Article Identifier
DOI: 10.22032/dbt.58912
URN: urn:nbn:de:gbv:ilm1-2023isc-100:4

60th ILMENAU SCIENTIFIC COLLOQUIUM
Technische Universität Ilmenau, 4 – 8 September 2023
URN: urn:nbn:de:gbv:ilm1-2023isc:1

Towards a deep reinforcement learning integration into model-based systems engineering

P. Trentsios, M. Wolf, D. Gerhard

Ruhr-Universität Bochum

ABSTRACT

The integration of Deep Reinforcement Learning (DRL) in Model-Based Systems Engineering (MBSE) is a promising approach that can lead to significant benefits for system designers and developers. DRL is a branch of machine learning where an agent learns to make decisions by interacting with an environment, receiving feedback in the form of rewards or punishments that indicate the quality of its actions, and adjusting its decision-making policy to maximize the cumulative reward over time. MBSE provides a structured approach to system design, which can help to clarify system requirements, identify potential issues, and improve the overall efficiency of the system development process. This model-based approach can be particularly useful for DRL, which requires a clear understanding of the system environment and objectives to develop the system's behavior.
We propose a method for integrating DRL into MBSE, where the desired system behavior is defined in a model-based representation using a modeling language to describe the relevant design components for DRL. The method's model framework is applied and evaluated to an example use case using SysML as the modeling language. This integration enables system designers to use DRL with the benefits and support of MBSE.

Index Terms **–** reinforcement learning, MBSE, simulation, SysML

1. INTRODUCTION

Model-based systems engineering (MBSE) has been widely used in the development of complex systems, allowing engineers to capture and analyze the system requirements, behavior, and architecture systematically. On the other hand, deep reinforcement learning (DRL) has shown great potential in enabling autonomous systems to learn and adapt to their environment without the need for explicit programming. DRL has been applied successfully in a variety of engineering fields, such as robotics and control systems [1].

The integration of DRL in MBSE can potentially provide a more efficient way of developing autonomous systems, as DRL can learn the context-relevant information processing of the system directly from simulation data generated by MBSE models in a virtual environment. By doing so, the integration of DRL in MBSE can reduce the development time and cost, as well as increase the performance and robustness of the system in changing operating conditions.

The objective of the paper is to demonstrate the feasibility and benefits of integrating DRL into MBSE, such as traceability, improved system validation and verification, reduction of development time and cost, exploitation of suistanibility and efficiency factors, support and enablement of change and variant management, and consideration of multiple stakeholders. For that, the paper investigates and addresses the challenges and limitations of the proposed approach, such as the need for domain expertise and the complexity of DRL development. The paper also presents an outlook for future research questions and hypotheses that can be addressed to evaluate the effectiveness of the proposed integration, as well as potential future directions for research and development in this area. Overall, this paper contributes to the growing research area of integrating AI into MBSE and provides insights for practitioners and researchers interested in developing intelligent systems using DRL and MBSE.

1.1 Research Questions

The following research questions clarify the focus of this paper and allow for later reflection on the results obtained:

1) How can MBSE facilitate the DRL-based development of a system's information processing unit?

2) How can system, environment, or requirement changes be addressed by integrating DRL into MBSE?

3) How can the use of DRL in MBSE be validated and tested to ensure the performance and safety of the resulting system?

To start of the discussion, the main hypothesis is, that the integration of DRL in MBSE with simulated virtual environments provides a more efficient and cost-effective way of developing autonomous systems compared to other methods such as rule-based systems or traditional model-based design, and DRL-based systems can achieve better performance in complex and dynamic environments. This preliminary hypothesis will be backed up by the following related work chapter.

2. RELATED WORK

The chapter introduces the intersection of the domains of Deep Reinforcement Learning (DRL), Model-Based Systems Engineering (MBSE), and virtual simulations. Furthermore, it explores existing research initiatives that aim to bridge these domains, highlighting the efforts made in combining their methodologies and applications. Additionally, this chapter delves into a discussion of the current challenges encountered in integrating DRL, MBSE, and virtual simulations, providing insights into the obstacles that researchers and practitioners face in this interdisciplinary field.

2.1 Deep Reinforcement Learning

An overview of DRL is given with a focus on some of the relevant design components.

2.1.1 Introduction to DRL

Reinforcement learning (RL) is a machine learning (ML) paradigm, which is assigned to the field of artificial intelligence (AI). According to Sutton and Barto RL consist of a learner also referred to as an *agent*, an *environment,* and a *goal (task)* [2]. Where subcomponents of RL are a *policy*, a *reward function*, a *value function*, and an optional *model of the environment*.

The agent can observe the environment's state, manipulate the environment through a set of possible actions, and get rewards based on the correct fulfillment of the task. The policy maps actions to states and is learned through a training process where rewards or punishments are given based on the reward function. A value function estimates the cumulative reward that an action can yield given the current state. In some RL methods, a model of the environment can be provided to the agent, which enables the planning of actions.

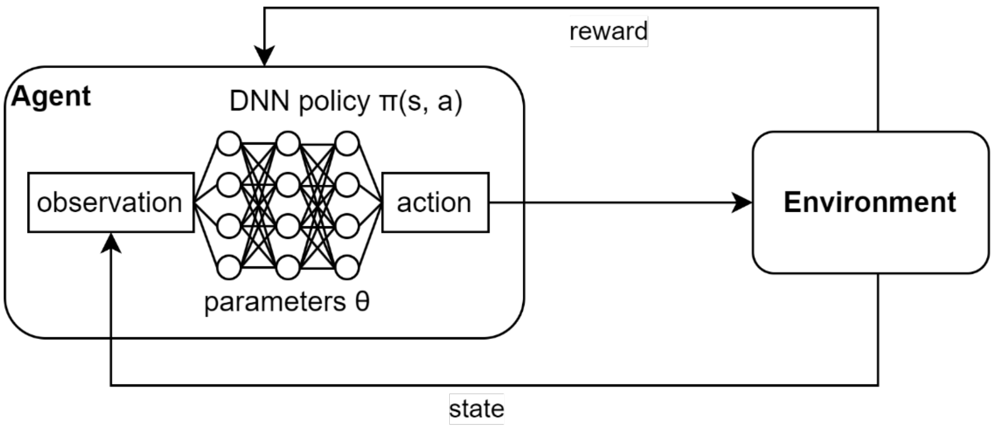

Figure 1 – The agent-environment interaction [3]

DRL is a subset of RL, where the term "deep" refers to the use of deep neural networks (DNN) to represent the policy of the agent. DNNs can act as universal function approximators, making them efficient for different fields of application. DNNs can have various structures depending on the task as well as the structure of the corresponding agent (system).
Task-related influences on the DNN can be, for example, if the task requires the agent to remember certain things, special DNN structures such as long-term short-term memory (LSTM)can be applied [4]. The complexity of the task can also influence the structure of the DNN, where complex tasks require larger DNNs with more artificial neurons and hidden layers than less complex tasks would. System-based influences of the DNN are for example the connections of the input and output layer to the sensors and actuator control of the system. Sensory inputs like a camera would also need special DNN structures like convolutional neural networks (CNN) [5].

During training, the agent-environment interaction shown in Figure 1, in which each interaction step consisting of a perceived state, an action performed, and a reward applied, is performed repeatedly. The state contains all the information of the system, the environment, and the current interaction between them. In this context, a training episode consists of a start state and a conditional end after which a new episode is then initiated. End conditions can be the successful task completion, failure of task completion, or a fixed time or step horizon. Episodes and steps are used as samples to update the parameters of the DNN or respectively the policy, in a way that the cumulative reward received by the agent is maximized. Therefore, the samples collected during the training process should provide various experiences to adequately address the post-deployment conditions of the task. The initial state of each episode can therefore be randomized to provide unique experiences.
In other ML paradigms, like supervised learning, it is common practice to have a training set and a test set of data points. However, since the agent in RL learns by interacting with a virtual representation of the actual use case, there should be no training and testing data needed. Nevertheless, some verification criteria should be defined.

There are several so-called hyperparameters that can influence the training process. For example, the greedy factor influences the agent's tradeoff between exploration and exploitation, while the discounting factor for rewards influences whether an agent is more likely to act short-sighted or long-sighted.

