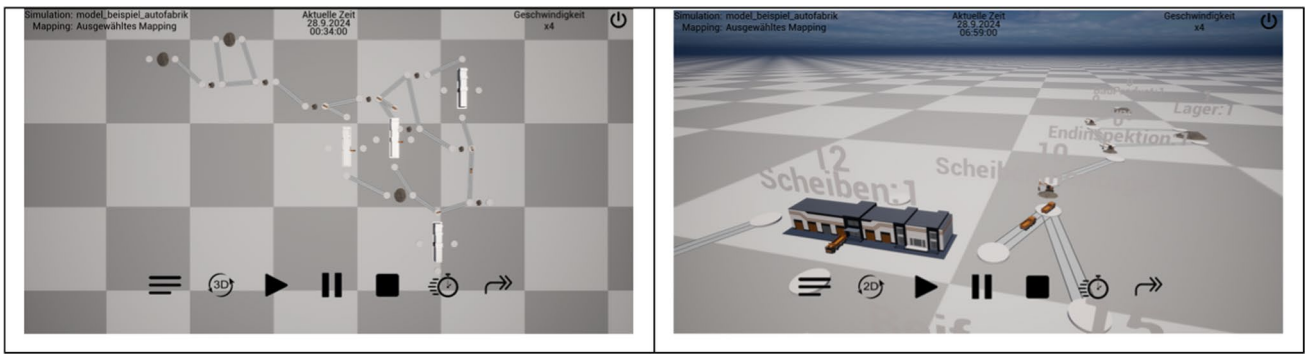


Fig. 5 Visualization of the DMPG with unreal

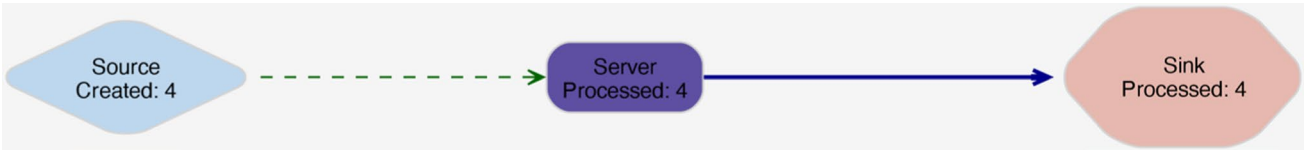


Fig. 6 Simulation model

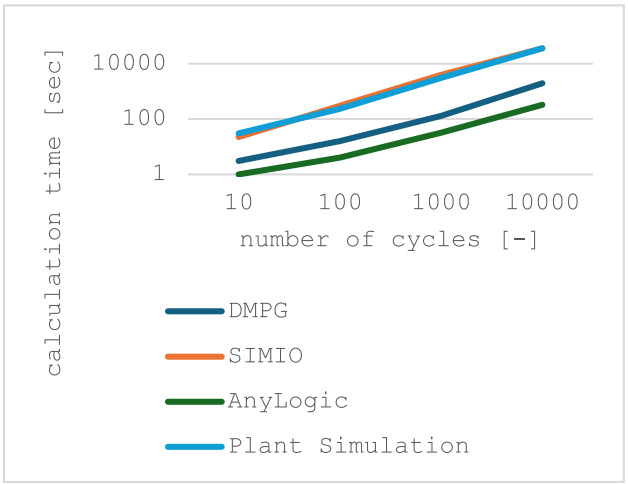


Fig. 7 Calculation time of the benchmark between commercial tools and DMPG

After describing the system architecture and basic features of DMPG, the next chapter will show a performance comparison between DMPG and multiple commercial DES-simulation tools.

3.2 Performance comparison between DMPG and commercial DES-tools

Since multiple replications are required to calculate statistic values of DES-Simulations, the computational performance of the simulation environment is important. To compare DMPG with some commercial DES-tools, the process shown in Fig. 6 is modelled in all simulation environments and different number of replications are performed. The simulation environments Any Logic (Version 8.9.1), Plant Simulation (Version 2404.0005) and Simio (Version 16.255.34527) are used for the Benchmark, since they are designed for DES and used to train RL-Agents [32]. The benchmark was performed on a Windows 11 23H2 System with an Intel i7-12800H 2.4 GHz 14 Core processor and 32 GB RAM. The DMPG Version used is available in the DMPG GitLab [28] under the commit hash 6b2bb59f. Figure 7 shows the total calculation time. Simio and Plant Simulation need considerably more time, to finish the simulation, as DMPG. AnyLogic, on the other hand, is much faster than DMPG. Figure 8 shows the processor utilization. While the processor utilization of Simio, AnyLogic and Plant simulation remains constant, DMPG has a much higher utilization which is above 80% after 100 cycles. A comparison of the processor time, which is calculated by multiplying the calculation time with the processor utilization (Fig. 9), shows that the efficiency of DMPG and Plant Simulation is comparable. SIMIO needs a bit more processor time, AnyLogic considerably less. The lesser calculation time of the DMPG in comparison to SIMIO and Plant Simulation is a result of the higher processor utilization. To distribute the simulation replications, for example with Simio, the Simio replication runner can be used. Since this is an additional program which must be configured, multiprocessing with DMPG is much easier because it runs automatically. The RAM utilization is shown in Fig. 10. It indicates that the RAM Utilization of all commercial simulation environments is constant below 1000 GB, while the RAM utilization of DMPG rises from 800 MB at 10 replications to 27,000 MB at 10,000 replications. This is because DMPG does not delete any data, until all replications are finished, to calculate statistics.