Updating the policy relies on the rewards applied through the reward function, which reinforces desired behavior of the system. It does so by interpreting the current states and interactions between the agent and the environment. Those can be defined as beneficial or non-beneficial. The reward function then provides the constraints for the system's desired behavior. The reward function does not represent or consider the state or action space of the system. In this sense, it does not directly affect the sensors or actuators of the system. Multiple DRL algorithms exist to update the policy and can be chosen based on various criteria, which are described by Kegyes et al. [6].
In conclusion what must be defined is the training process, the DRL method and algorithm used, the configuration of hyperparameters, the reward function, the DNN structure, and the state space.

2.1.2 RL use cases and potential in engineering
Multiple identified literature reviews on the use of reinforcement learning in engineering academia and industry suggest that (D)RL has significant potential for improving a range of engineering systems and processes, including control systems, smart manufacturing, production planning [7], allocation [8] as well as digital twins [9]. In addition, they demonstrate the diversity of applications for (D)RL. Showing that (D)RL is beneficial for the application in non-linear and non-deterministic environments [10]. As a result, (D)RL proves to be well-suited for addressing dynamic and complex tasks. Notably, numerous studies presented consistently showcase that (D)RL outperforms "traditional" techniques and algorithms, where "traditional" refers to rule-based approaches in developing the information processing unit of systems [11].

2.2 Virtual Environments
The training process of a DRL agent can be performed in the real world, but this approach has obvious limitations and drawbacks. It requires a functional physical prototype, which is problematic in the early stages of system development. There is also a risk of damaging the system or the environment. An alternative approach is to use a virtual environment for training (following referred to as simulation). The use of simulations allows the exploitation of various advantages. E.g., they allow automatic resetting of settings and application of rewards, can provide a safe way to explore new problem-solving methods without physical harm and multiple simulation instances can be run in parallel to reduce training time. However, there is a "Sim2Real gap" between simulation and reality caused by imperfect simulations or errors. Several approaches, such as Domain Randomization and Zero-Shot Transfer, aim to overcome this gap by simulating the system and the system's environment "realistically enough"[12].

2.3 Model-Based Systems Engineering
MBSE aims to combine different engineering disciplines during product development by establishing one single source of truth system model [13]. For that purpose MBSE consists of three pillars: method, language, and tool. The *method* is the actual development method applied, the *language* is used for the description of the system model and the *tool* supports the whole process.

Some of the major current modeling languages are UML, SysML, Modellica, and Capella. The Systems Modeling Language (SysML) focusses on modeling systems; for which it provides various diagram types and modeling options. It is a dialect of UML 2.0 and is currently available in version 1.6 [14]. However, a major revision of SysML, SysML v2, is soon to be released and will be independent of UML 2.0. Itwill offer multiple new modeling possibilities. The method presented in this paper mainly considers SysML v1.6, with an outlook on the possibilities of SysML v2.0.

A SysML model can consist of allocation tables as well as diagrams based on the categories requirement, structure, and behavior. With behavior and structure consisting of various graphical diagram types. The structure diagrams are used to represent the structure of the system's design and architecture as well as to structure the SysML model itself, with diagrams like *blocks*, *internal blocks*, *parametric diagrams*, and *packages*.

The system's behavior can be modeled with diagrams like *state machines*, *activities*, *sequences,* and *use cases*. Where those diagram types are typically focused on rule-based development approaches.

In terms of modeling, the environment is exclusive to the modeling of the system, e.g., the system could be a driverless transport vehicle and the corresponding environment is a virtual warehouse. When considering a subsystem of said driverless transport vehicle, the environment may be the overall system itself, e.g., the system is the battery management component of the transport vehicle, and the environment is the rest of the transport vehicle. Modeling the environment as a sole entity, independent of the system, is often not considered in SysML.

2.4 Initiatives for modeling DRL

DRL is a wide field with various aspects that have to be modeled for the training of a capable task fulfilling policy. Following the initiatives for a general approach on DRL as well as especially on the definition of a reward function, which plays a major role in the application of DRL.

REWARD FUNCTIONS

In traditional rule-based approaches, one must define the desired behavior. In RL, the constraints of what is desired and undesired must be defined, with the policy being learned be the agents. In DRL those constraints are defined through the reward function, which can be a numerical function. There are also approaches to model the reward function in existing diagram types, like reward machines [15], which are a special kind of state machines, and behavioral trees [16], or as linear temporal logic [17].

Reward machines can represent the reward function as a modified state machine, where transition conditions are based on events. The events mark crucial interactions between system parts or environment parts. Reward machines differ fromregular state machines,even though they logically function the same way. In classical state machines, the behavior of the system is directly described, whereas reward machines describe the rewards applicable for beneficial states from which the optimal behavior is synthesized in the RL training process.

INITIATIVES TO INTEGRATE AI, ML OR RL IN MBSE

In the context of MBSE Vision 2035, due attention is given to the incorporation of artificial intelligence (AI), particularly machine learning, into system modeling. The report highlights the emerging trend in which systems use AI, including machine learning techniques, to facilitate adaptability to dynamic environments and changing conditions. In addition, the interconnected nature of such systems leads to new system design challenges [13].

Some considerable initiatives to integrate AI and ML into MBSE are the following papers by: Gerschütz et al., which introduces the AI4PD ontology, to combine product development processes and data-driven processes [18]. Radler et al., introduce a task definition for data science purposes. The Authors use SysML packages to structure relevant aspects of Data Science with the SysML language [19]. A similar work introduced by Wilking et al. where a model of a ML algorithm is modeled by SysML parametric diagram and used in a digital twin [20].

INITIATIVES TO INTEGRATE RL IN PRODUCT DEVELOPMENT.

Hillebrand et al. propose a design methodology for DRL [21], which considers some of the relevant design decisions presented in **Fehler! Verweisquelle konnte nicht gefunden werden.**. The Q-Model introduced by Kurrek et al. does also describe a design methodology for DRL it does so by describing the overall process without special consideration of design decisions [22].

3. NEED FOR ACTION AND APPROACH

DRL is to be used solely for the development of a system's information processing unit, whereas a system is defined by the VDI 2206, as a cyber-physical system [23]. In this definition, the information processing unit maps sensory inputs to actuator outputs. The application of DRL is not considered for other purposes besides the designing of the information processing unit.

For the method, MBSE is considered mostly to support the development process for the information processing unit. Considering the MBSE part the focus is on the method and language pillars.

The related work presented in chapter 2 and especially the initiatives to integrate AI and ML into MBSE, as well as the initiatives to integrate RL into product development, show that either the integration of other AI aspects is modeled in MBSE or the DRL process without the modeling aspect no model-based development approach for DRL. The mentioned papers on the integration of RL into product development introduce new methods for the RL development process, which present interesting approaches to address the problem of missing standards and methods in the field. However, it becomes apparent that both methods could benefit from a holistic model, that contains all relevant DRL design decisions.

Dulac-Arnold et al. present nine challenges for applying RL in the real world, these include defining an appropriate reward function, security constraints, and explainable policies [24].
The reward function is often tailored to one specific task. In the field of systems engineering and product development, there are also only few modeling approaches and development processes that consider DRL.

To the authors knowledge, there is no method explicitly targeted to support the implementation of DRL for a systems information processing unit by utilizing MBSE approaches that also target current DRL challenges.

3.1 Approach

3.1.1 Identify Similarities

To connect the DRL domain with the MBSE domain, one must first create a basic understanding of both technologies and equate certain terms and procedures.

The agent can be considered as the CPS, whereas the policy can be considered as the information processing unit.
The meaning of state or state space can be understood differently depending on the domain under consideration. According to Sutton and Barto (fig.1), the agent has no intrinsic state in RL, as it solely observes the state of the environment, however in practice the actual observation of the agent contains only a subset of the information of the current state [25]. The observation is therefore restricted to information that the agent (system) is capable of perceiving through its sensors and will therefore be referred to as observation space. We consider additionally that the system, being a physical entity, will also have a state space, where a state can be defined as the entirety of the values of all properties of the system or the environment at a given time. Whereas some states can be understood as beneficial states based on the requirements and should be rewarded.

3.1.2 Derive Synergies

MBSE can help to structure and modularize the design components of DRL. It can make the system requirements, the fulfillment of the requirements, as well as the individual modules of the system and the environment interdependent. For the application of DRL, a virtual simulation including a digital prototype of the system as well as a virtual model of the environment is needed. Due to the MBSE approach, those models could be provided. MBSE should also include modeling languages such as action space and sensor (state) space. Meaning that the sensory input and possible actor outputs are somewhat defined. Which is beneficial and needed for DRL. The emphasis is on behavior description. DRL can help develop intelligent systems that would not be possible with classical rule-based development methods.

3.1.3 Hypotheses

Based on the related work, the identified need for action and challenges, as well as the elaboration of similarities and synergies between MBSE and DRL, we formulate the following hypotheses on the research questions defined at the beginning.

Hypothesis to research question 1: A model framework is needed that integrates DRL into MBSE with virtual simulations to learn a DNN-based policy acting as the information processing unit for a system under consideration. The framework must include aspects like, modeling the desired system behavior as well as possible system interactions with the environment, deriving the action and observation spaces from the system description, modeling the DNN and the training parameters and designing the reward function, and the consideration of using virtual simulations to train and evaluate the DRL agent. The framework must represent the mentioned aspects in a suitable modeling language.

Hypothesis to research question 2: Integrating DRL into MBSE can enable autonomous systems to dynamically adapt to changes in requirements without requiring extensive manual updates to the information processing unit. By using DRL-based information processing units in MBSE, the development process can be more flexible and responsive to changes compared to traditional rule-based design. The integration of DRL into MBSE can improve the traceability of system requirements, as the behavior of the autonomous system is more explicitly linked to the system's intended tasks as well as the possible interactions between the system and the environment. The use of DRL in MBSE can enable the creation of more robust and fault-tolerant autonomous systems, as the information processing unit can learn to adapt to unexpected changes in the system's environment or behavior. Existing components can be reused or adapted to satisfy the changes.

Hypothesis to research question 3: By using a combination of simulation-based testing and real-world validation, the performance and safety of the resulting system can be thoroughly evaluated and verified. Through the modular and model-based development approach, problems identified in the evaluation process can be targeted efficiently. Through the MBSE approach, the verification and validation process can be linked to corresponding requirements and criteria.

4. METHOD

The proposed method mainly presents the innovative model framework for integrating DRL into MBSE. In addition, the ability of the presented model to consider change and variant management, human factor, and sustainability and efficiency aspects is presented.
The person applying the method can be considered as a developer of DRL-based information processing. The development process itself is not further described. In previous work, the authors have already published a process for the development of DRL-based information processing [Blind] that can be supported by the presented model. However, the development process would need to be adapted.

4.1 Model Description

The presented model can be considered as a supplement to an existing system model.
For the development of the DRL model, we identified the relevant design elements that need to be modeled based on the related work and the results of the need for action and approach, which are shown as SysML packages in Figure 2. The package shows the overview of the structure of the model framework. The contents of the packages and the connections between them are not included in the figure, but are discussed in the description of each package.
The modeling procedure applied in this chapter shall provide a general understanding of the presented model framework, while the content of the presented modeling aspects is considered in the exemplary use case introduced in chapter 5. The criteria for selecting appropriate modeling parameters will not be explained further, as the focus is on modeling.

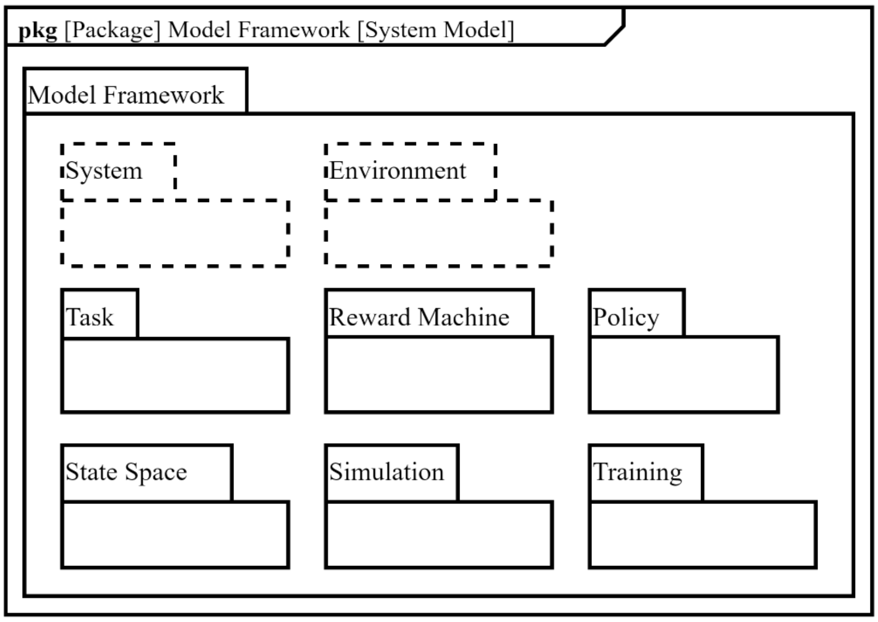

Figure 2 – Package overview of the model framework

4.1.1 System & Environment

The system and environment are not designed by the presented approach, excluding the systems information processing unit, however many of the included contents are needed to model the contents of the other packages. Including the description of system sensors and actuators as well as their interface types, which are relevant to the DNN structure. Also, system and environment architecture models with functional and structural models, that are relevant for the simulation.

4.1.2 Task

The system task definition can be derived from existing requirements or use cases. They can be modeled with requirements and requirements diagrams that describe the task and identify the relevant components of the system and environment. The task description should also consider which relationships between system parts, environment parts, or requirements are relevant for verification of the requirements. This can be modeled as a constraint block, where the associated components can be modeled as parameters and the constraint formula represents the relationship between these components. The fulfillment of a constraint can be considered as an event or signal that is used in other model elements.
In SysML v2, requirements will have additional description capabilities as well as built-in verification constraints that could benefit the task modeling.

4.1.3 Reward Machine

We propose to integrate the reward function as a reward machine modeled as a state machine. The states of the reward machines can be derived from the task constraints and are only a subset of the state spaces of the system and environment. State transitions can utilize the events and signals defined by the task constraints. The application of rewards is done inside states, with the intensity of the reward being associated with the priority of the task constraint. The conditional terminations of training episodes can be modeled inside the corresponding state. Instead of one holistic reward machine that, in case of changes, must be constantly adapted and can be in a superposition of multiple states we propose to use dedicated reward machines for the individual (sub)tasks. Additionally, reward machines in the form of HFSM could be used to model a task including a subtask, modeled as a state containing a nested state machine. For use in other parts of the model, the reward machines created using a state machine (STM) can be applied to a block. E.g., this block can be used to verify the associated task requirement.

The advantages of the reward machines are, that they are in a sense a reward function and visualize all the beneficial states, the rewards applied in those states as well as the condition or events that trigger state transitions. A reward machine can describe the fulfillment of a task. that not only describe the desired states but also describe the conditions that lead to those states. Also in comparison to the other reward function models presented in 2.4, reward functions are in a sense modified STM, which are included in SysML and many other modeling languages.

4.1.4 State Space

The state space is a definition of the possible state spaces of the system (system parts) and the environment (environment parts). It defines every component of the system and the environment as well as their attributes and the initial attribute value. Those attributes should be provided by the system and environment model. Additionally training specific attributes are needed, that are not part of the modeling aspect of the system or the environment. (e.g., current system position in relation to the environment, current velocity, temperature, etc.).
This definition of the state space can be realized with a block diagram.

The development-specific characteristics can be associated with the according system and environment elements, where the use phase-specific characteristics could be initially defined in the state space.
The state space is used to describe how the initial state of the environment and the system should be configured or reconfigured during a training episode. We consider using various blocks that describe different configurations of states, that are in a generalization association to the state space block. The rules defined for those states could be described with parametric diagrams and be attached to the configuration blocks.
The modeling of the state space could benefit from the snapshot functions introduced in SysML v2, where multiple "snapshots" of a block containing different attribute values can be defined.

4.1.5 Policy
For the modeling of the policy, we propose to model the structure of the DNN that represents the policy to be trained, as well as existing policies that can be adapted.

DNN Structure
The various DNN structure parameters used and their values can be modeled as blocks, with the individual blocks accounting for system-specific and task-specific DNN structure elements. The system-specific parameters can be derived from the sensory and actuator interfaces of the system, which can be modeled as input and output layers of the DNN. Additionally, the type of perceived data or the type of data provided to the actuators can be taken into account by using matching DNN structures, e.g., the use of CNNs for visual data.
The task-specific parameters are closely dependent on the task of the system. E.g.: The complexity of the task, which can influence the number of hidden layers, the number of neurons per hidden layer, and the type of activation function needed for the DNN, to approximate a task-solving function. The use of stacked observations to consider recently passed states. The use of recurrent neural networks includes a sort of feedback loop. The use of LSTMs if the existence of memory is relevant to solve the task.