After presenting the structure of the DMPG, the integration of RL and a simple example, which proofs the functionality of the concept of optimizing production scheduling problems with RL.

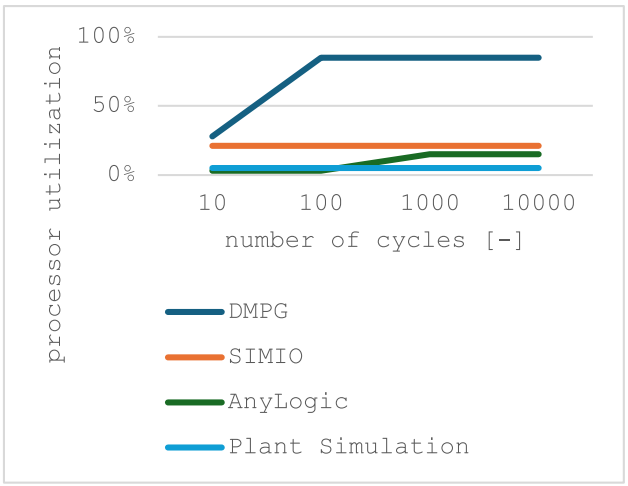


Fig. 8 Processor utilization of the benchmark between commercial tools and DMPG

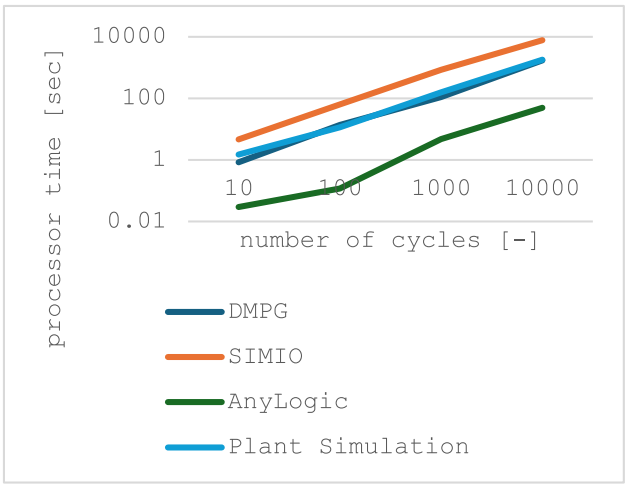


Fig. 9 Processor time of the benchmark between commercial tools and DMPG

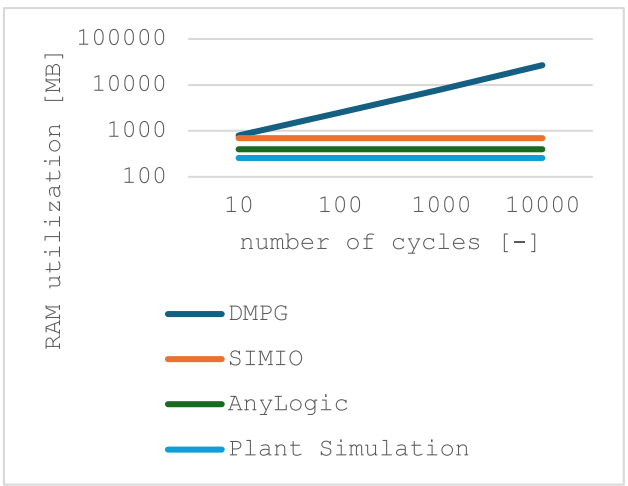


Fig. 10 RAM utilization of the benchmark between commercial tools and DMPG

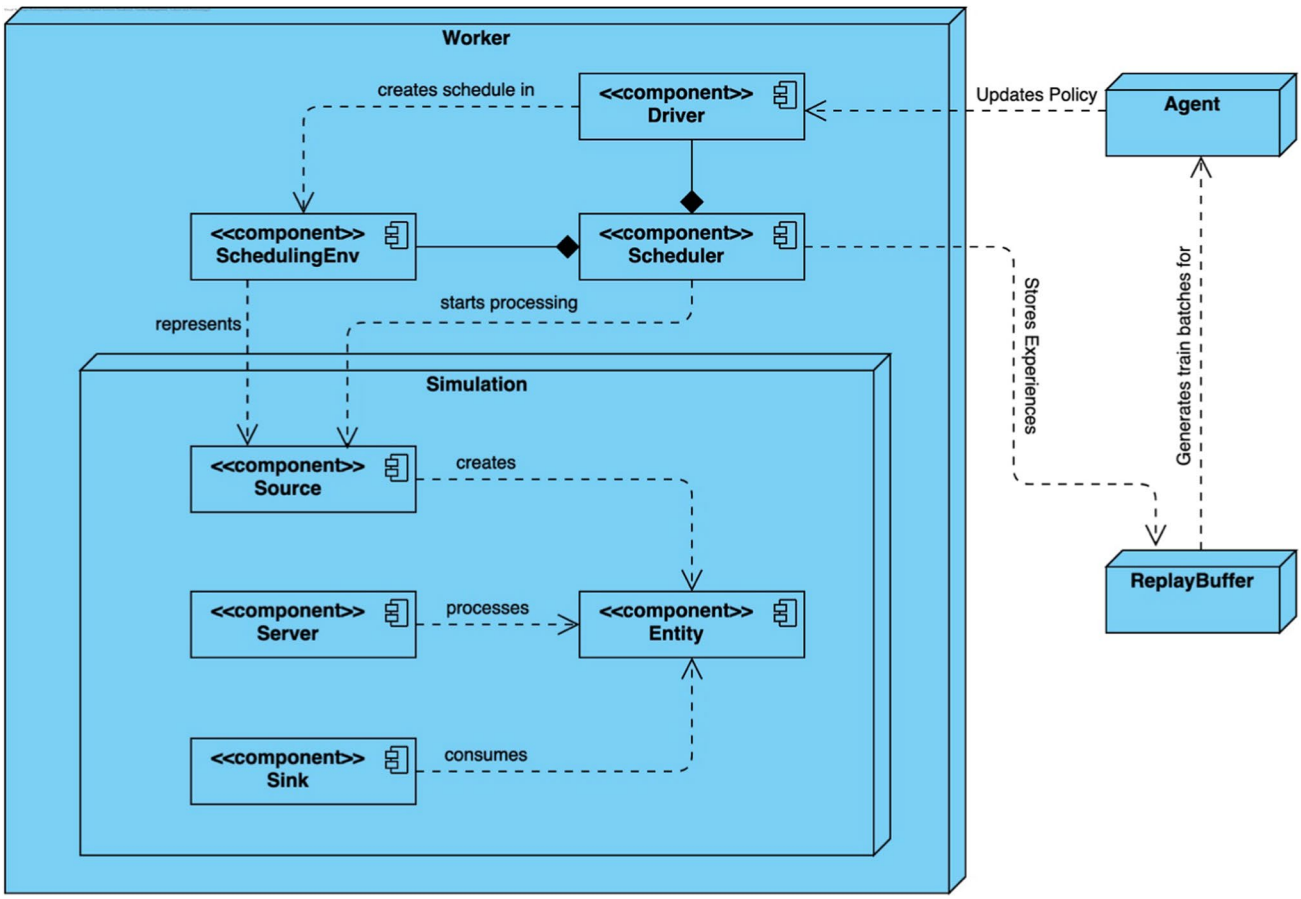


Fig. 11 Deployment diagram illustrating the integration within DMPG

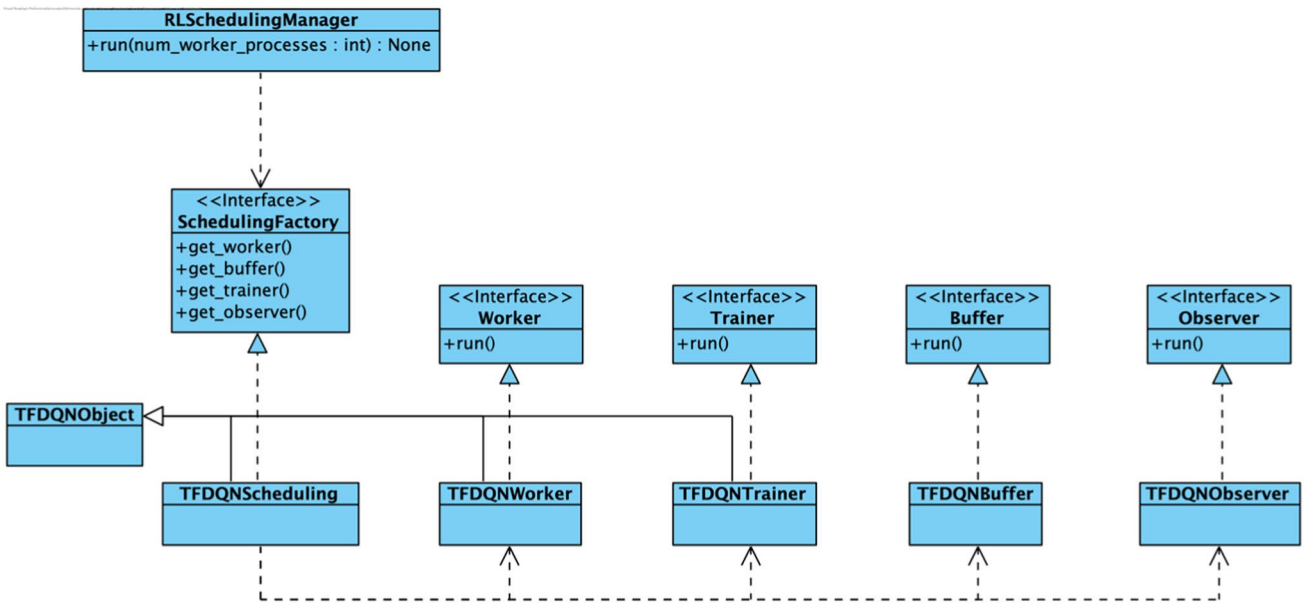


Fig. 12 Creation of the different Processes with the abstract factory design pattern