Existing Policies
Existing policies can be modeled as a block instance containing all the information of all model elements and property values used to train them. Existing policies can also be used to train policy variants. Policies can be connected to one or multiple reward machines, that are used to train them.

4.1.6 Training
The Training model consists of three parts *DRL method and algorithm*, *training parameters*, and *training process*.

DRL Method and Algorithm
The DRL method and algorithm used to update the policy can be represented as a block. A parametric diagram can be used to represent the algorithmic details but is not further considered, since the modeling of the algorithm should be modeled in an appropriate tool used for the actual training process.

Training parameters
The training parameters can be influenced by system-, task- and DRL-method-specific conditions, whereas an individual training parameter can be influenced by multiple conditions. The individual training parameters can be modeled as blocks, with associations to the influencing element of other models. Examples of considerable training parameters are the

learning rate, the discount factor, the exploration rate, the number of steps, of training episodes, and the sample batch size used to update the policy.

Training process

The training process can be modeled as an activity diagram containing multiple actions for the individual training steps. Where action can either be seen as a "normal" DRL training of the policy or as another training technique, like imitation learning referring to corresponding stakeholders modeled as SysML actors, or even for the evaluation process of a completely trained policy. The individual actions can have multiple object nodes as inputs, containing the block for the policy to be trained, which itself consists of the DNN structure and the associated reward machine(s), and containing a block for the state space configuration used. It can be considered modeling the policy additionally as an object node connected to an output of the training action, due to the policy being constantly adapted during the training process. Since the goal of the training is to learn a task-solving policy this condition must be monitored, which can be done using a decision node. Due to many different problems, however, this goal is never reached. The guards used could therefore consider the reach of a defined reward threshold, that marks the fulfillment of the task as well as a defined abort condition that should warn the developer, that the policy training has failed or staggered, which can be done by a defined training episode limit reached. As stated, before the evaluation of the trained policy can also be included in the training process like the already describe modeling approach, but without further updating the policy. In this context, the association of verification requirements can also be considered.

4.1.7 Simulation

A model of the simulation is needed for the training process, the emphasis is not on the creation of the actual simulation model in the sense of creating e.g., physical, visual, mathematical, and electronic models, but rather on the definition of its architecture and the definition of what existing system and environment (simulation) models must be used and how they should be combined. The simulation in a sense therefore combines every mentioned aspect of the proposed model. It must be able to represent every possible state of the systems and environment state space, which as stated can be influenced by the training process. Further, it should be able to also perceive the current state, which is needed for the reward machine to apply the appropriate reward. Rewards are in that sense not assigned by the environment. It is the interaction between systems and environments, that can lead to a beneficial state. The simulation can be modeled as a block diagram.

If a needed simulation component is not available or if it is not accurate enough, the presented model can describe to the stakeholders what system, and environment parts are relevant and need to be (re)modeled.

The number of simulation instances used during training as well as the time scale of the simulation can also be modeled.

4.2 Change-and Variant-Management

The change and variant management are supported through the modular and model-based design, where changes can be traced due to the associations between the individual model elements. The major advantage is, that the initial configuration can be maintained with only the need to adapt the model elements influenced. A change or variant can be clustered into the categories of *task*, *system*, or *environment* where the focus is most likely on the product and its use phase. Changes and variants can also be considered for the actual development purpose itself, since many modeling and design choices rely on experience, the development process has room for improvement considering sustainability and efficiency factors. The severity of a

change or variant compared to the initially developed model influences if a policy must be retrained from scratch or if it can be adapted. Also, major changes to the DNN structure, like the change of a systems sensor or actuator interface can cause the retraining of a policy. Changes and variants can be modeled as instances of existing models.
The modeling of changes and variants is more strongly considered in SysML v2 with special model elements.

4.3 Sustainability and Efficiency

Sustainability and efficiency can be seen from the viewpoint of the policy in terms of task completion and the development process of the policy itself.
The policy is mainly dependent on the defined requirements. However certain efficiency parameters could also be defined without compromising the quality of the task completion and should be considered by the DRL developer.
Considering the development process of the policy, which in a sense is based on the model representation, the influence of the DRL developer is much higher. Making the overall development process more sustainable and efficient by avoiding training resources in the form of computational time or being more sample efficient, which means that the agent needs fewer samples to learn a valid policy, should be considered and practiced. Herby maintains the same policy quality by reducing cost and time.
Tuning of parameters can be considered to improve training results and reduce training time.
As already stated in the change and variant management chapter, the reuse of model elements also allows and supports sustainability in the development process.

4.4 Human factor

The human factor is often not considered in DRL [26], while it is much more common and researched in the realm of product development and MBSE [27]. Human stakeholders identified in the proposed model, in addition to the DRL developer, can be experts in the development phase, e.g., experts who can be consulted during training for demonstration purposes to teach the agent in terms of imitation learning. Collaborators who work with the system in the product use phase to accomplish a common task and who can be consulted for validation purposes. Simulation expertscan provide or update simulation models needed for the training purpose. The consideration of stakeholders can be additionally described in the proposed model.

5. IMPLEMENTATION

For the implementation of the proposed model framework, we use SysML v1.6 as the modeling language. We consider a dynamic exemplary use case for the application of the presented method. Dynamics in this case refers to the changes and variants of the system, environment, and task during the development and deployment process. The use case is a transport task of an autonomous transport robot system in an environment containing a package and a delivery zone, where the initial task is to deliver the package to the delivery zone. During the delivery, a certain speed should not be exceeded. There is also a variant of the environment that contains obstacles such as walls and boxes. An additional task is to avoid the obstacles. The system must avoid collisions with crates and should avoid collisions with the wall. There are two variants of the system, which are built differently. In particular, the locomotion and package handling differ significantly. One system variant uses differential locomotion and a forklift for package handling, while the other variant uses skid control and a gripper. Both the use case and the models presented are intentionally minimalist and simplified, as the focus is on demonstrating the use of the model framework.

5.1 Model Framework applied to Use Case

Following the the exemplary use case is modeled using the proposed model framework.

5.1.1 System & Environment

Block definition diagrams are used for the description of the system (see Figure 3) and environment (see Figure 4) as a high-level overview of the system and environment components and their relationships. For the autonomous transport system using DRL, the diagram includes the following blocks:

The System: The block represents the system. Including variants of the system, SysML v2 will have special considerations for system variants.

> **Sensors:** The block represents the various sensors used to perceive the environment, such as cameras, lidars, and radars.
>
> **Actuators:** The block represents the various actuators used to control the vehicle, such as the accelerator, brakes, and steering wheel.
>
> **Information processing unit:** The block represents the information processing of the system with a block representing the trained policy satisfying a certain requirement/task.