3.3 RL-based production scheduling with DMPG

The integration of RL in DMPG must be flexible and extendable enough, to leverage different RL-Frameworks. Furthermore, the simulation of process models must be independent of the simulation. This enables the testing of one or multiple pretrained RL-agents in the simulation. The current integration of Reinforcement Learning (RL) into the Digital Model Playground (DMPG) for enhancing production scheduling leverages TensorFlow [33], which provides TF-Agents, a popular AI framework for Python that supports several RL-Algorithms, including Deep-Q-Learning (DQN) as highlighted by Minh et al. [34]. In DMPG, a unique scheduler class has been introduced, incorporating a SchedulingEnv as well as a DQN-Agent and a replay buffer from TF-Agents for efficient learning (Fig. 11). The system is designed for high performance, with the scheduler, buffer, and agent operating in separate processes and allowing for parallel simulations to generate extensive training data. To keep the framework extendable, these processes are created by an abstract factory design pattern (Fig. 12). After the RLSchedulingManager is created, a concrete SchedulingFactory is defined, which implements the abstract factory class. The TFDQNScheduling class implements operations, which are required to deploy a TensorFlow DQN Agent to control the utilization of a server by scheduling the products. The User can define different aspects of the RL-agent, for example the learning rate. After the Factory is assigned to the RLSchedulingManager, the run function of the RLSchedulingManager is performed. This will create all necessary objects and start the training of the RL-Agent.

The practical application of this setup is demonstrated through a simplified process (Fig. 6), where the objective for the RL-Agent is to manage entity creation to maintain a server utilization close to but below 80%. The source controls the production scheduling, initiating with an empty schedule and updating it based on the RL-Agent's decisions. The Entities are processed at the server and destroyed in the sink. The processing time at the server is a triangular function, with a minimum value of 8, a modal value of 10 and a maximum value of 11 time steps. Although DMPG can simulate additional influences on the processing time like machine breakdowns, no more constraints are considered. The Simulation time is 500 time steps and every 100 time steps, a new schedule is created.

To realize this, the source has a scheduling period, a list of products which have to be produced and a list of products, which will be produced in the next scheduling period, called schedule. Initially, the schedule is empty.

If the source is called the first time by the simulation, the agent starts a new *SchedulingEnv*. The Observation of the environment is a list of three integers:

1. Products in the queue:
   The first integer represents the number of products to be produced in the next scheduling period. Initially, it corresponds to the number of products in the queue of the subsequent server.
2. Scheduling status:
   The second integer indicates whether the scheduling for the next scheduling period is finished. Initially, this value is 0, which indicates the scheduling is not finished.
3. Utilization:
   The third integer represents the utilization which is required to calculate the reward. If the current scheduling period is the initial one, it is also 0.

This observation is passed to the agent to decide which action to take. The Agent can choose between the two actions 0 and 1. If the action of the agent is 1, the next product of the list of products, which should be produced, is added to the schedule of the source. The next steps start, and the first value of the state is increased by one, since another product is produced in the next scheduling period. As soon as the agent chooses a 0, the scheduling is finished. No product is added to the schedule and the second value of the state is set to 1, which indicates to the agent, that the scheduling is finished.

If the scheduling is finished, the source creates the entities as scheduled and waits for the scheduling period to end. Then, the agent makes another step. In this step, the action is irrelevant. Now, the first value of the observation states the number of products in the queue of the next server, and therefore the number of products, which could not be produced. The second value is a 1, which indicates that the scheduling is still finished. The third one is the integer value of the utilization of the next server. This is the final step of the episode, and the only one, in which the agent receives a reward. The reward is calculated as follows: if the utilization U is zero or higher than the target utilization U_t the reward is zero. If it is between the target utilization and the target utilization, it is equal to the target utilization. This describes a scenario in which work is planned by people. However, the aim is to ensure that they are not fully utilized to prevent exhaustion.

```python
import src.rl_scheduling.rl_scheduler as scheduling
from src.models.test_DQN import setup_model_pcb as model
import pandas as pd
import math
import tensorflow as tf
from tf_dqn_scheduling import TFDQNScheduling

1 usage
def generate_test_products():
    products = []
    for i in range(50):
        products.append([i, i * 4, False])  # ID, due_date, Scheduled?
    products = pd.DataFrame(products, columns=['ID', 'Due Date', 'Scheduled'])
    return products

1 usage
def epsilon_fn(train_step):
    epsilon = max(math.exp(-train_step / 200), 0.2 - (0.2 / 4000) * train_step, 1 / 1000)
    return epsilon

if __name__ == '__main__':
    scheduling_manager = scheduling.RLSchedulingManager()

    products = generate_test_products()
    factory = TFDQNScheduling(model)
    factory.set_random_seed(3)
    factory.set_env_data( scheduling_period: 100, products)
    factory.set_q_net_data([16, 32])
    factory.set_learning_rate(tf.keras.optimizers.schedules.PiecewiseConstantDecay(
                                        boundaries: [100*5000], values: [10e-4, 10e-8]))
    factory.set_epsilon_fn(epsilon_fn)
    factory.set_target_utilization( source: "Source", target_utilization: 80, simulation_duration: 500)

    scheduling_manager.set_factory(factory)
    scheduling_manager.run(1)
```

Fig. 13 Code to deploy the training of the described DQN Agent

The deployment of this scenario is shown in Fig. 13. After the required imports, two functions are defined. The first is to generate a set of products, which must be scheduled. This is a very basic example, therefore no due dates, priorities are considered yet. The second one is a custom function to schedule the epsilon for the epsilon greedy strategy of the agent. In the main function, a scheduling manager is created by passing the simulation model. The simulation model is created according to Fig. 3. Next, the factory is parametrized by setting a random seed and the environment data. The q-net is defined by a list of the number of neurons for each hidden layer. Then the epsilon function and the learning rate are defined. Both can be a constant value, a TensorFlow function or a custom function. At last, the target utilization is set by passing the target source, the target value and the simulation duration. Lastly, the defined function is passed to the

Table 1 Hyperparameter

Parameter	Value
Optimizer	ADAM
Number of Neurons in first hidden layer	16
Number of Neurons in second hidden layer	32
Activation Hidden Layer	ReLu
Loss Function	Mean Squared Error
Train batch size	32
Number of Worker Proceses	1

scheduling manager and the training is started. When starting the training, several worker processes can be passed to create enough data points for computational expensive simulation models.

After the simulation is finished, the collected experiences are stored in the replay buffer. This enables to distribute multiple worker-processes, which is necessary to generate large datasets of computation intensive simulations. The replay buffer prepares datasets which are sent to the Agent. The agent uses these datasets to train the neural network which predicts the q-values. The neural network, which is the core of the driveŕ s policy, is distributed to the worker process, so that new simulation runs can be performed with the updated policy.

To show the functionality of the system, the above-described model is used to train an DQN-Agent. The training is performed on an Ubuntu 22.04.3 LTS system with a 12th Gen Intel® CoreTM i7-12800H × 20 Processor and NVIDIA RTX A2000 8GB Laptop GPU. The used Hyperparameters are stated in Table 1. The Optimizer, the activation of the hidden layer and the train batch size were set initially, based on previous experience. Since the simulated scenario is not computational expensive, a single worker process is used. Different loss functions were tested and the number of neurons per hidden layer where increased, starting with 16 neurons in each layer.

For epsilon, a function was used to reduce it during the train process. Every train iteration s, Epsilon is calculated with:

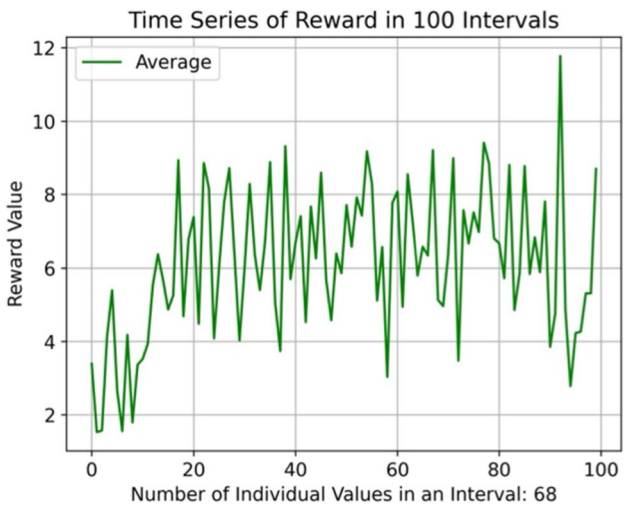
$$\varepsilon = \max\left(exp\left(-\frac{s}{200}\right), 0{,}2 - \left(\frac{0.2}{4000}\right) * s, \frac{1}{1000}\right)$$

This function allows a couple of random actions in the beginning. The fast-decaying exponential function allows a lower epsilon after a short time and is slowly decaying further, until most of the actions are based on the agent's policy.

Table 2 Ressource utilization

Average CPU	Average GPU	Average RAM	RAM (end of training)
352%	25,1%	7.355 GB	11.501 GB

Fig. 14 Reward



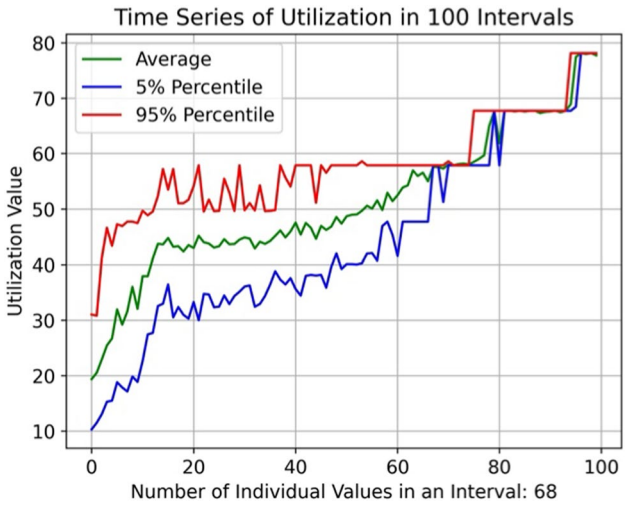


Fig. 15 Utilization

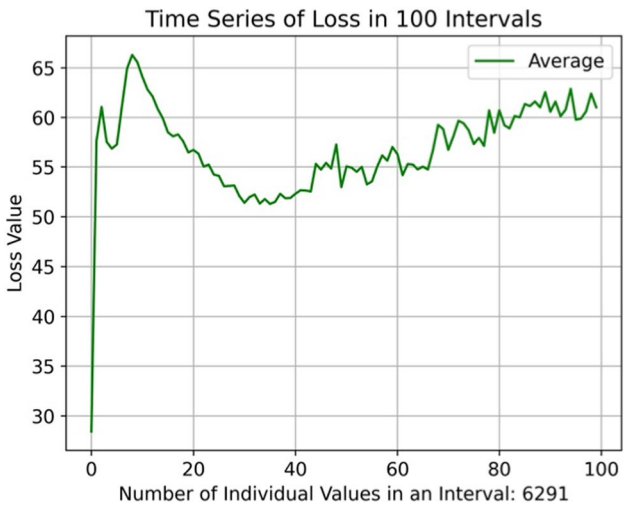


Fig. 16 Loss

100 train steps are conducted, in which the Q-Network is trained with the TensorFlow function. Then, the updated Q-Network is distributed to the worker. For the learning rate, a PolynomialDecay, as implemented in Tensorflow, is used with an initial learning rate of 10 – 4, end learning rate of 10 – 8 and 500,000 decay steps. Therefore, after 5000 iterations, the learning rate is decayed. If it would be higher, the training would diverge. After 4411 iterations, the training is interrupted because the training is stable and finished. This took about 55 min. It took several runs to get this result. The exact Code is available in DMPG [28] Gitlab under the commit hash d4be372c.