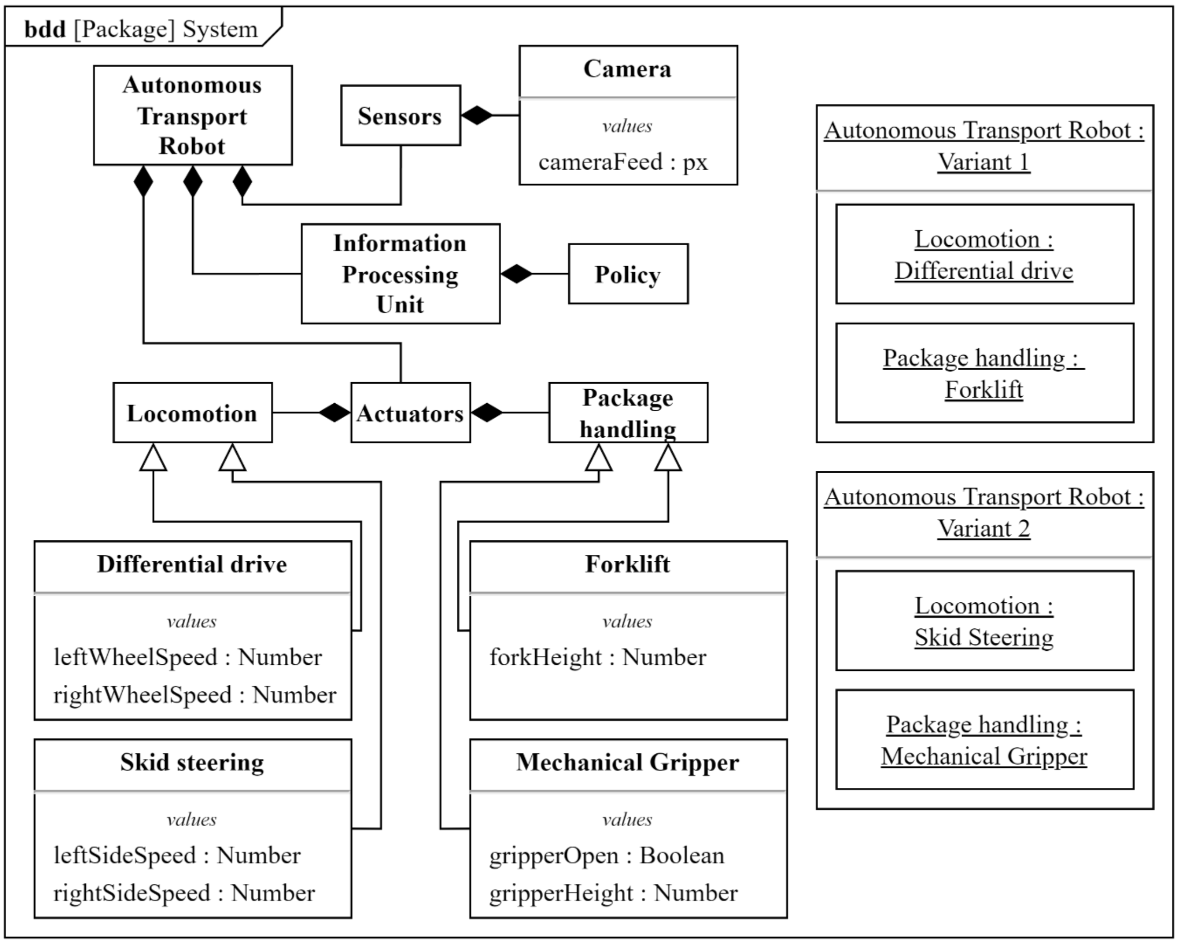

Figure 3 – Block diagram of the system including system variant instances

Environment: The block represents the environment in which the system operates (see Figure 4). This includes the delivery zone, the package to be delivered and the type of obstacles, which are boxes and walls.

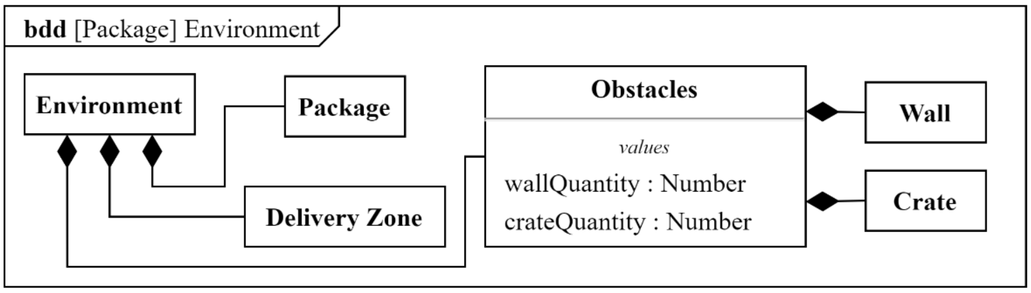

Figure 4 – Block diagram of the environment

5.1.2 Task

The task description of the system, consisting of the two tasks package delivery and obstacle avoidance, is represented as a requirements diagram (see Figure 5). The package delivery task also includes a subtask, namely that the system should not exceed a speed limit during delivery. The tasks are described by requirements that are further refinied with other requirements as well as constraint blocks. The constraint for the speed limit is described mathematically, while the other constraints have a semantic description. The constraint blocks are named after the events they trigger.

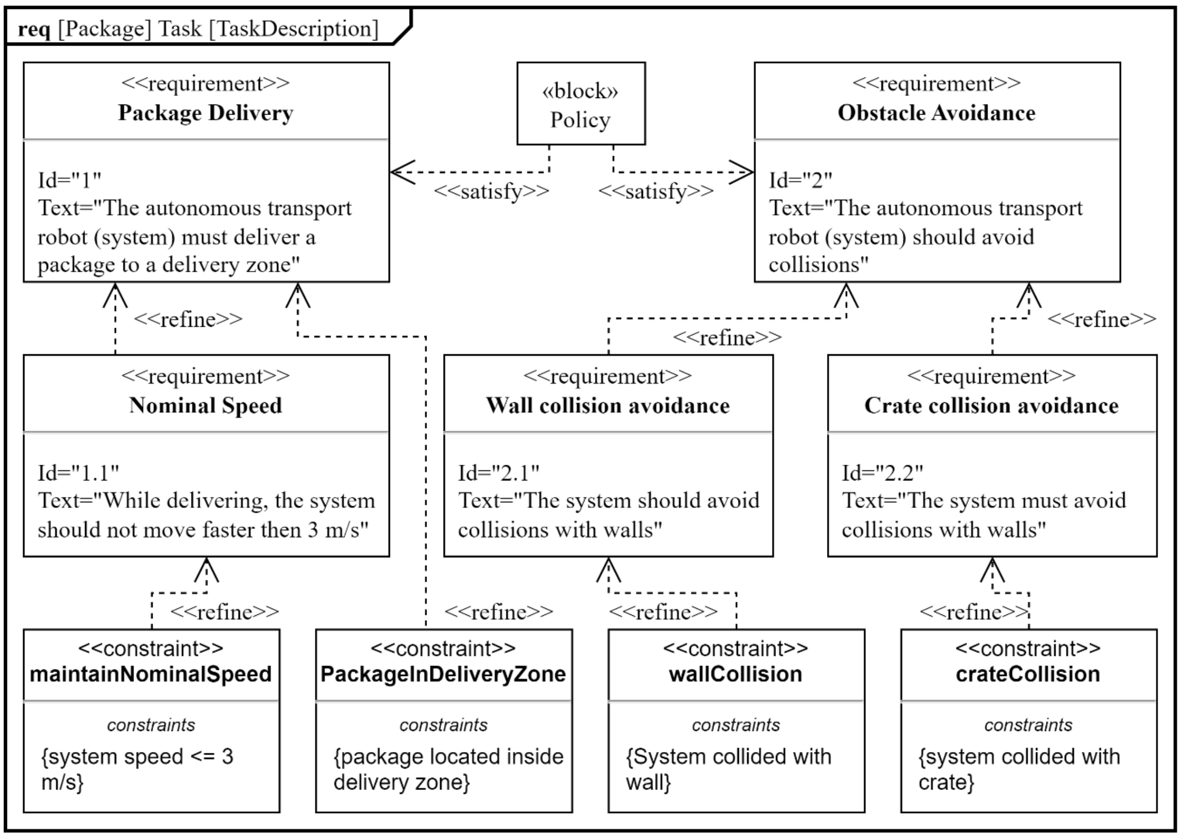
req [Package] Task [TaskDescription]
<<requirement>>
Package Delivery
Id="1"
Text="The autonomous transport robot (system) must deliver a package to a delivery zone"
«block»
Policy
<<satisfy>>
<<satisfy>>
<<requirement>>
Obstacle Avoidance
Id="2"
Text="The autonomous transport robot (system) should avoid collisions"
<<refine>>
<<refine>>
<<refine>>
<<requirement>>
Nominal Speed
Id="1.1"
Text="While delivering, the system should not move faster then 3 m/s"
<<requirement>>
Wall collision avoidance
Id="2.1"
Text="The system should avoid collisions with walls"
<<requirement>>
Crate collision avoidance
Id="2.2"
Text="The system must avoid collisions with walls"
<<refine>>
<<refine>>
<<refine>>
<<refine>>
<<constraint>>
maintainNominalSpeed
constraints
{system speed <= 3 m/s}
<<constraint>>
PackageInDeliveryZone
constraints
{package located inside delivery zone}
<<constraint>>
wallCollision
constraints
{System collided with wall}
<<constraint>>
crateCollision
constraints
{system collided with crate}

Figure 5 – Requirements diagram representing the different tasks and their constraints

5.1.3 Reward machine

A state machine diagram is used to epresent the reward machines used (see Figure 6). A separate reward machine is used for the two tasks of package delivery and obstacle avoidance. Each reward machine considers its own task independently of the other. The reward machine for the Package Delivery task consists of the Delivery and Delivered states, with the Delivered state being the starting state and applying small negative rewards. From the Delivery state, the PackageInDeliveryZone transition guard can lead to the Delivery state, which applies a large

positive reward and also triggers the end of the current episode. The Delivery state contains a nested state machine for the target speed control subtask.
The obstacle avoidance reward automaton has three states, Collision Avoidance, Collision with Wall, Collision with Crate. The different priorities of the obstacle avoidance subtasks are taken into account here by the amount of reward as well as the end of an episode. Collisions with crates that need to be avoided therefore have a larger negative reward and the end of the episode, while collisions with wall that should be avoided have a smaller negative reward and the episode continues. Both reward machines are used to train the same strategy.