Table 2 shows the resource utilization. The average CPU load is mainly the worker process (104.5%), the train process (179.6%) and the buffer process (114.8%). Since most of the CPU-cores are not used and the GPU-utilization is only 25.1%, a parallelization of the train process could speed up the training, a parallelization of the worker process is already possible. Since the RAM-utilization at the end of the training is higher than the average RAM utilization, it can be concluded that some variables are not deleted, and storage is accumulated during the training.

Figure 14 shows the reward, Fig. 15 the utilization and Fig. 16 the loss during the train process. Every diagram shows the moving average in 100 intervals. For the utilization, additionally the 10% and 90% percentile of the interval are shown. While the reward is fluctuating from interval to interval, but clearly rises, the utilization is rising stable. In the end, the utilization remains stable, at 78.5%, which is close to the optimum of 80%.

3.4 Results

To propose DMPG as a suitable OS-DES framework, first the architecture and the capabilities are introduced. Furthermore, two extensions are shown: one, which deploys the DMPG on a high-performance cluster, another one which uses Unreal, to create a 2D/3D-Visualization of the simulation. This shows, that the DMPG can be easily adapted, to fit the user's individual needs.

Next, a performance comparison between DMPG and several commercial DES-Frameworks is made. It can be shown that the simulation speed of DMPG is comparable to the commercial frameworks. DMPG has the best utilization of available CPU-cores without additional tools. Since DMPG currently don't delete simulation data, it accumulates RAM, which can become critical if many simulation runs are performed on low RAM hardware.

In the end, the structure of the implementation of RL in the DMPG is shown and demonstrated. With the abstract factory design pattern, an extension with other algorithms can easily be realized. In a simple example, a RL agent is trained to control the utilization of a server to about 80%. This shows the functionality of the framework to train an RL-Agent to control aspects of the simulation model.

4 Discussion

As shown above, the DMPG can be used to model and simulate production processes. Nevertheless, there are still several limitations. Notably, resources utilizing facilities pose a challenge due to the inherent simplicity of the current implementations. Servers or simulated workstations within the framework solely depict entity processing. The incorporation of additional processing resources cannot be modeled, thereby confining simulated workstations to transformative activities rather than additive manufacturing. Furthermore, entities are conveyed without the involvement of vehicles or workers, thus neglecting the transportation and logistical aspects inherent in comprehensive facility simulations. To address this deficiency and accurately represent the movement of products between workstations, the integration of new simulation elements is imperative.

As the number of simulation objects expands, so does the demand for computational resources. At present, the DMPG operates within the constraints of utilizing separate cores of a single CPU. The RL-Training also uses a GPU, if available. However, to simulate various identical models, or, for instance, production facilities with varying numbers of workers, requires multiprocessing capabilities facilitated through network connections to distribute simulations across distinct computing nodes. This becomes particularly relevant when optimizing one model with diverse parameters in parallel using a Reinforcement Learning Agent. The Framework to deploy the DMPG to the high-performance cluster of the Osnabrueck University of Applied Sciences should be integrated into the main repository and made compatible with other features of the DMPG, like RL. This would make it easier for users to use this feature on their hardware. Moreover, since the GPU is only used to about 25% percent, a parallelization of the training of the RL-Agent could increase the training speed of the Agent.

The utilization of the DMPG requires a proficient command of both Python programming language and the framework itself. Moreover, retracing the behavior of a model necessitates debugging of the implementation, rather than providing visual representations of workstation activities and interactions between simulated real-life objects. On the other hand, the code-based design improves the flexibility of the framework and Python is currently one of the most popular programming language [35]. Therefore, the advantages outweigh the disadvantages at this point. Nevertheless, enhancing the accessibility of the DMPG is essential to broaden its user base, which can be beneficial for an OS-project [24]. The developed Framework to leverage Unreal allows users to visualize, rewind, and fast-forward simulation scenarios is a big improvement in this regard. It has the potential to deepen understanding of intricate processes and outcomes, thus facilitating more effective utilization of the framework. Although a user needs knowledge of the programming language and the framework, solutions can be visualized individually, according to the specific use case.

The modeled example shows that it is possible to train a DQN-Agents in the DMPG for production scheduling. It would also be possible, to use other RL-Algorithms: Since the optimization problem in the shown example has a relatively small size, simpler algorithms could lead faster to better results. For example, instead of training a Neural Network to predict the Q-Values, the Q-Values could be iterated directly and stored into a Q-Table. Besides RL-algorithms, other classes like genetic algorithms could be promising, since they are also in the focus of the literature

[36]. Moreover, many aspects which complicate the Production Scheduling, were not considered, for example the breakdowns of machines, product due dates, multiple machines which work in parallel and so on. In this case, maybe other algorithms are required. Based on the provided framework, more algorithms should be implemented and evaluated, so general conclusions can be drawn on which algorithm is suitable for which PS-problem.