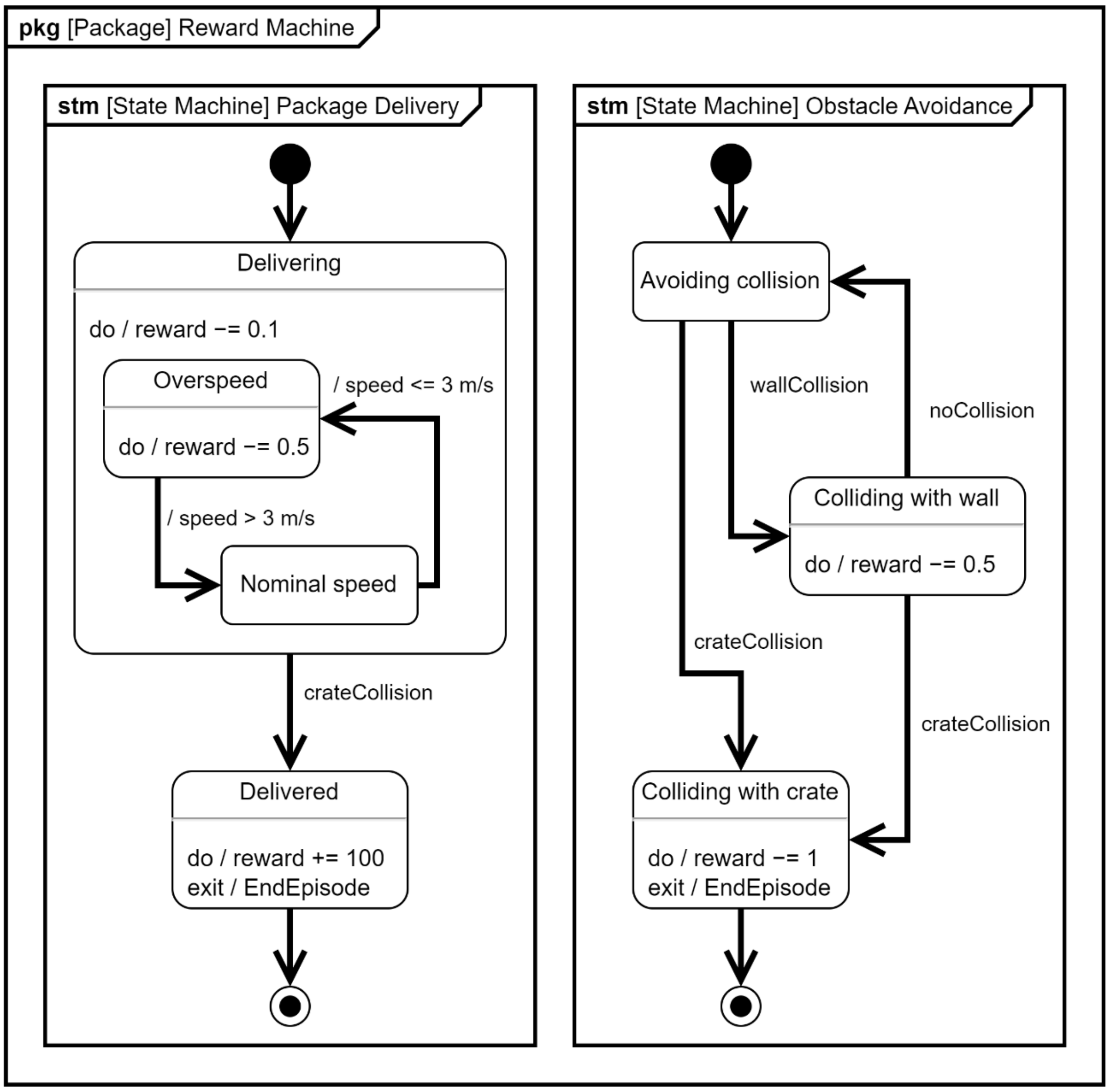

Figure 6 – State machine diagram representing the reward machines

5.1.4 State Space

The state space block diagram (see Figure 7) shows the relevant system and environment parts that can be varied and their possible range of values. The obstacle set describes how many obstacles should be placed in the environment during a training episode. The system, package, and delivery zone position values are represented as a three-dimensional vector representing the initial x, y, and z positions of the components. Three configurations are created as generalizations of the state space, with each configuration redefining the values of the state space differently. Additional parametric diagrams are not used to further describe the state space configurations.

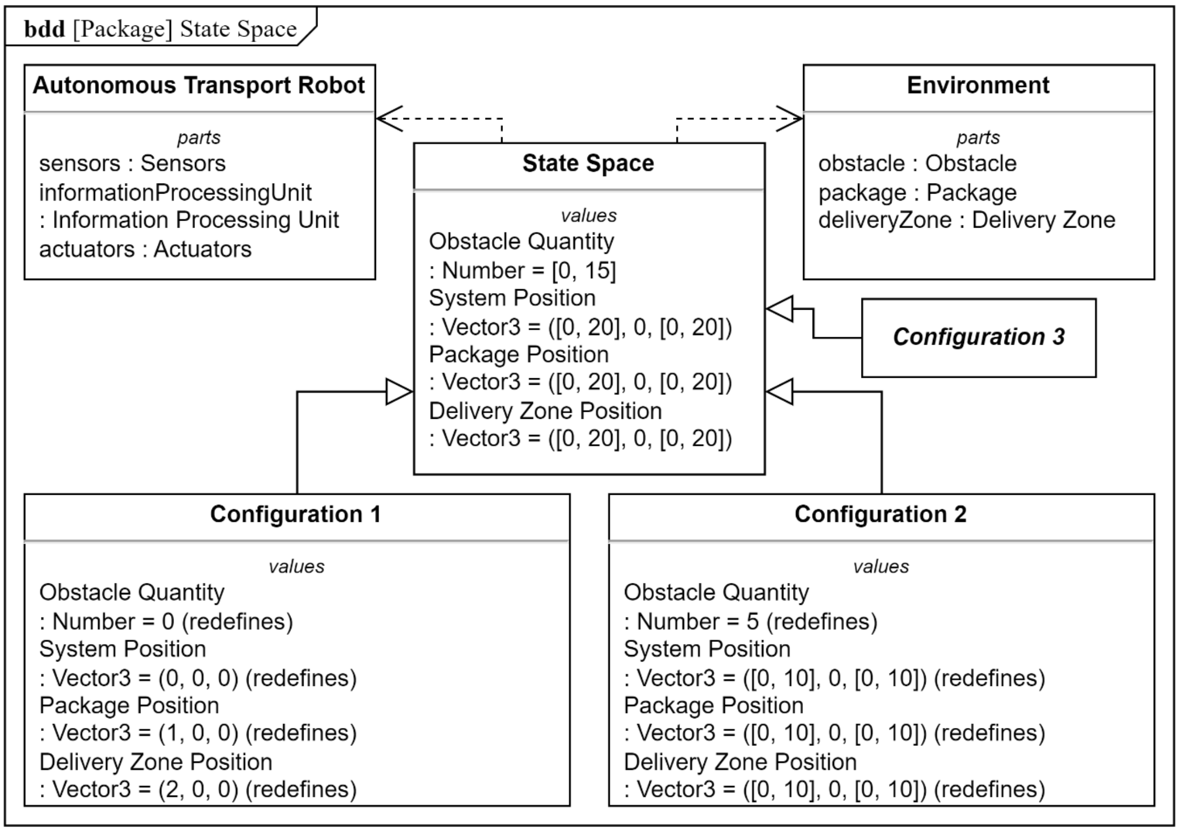
bdd [Package] State Space
Autonomous Transport Robot
parts
sensors : Sensors
informationProcessingUnit
: Information Processing Unit
actuators : Actuators
State Space
values
Obstacle Quantity
: Number = [0, 15]
System Position
: Vector3 = ([0, 20], 0, [0, 20])
Package Position
: Vector3 = ([0, 20], 0, [0, 20])
Delivery Zone Position
: Vector3 = ([0, 20], 0, [0, 20])
Environment
parts
obstacle : Obstacle
package : Package
deliveryZone : Delivery Zone
Configuration 3
Configuration 1
values
Obstacle Quantity
: Number = 0 (redefines)
System Position
: Vector3 = (0, 0, 0) (redefines)
Package Position
: Vector3 = (1, 0, 0) (redefines)
Delivery Zone Position
: Vector3 = (2, 0, 0) (redefines)
Configuration 2
values
Obstacle Quantity
: Number = 5 (redefines)
System Position
: Vector3 = ([0, 10], 0, [0, 10]) (redefines)
Package Position
: Vector3 = ([0, 10], 0, [0, 10]) (redefines)
Delivery Zone Position
: Vector3 = ([0, 10], 0, [0, 10]) (redefines)