Another planned extension of DMPG is the integration of Process Mining. Process Mining provides a comprehensive collection of algorithms and functions to create, check and monitor a process model based on data from the event logs of the process itself [37]. This allows for the easy creation of a process model that accurately depicts how the process is executed in the real world. The process model can be further enhanced with additional information from the process data, such as production times, social networks, etc. This enhanced process model can then be used in DMPG to create a simulation model of the process. This is possible by converting the various pieces of information from the process model into their counterparts in the simulation. As a result, the simulation model already contains a substantial amount of information and does not need to be modeled by hand. This saves time and significantly reduces errors. It is also possible to monitor the process to check for any changes in the real world which then can be reflected to the simulation model, keeping it always up to date. This moves the simulation from a Digital Model to a Digital Shadow of the process.

5 Conclusion

As demonstrated in this paper, DMPG is an effective framework for simulating production processes and utilizing RL agents to optimize the modeled production processes. It serves as a foundational tool already employed in research and teaching at Osnabrueck University of Applied Sciences. In research projects, complex real-life processes can already be modelled. Furthermore, the flexible OS-Structure of the DMPG allows adapting the framework to the user's specific needs, for example to connect the simulation to a Digital Thread, creating a Digital Shadow or Digital Twin. Moreover, several enhancements are currently planned or underway, including:

- More detailed transport logic, e.g. vehicles
- Working schedules
- Process mining
- Additional optimization algorithms

These developments will further enhance the framework's utility. The goal is to extend the use of DMPG beyond Osnabrueck University of Applied Sciences, providing a comprehensive platform that supports both academic and industrial users in modeling and optimizing production processes.

Acknowledgements This work is supported by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under grant No. 01MD22001C as part of the "edge data economy initiative".

Author contributions A.S. is the contributing author and wrote the main manuscript text. A.S. programmed the Reinforcement Learning part of the DMPG. M.S. conducted the comparison between SIMIO and DMPG. M.S. and M.H. supported the programming of the Reinforcement Learning part of the DMPG. V.H. described the architecture of the DMPG. V.H. and H.W. programmed the DMPG, excluding the Reinforcement Learning part. H.W. described current features of the DMPG, excluding Reinforcement Learning. R.B. supervised the programming of the DMPG. W.H. and R.B. supervised the project. All authors reviewed the manuscript.

Data availability The code used to show the functionality of the framework to train an RL-Agent to control aspects of the simulation model is available at https://gitlab.com/digitaltwinml/DMPG/-/tree/d4be372c2261166c52bbcf5f6aa751df99cb96b7. The code used for the comparison between DMPG and SIMIO is available at https://gitlab.com/digitaltwinml/DMPG/-/blob/48ca019727978a4c7b6224845c6c90e87751f758/comparison/test_model.py.

Declarations

Competing interests The authors declare no competing interests.

References

1. Joshi S, 5 major benefits of data collection for manufacturing companies manufacturing tomorrow. 2022. https://manufacturingtomorrow.com/article/2022/08/5-major-benefits-of-data-collection-for-manufacturing-companies/19116.
2. Barbie A, Hasselbring W. From digital twins to digital twin prototypes: concepts, formalization, and applications. IEEE Access. 2024;12:75337–65. https://doi.org/10.1109/ACCESS.2024.3406510.
3. Minevich M, How to reinvent continuous improvement with intelligent digital twins in manufacturing. https://www.forbes.com/sites/markminevich/2022/01/28/how-to-reinvent-continuous-improvement-with-intelligent-digital-twins-in-manufacturing/. Accessed 21 Sep 2023.
4. Gya R, et al., Digital twins: adding intelligence to the real world'. 2022. https://www.capgemini.com/gb-en/wp-content/uploads/sites/3/2022/05/Capgemini-Research-Institute_DigitalTwins_Web.pdf
5. Lenstra JK, Rinnooy Kan AHG, Brucker P. Complexity of machine scheduling problems. In: Hammer PL, Johnson EL, Korte BH, Nemhauser GL, editors. Studies in integer programming, in annals of discrete mathematics. North-Holland: Elsevier; 1977. p. 343–62. https://doi.org/10.1016/S0167-5060(08)70743-X.
6. Mazyavkina N, Sviridov S, Ivanov S, Burnaev E. Reinforcement learning for combinatorial optimization: a survey. Comput Oper Res. 2021. https://doi.org/10.1016/j.cor.2021.105400.
7. Crites RH, Large-scale dynamic optimization using teams of reinforcement learning agents.
8. Crites RH, Barto G, 'Elevator group control using multiple reinforcement learning agents.
9. Feldkamp N, Bergmann S, Strassburger S, Simulation-Based Deep Reinforcement Learning For Modular Production Systems, in 2020 Winter Simulation Conference (WSC), Orlando, FL, USA: IEEE, Dec. 2020, pp. 1596–1607. https://doi.org/10.1109/WSC48552.2020.9384089.
10. İnal AF, Sel Ç, Aktepe A, Türker AK, Ersöz S. A multi-agent reinforcement learning approach to the dynamic job shop scheduling problem. Sustainability. 2023;15(10):8262. https://doi.org/10.3390/su15108262.
11. Panzer M, Bender B, Gronau N. Neural agent-based production planning and control: an architectural review. J Manuf Syst. 2022;65:743–66. https://doi.org/10.1016/j.jmsy.2022.10.019.
12. Esteso A, Peidro D, Mula J, Díaz-Madroñero M. Reinforcement learning applied to production planning and control. Int J Prod Res. 2023;61(16):5772–89. https://doi.org/10.1080/00207543.2022.2104180.
13. Rathore MM, Shah SA, Shukla D, Bentafat E, Bakiras S. The role of AI, machine learning, and big data in digital twinning: a systematic literature review, challenges, and opportunities. IEEE Access. 2021;9:32030–52. https://doi.org/10.1109/ACCESS.2021.3060863.
14. Ouahabi N, Chebak A, Kamach O, Laayati O, Zegrari M. Leveraging digital twin into dynamic production scheduling: A review. Robot Comput-Integr Manuf. 2024;89: 102778. https://doi.org/10.1016/j.rcim.2024.102778.
15. Zhang L, Yan Y, Hu Y, Ren W. Reinforcement learning and digital twin-based real-time scheduling method in intelligent manufacturing systems. IFAC-PapersOnLine. 2022;55(10):359–64. https://doi.org/10.1016/j.ifacol.2022.09.413.
16. Xia K, et al. A digital twin to train deep reinforcement learning agent for smart manufacturing plants: environment, interfaces and intelligence. J Manuf Syst. 2021;58:210–30. https://doi.org/10.1016/j.jmsy.2020.06.012.
17. Mueller-Zhang Z, Antonino PO, Kuhn T. Integrated planning and scheduling for customized production using digital twins and reinforcement learning. IFAC-PapersOnLine. 2021;54(1):408–13. https://doi.org/10.1016/j.ifacol.2021.08.046.
18. Kritzinger W, Karner M, Traar G, Henjes J, Sihn W. Digital twin in manufacturing: a categorical literature review and classification. IFAC-PapersOnLine. 2018;51(11):1016–22. https://doi.org/10.1016/j.ifacol.2018.08.474.
19. Xu H, Wu J, Pan Q, Guan X, Guizani M. A survey on digital twin for industrial internet of things: applications, technologies and tools. IEEE Commun Surv Tutor. 2023;25(4):2569–98. https://doi.org/10.1109/COMST.2023.3297395.
20. Eramo R, Bordeleau F, Combemale B, Brand MVD, Wimmer M, Wortmann A. Conceptualizing digital twins. IEEE Softw. 2022;39(2):39–46. https://doi.org/10.1109/MS.2021.3130755.
21. Blazewicz J, Ecker KH, Pesch E, Schmidt G, Sterna M, Weglarz J, Handbook on scheduling: from theory to applications, 2. Auflage. Springer Nature, 2019.
22. da Righi R. 'Preface', in production scheduling. InTech. 2012. https://doi.org/10.5772/1392.
23. Usuga Cadavid JP, Lamouri S, Grabot B, Pellerin R, Fortin A. Machine learning applied in production planning and control: a state-of-the-art in the era of industry 4.0. J Intell Manuf. 2020;31(6):1531–58. https://doi.org/10.1007/s10845-019-01531-7.
24. Dagkakis G, Heavey C. A review of open source discrete event simulation software for operations research. J Simul. 2016;10(3):193–206. https://doi.org/10.1057/jos.2015.9.
25. Kuhnle A, SimPyRLFab, 2020. https://github.com/AndreasKuhnle/SimRLFab.
26. Rinciog A, Meyer A, FabricatioRL-v2, GitHub repository. GitHub, 2023. https://github.com/malerinc/fabricatio-rl.git.
27. Hubbs CD, Perez HD, Sarwar O, Sahinidis NV, Grossmann IE, Wassick JM, OR-Gym: a reinforcement learning library for operations research problems. 2020.
28. 'DMPG—Digital model playground'. https://gitlab.com/digitaltwinml/DMPG
29. 'SimPy—Discrete event simulation for Python'. https://simpy.readthedocs.io/en/latest/
30. Staib T, SEP_DMPG, 2024. https://github.com/Tomstaib/SEP_DMPG.

31. DMPG Animated, 2024. https://gitlab.com/digitaltwinml/DMPG/-/tree/Animated-stable?ref_type=heads.
32. Belsare S, Badilla ED, Dehghanimohammadabadi M. Reinforcement learning with discrete event simulation: the premise, reality, and promise', in *2022* Winter Simulation Conference (WSC), Singapore: IEEE, Dec. 2022, pp. 2724–2735. https://doi.org/10.1109/WSC57314.2022.10015503.
33. TensorFlow Developers, TensorFlow. (Jul. 11, 2024). Zenodo.
34. Mnih V, et al. Human-level control through deep reinforcement learning. Nature. 2015;518(7540):529–33. https://doi.org/10.1038/nature14236.
35. 'TIOBE Index'. https://www.tiobe.com/tiobe-index/.
36. Guzman E, Andres B, Poler R. Models and algorithms for production planning, scheduling and sequencing problems: a holistic framework and a systematic review. J Ind Inf Integr. 2022;27:100287. https://doi.org/10.1016/j.jii.2021.100287.
37. Van Der Aalst W. Process mining. Berlin: Springer, Berlin Heidelberg; 2016.