Figure 7 – Block diagram of the state space

5.1.5 Policy

The policy is represented in a block diagram (see Figure 8). The reward machines used to train the policy are linked to it via a dependency. The system-specific parts of the DNN are linked to the first variant of the system. The task-specific parts are associated with the task package. The input layer is linked to the camera block and the output layer is linked to the differential drive and forklift blocks. The individual parts and values of the presented blocks are not shown in the figure.

5.1.6 Training

The training process is represented in the form of an activity diagram (see Figure 9). The training process uses a curriculum to break down the complexity of the task into smaller, easier lessons. The curriculum uses three progressively more difficult lessons represented as individual actions in which training occurs. Each training action has the current strategy and a particular state space configuration as its object input and the updated strategy as its object output. The three state space configurations are used to represent the lessons of the curriculum. The first configuration contains no obstacles and places the system, package, and delivery close together, making task completion much more likely. It allows for early rewards and "guides" the system through the task. The second and third configurations have a larger random range for the initial positions of the parts and introduce obstacles. Decision nodes observe whether the expected reward threshold is reached during a training action, and if so, proceed to the next training action. Otherwise, another decision node observes whether an episode threshold has been reached, in which case the training is marked as failed. When all three training actions have reached the reward threshold, a verification action takes place, using the third configuration and policy as inputs, without further training. Finally, a decision node checks whether the verification has reached a reward threshold and was therefore successful or whether the verification has failed.

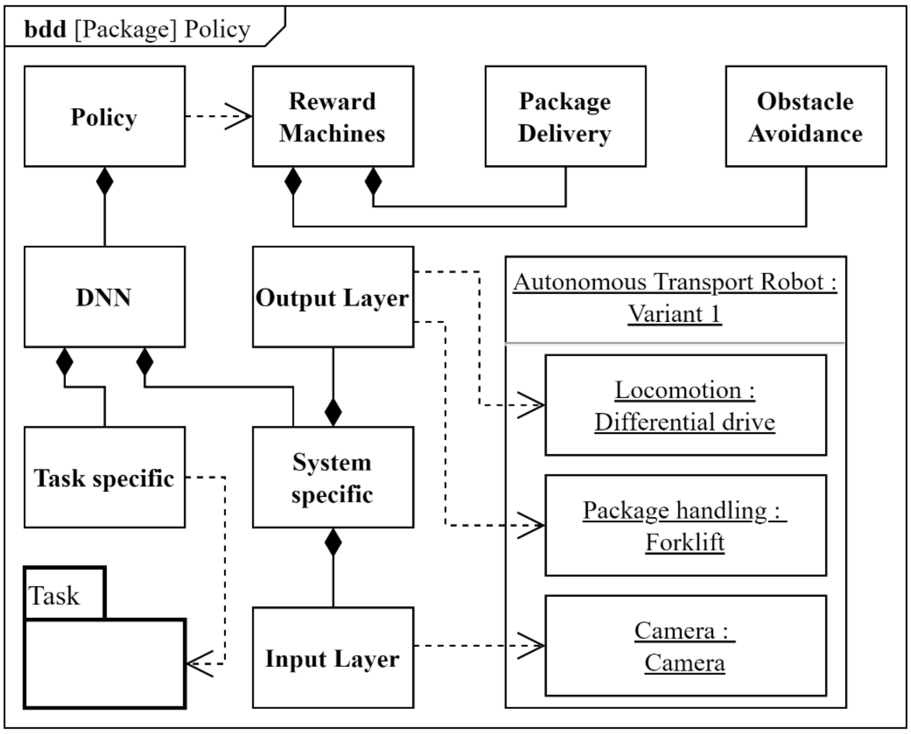

Figure 8 – Block diagram representing the policy

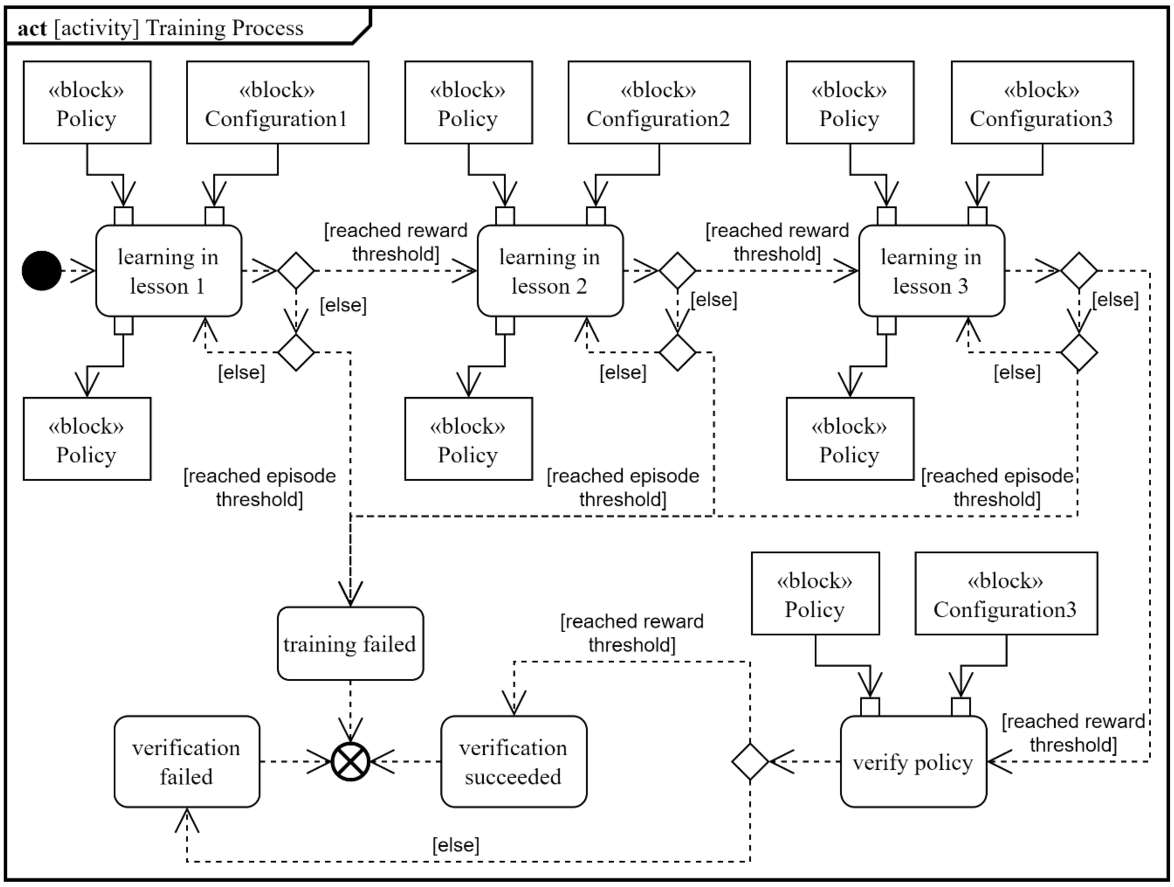

Figure 9 – Activity diagram representing the training process

5.2 Variant management

The second variant of the autonomous transport system, which uses different actuator interfaces for locomotion and package handling, is considered to demonstrate the possibility of variant management. Since the training works for the first system variant and it can be safely demonstrated that the learned strategy can accomplish the task, it can be assumed that the training process should also be applicable for the second system variant. It should be noted that the system architecture has changed. While the camera sensor has remained the same in terms of inputs, its position on the transport system has changed. In addition, the actuators have also changed. However, the task remains the same. Therefore, any design element associated with the task can remain unchanged in the model. The DNN parts specified by the system must be adjusted, especially the output layer. The training process can be applied again to train a new policy instance designed for the second system variant.

6. DISCUSSION

The model framework has been applied to an exemplary use case, on which the established hypotheses were confirmed, without compromising the design language.The proposed model framework represents a possible approach for modeling DRL-specific design elements using the SysML v1.6 modeling language.

6.1 Advantages

The advantages and novelty of the method are the ability to train a powerful system behavior based on DRL and to adapt, test, and evaluate it through the capabilities of the MBSE approach and the use of simulations. In conclusion, by using the proposed method, it becomes possible to develop more efficient, effective, and adaptable systems that meet the needs of users and stakeholders.

The presented framework addresses the challenges of reward function description, by strictly modeling the reward function to the task, which is derived from the system requirements. Furthermore, it provides extensive documentation about the modeling elements and training processes. Thereby addressing the mentioned challenges of security constraints and explainable policies.

6.2 Disadvantages

The model does not support all modeling techniques and diagram types available in SysML v1.6, like allocation tables, internal block definition diagrams, sequence diagrams, parametric diagrams as well as custom stereotypes. All diagram types that are behavioral diagrams, such as state machines and activity diagrams, must somehow have their behavior implemented in the actual simulation and training environment. If the SysML model is not directly connected to the simulation and is only used as a static model representation, there is an additional implementation overhead.

There is no dedicated modeling approach in SysML v1.6 for modeling events and interactions as presented and needed for DRL. Both sequence diagrams and parametric diagrams can be used to visualize the possible events or interactions between blocks or values of blocks. However, the focus of sequence diagrams is mostly on the exchange of information, while parametric diagrams concentrate more on the modeling of mathematical functions.

Lastly depending on the application domain, adaptation of the proposed model might be required.

7. OUTLOOK

Future work could benefit from an exportable SysML model to the actual simulation environment or an executable SysML model associated with the simulation.
This could leverage the power of an automation mechanism to generate the desired policy.
It may be possible to further extend the modeling languages with custom diagram types more suitable for use with DRL and to refine and extend the modeling framework with further consideration of all SysML functionalities. For further maturation of the presented model framework, application to various real-world use cases should be considered. Because of the future release of SysML v2, the presented method should be considered among the new possibilities arising from the significant change in the modeling language.

REFERENCES

[1] T. Rupprecht and Y. Wang, "A survey for deep reinforcement learning in markovian cyber–physical systems: Common problems and solutions," *Neural Networks*, vol. 153, pp. 13–36, Sep. 2022, doi: 10.1016/j.neunet.2022.05.013.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. in Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018.

[3] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource Management with Deep Reinforcement Learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, Atlanta GA USA: ACM, Nov. 2016, pp. 50–56. doi: 10.1145/3005745.3005750.

[4] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[6] T. Kegyes, Z. Süle, and J. Abonyi, "The Applicability of Reinforcement Learning Methods in the Development of Industry 4.0 Applications," *Complexity*, vol. 2021, pp. 1–31, Nov. 2021, doi: 10.1155/2021/7179374.

[7] M. Panzer, B. Bender, and N. Gronau, "Deep Reinforcement Learning In Production Planning And Control: A Systematic Literature Review," 2021, doi: 10.15488/11238.

[8] R. de R. Faria, B. D. O. Capron, A. R. Secchi, and M. B. de Souza, "Where Reinforcement Learning Meets Process Control: Review and Guidelines," *Processes*, vol. 10, no. 11, p. 2311, Nov. 2022, doi: 10.3390/pr10112311.

[9] A. Mazumder *et al.*, "Towards next generation digital twin in robotics: Trends, scopes, challenges, and future," *Heliyon*, vol. 9, no. 2, p. e13359, Feb. 2023, doi: 10.1016/j.heliyon.2023.e13359.

[10] V. Zambaldi *et al.*, "Relational Deep Reinforcement Learning." arXiv, Jun. 28, 2018. Accessed: Jun. 30, 2023. [Online]. Available: http://arxiv.org/abs/1806.01830

[11] C. Li, P. Zheng, Y. Yin, B. Wang, and L. Wang, "Deep reinforcement learning in smart manufacturing: A review and prospects," *CIRP Journal of Manufacturing Science and Technology*, vol. 40, pp. 75–101, Feb. 2023, doi: 10.1016/j.cirpj.2022.11.003.

[12] A. del Real Torres, D. S. Andreiana, Á. Ojeda Roldán, A. Hernández Bustos, and L. E. Acevedo Galicia, "A Review of Deep Reinforcement Learning Approaches for Smart Manufacturing in Industry 4.0 and 5.0 Framework," *Applied Sciences*, vol. 12, no. 23, p. 12377, Dec. 2022, doi: 10.3390/app122312377.

[13] International Council on Systems Engineering, "Systems Engineering Vision 2035".

[14] "OMG Systems Modeling Language (OMG SysML^{TM})."

[15] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith, "Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., in Proceedings of Machine Learning Research, vol. 80. PMLR, Jul. 2018, pp. 2107–2116. [Online]. Available: https://proceedings.mlr.press/v80/icarte18a.html
[16] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. 2018. doi: 10.1201/9780429489105.
[17] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, Oct. 1977, pp. 46–57. doi: 10.1109/SFCS.1977.32.
[18] B. Gerschütz, S. Goetz, and S. Wartzack, "AI4PD—Towards a Standardized Interconnection of Artificial Intelligence Methods with Product Development Processes," *Applied Sciences*, vol. 13, no. 5, p. 3002, Feb. 2023, doi: 10.3390/app13053002.
[19] S. Radler, E. Rigger, J. Mangler, and S. Rinderle-Ma, "Integration of Machine Learning Task Definition in Model-Based Systems Engineering using SysML," in *2022 IEEE 20th International Conference on Industrial Informatics (INDIN)*, Perth, Australia: IEEE, Jul. 2022, pp. 546–551. doi: 10.1109/INDIN51773.2022.9976107.
[20] F. Wilking, C. Sauer, B. Schleich, and S. Wartzack, "Integrating Machine Learning in Digital Twins by utilizing SysML System Models," in *2022 17th Annual System of Systems Engineering Conference (SOSE)*, Rochester, NY, USA: IEEE, Jun. 2022, pp. 297–302. doi: 10.1109/SOSE55472.2022.9812700.
[21] M. Hillebrand, M. Lakhani, and R. Dumitrescu, "A design methodology for deep reinforcement learning in autonomous systems," *Procedia Manufacturing*, vol. 52, pp. 266–271, 2020, doi: 10.1016/j.promfg.2020.11.044.
[22] P. Kurrek, F. Zoghlami, M. Jocas, M. Stoelen, and V. Salehi, "Q-Model: An Artificial Intelligence Based Methodology for the Development of Autonomous Robots," *Journal of Computing and Information Science in Engineering*, vol. 20, no. 6, p. 061006, Dec. 2020, doi: 10.1115/1.4046992.
[23] VDI2206, *Development of mechatronic and cyber-physical systems*. VDI - Verein Deutscher Ingenieure, 2021.
[24] G. Dulac-Arnold *et al.*, "Challenges of real-world reinforcement learning: definitions, benchmarks and analysis," *Mach Learn*, vol. 110, no. 9, pp. 2419–2468, Sep. 2021, doi: 10.1007/s10994-021-05961-4.
[25] D. Silver, "Lectures on Reinforcement Learning." 2015.
[26] X. Liu, H. Xu, W. Liao, and W. Yu, "Reinforcement Learning for Cyber-Physical Systems," in *2019 IEEE International Conference on Industrial Internet (ICII)*, Nov. 2019, pp. 318–327. doi: 10.1109/ICII.2019.00063.
[27] I. Gräßler, D. Wiechel, and D. Roesmann, "Integrating human factors in the model based development of cyber-physical production systems," *Procedia CIRP*, vol. 100, pp. 518–523, 2021, doi: 10.1016/j.procir.2021.05.113.

CONTACTS

Pascalis Trentsios, M.Sc. email: pascalis.trentsios@rub.de
ORCID: https://orcid.org/0000-0002-9557-8731

Dr.-Ing. Mario Wolf email: mario.wolf@rub.de
ORCID: https://orcid.org/0000-0002-0628-7570

Univ.-Prof. Dr.-Ing. Detlef Gerhard email: detlef.gerhard@rub.de
ORCID: https://orcid.org/0000-0002-3266-7526